UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Thesis presented in partial fulfillment
of the requirements for the Degree of
Ph.D. in Computing

Logic Decomposition and Adaptive Clocking for the Optimization of Digital Circuits

Lucas Machado

Advisor: Jordi Cortadella Fortuny
Computer Science Department
Universitat Politècnica de Catalunya

Barcelona, January 2019

Abstract

Over the course of 60 years, since the invention of the integrated circuit (IC), exponential improvements in cost, performance and power consumption were observed. Such advances have been strongly linked with the continuous reduction of the dimensions in manufactured ICs, but this trend has shown decreasing benefits as fundamental limits are reached.

Notice that such tiny devices have increased variability, which generates unpredictable variations in the behavior of the manufactured devices. These uncertainties are typically addressed by defining margins on the clock period, estimated during the design phase. However, the overly conservative margins produce significant degradations in performance.

Additionally, the evolution that enabled circuits with increasingly higher density of components, also resulted in an extremely complex IC design. At every step, electronic design automation (EDA) tools are challenged to handle this increasing complexity, requiring more powerful techniques to comply with the specification constraints within an affordable runtime.

This thesis investigates alternatives in order to improve power, performance, area, and cost, using established IC manufacturing technologies. Advances in EDA are proposed in three distinct topics: area minimization using Boolean methods, area and delay reduction for designs based on field-programmable gate array (FPGA), and an alternative clocking scheme to reduce timing margins.

The first contribution consists of a technology-independent method for area minimization of combinational logic. Local optimization is applied on and-inverter graphs (AIGs), performing multi-output Boolean decomposition using two-literal divisors, targeting node count reduction.

The second contribution regards two methods targeting technology mapping of FPGAs. On one hand, a functional decomposition approach, which uses the support size as cost function, exploring the inherent characteristics of FPGAs. On the other hand, an approach for recursive remapping, which reduces the structural bias of the subject graph, uses the mapping results as cost function, and obtains significant reductions in area and delay.

The third contribution evaluates the dynamic variability mitigation and simplification of power delivery networks (PDNs) using an adaptive clocking scheme based on ring oscillator clocks (ROCs). The impact of the PDN parameters and ROC location is investigated, showing potential improvements in performance, leakage power and cost.

The contributions of this thesis have been published in the following papers:

- Lucas Machado, Antoni Roca, Jordi Cortadella. Increasing the Robustness of Digital Circuits with Ring Oscillator Clocks. In *Proceedings of the International Workshop on Resiliency in Embedded Electronic Systems* (REES), pages 29-34, March 2017.

- Lucas Machado, Jordi Cortadella. Boolean Decomposition for AIG Optimization. In *Proceedings of ACM Great Lakes Symposium on VLSI* (GLSVLSI), pages 143-148, May 2017.

- Lucas Machado, Antoni Roca, Jordi Cortadella. Voltage Noise Analysis with Ring Oscillator Clocks. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI* (ISVLSI), pages 1-6, July 2017.

- Lucas Machado, Jordi Cortadella. Support-Reducing Functional Decomposition for FPGA Technology Mapping. In *Proceedings of International Workshop on Logic & Synthesis* (IWLS), pages 79-86, 2018.

Also, extensions of the published papers have been submitted to journals:

- Lucas Machado, Jordi Cortadella. Support-Reducing Decomposition for FPGA Mapping. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (TCAD), 2018 (Accepted for publication).

- Lucas Machado, Antoni Roca, Jordi Cortadella. Robustness to Voltage Noise with Ring Oscillator Clocks. In *IEEE Transactions on Nanotechnology* (TNANO), 2018 (Under review).

Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. Jordi Cortadella Fortuny for giving me the possibility of pursuing my PhD degree in Barcelona. His guidance, motivation, strive for excellence, and unending source of ideas helped me a lot during every piece of work we did together. It was an incredible opportunity to work with him.

I also thank the advisors of my Master's thesis in Brazil, André Reis and Renato Ribas, as they both encouraged me in thriving for a PhD degree in Europe. Especially André, for the connection between me and Jordi, and the support during the scholarship proposal and the first year in Catalonia.

I thank every person that I had the opportunity to work side by side these years in Barcelona. My lab colleagues Àlex Vidal, Alberto Moreno, Javier de San Pedro, Tuomas Hakoniemi, and Josep Sanchez, and the ones I had the pleasure to work with, Antoni Roca and Mayler Martins.

I also thank all my friends and family from Lajeado, Cachoeira do Sul, Porto Alegre, and now spread around the globe. You all helped me be who I am today, and certainly have a part in this degree. Particularly, my grandparents (*in memoriam* to my grandma Maria, who passed away in this period), my father Carson, my mother Gisele and my brother Jonas. I certainly missed you a lot during these years, separated by 9642 km and several hours by plane. You were, are and will always be my support to everything, the giants that took me in their shoulders and made me look further.

For my wife Rafaela Bortolini, I may not have enough words to express my gratitude. For marrying me, embarking with me in such a life-changing adventure, getting way out of our comfort zones. For the understanding, loving and caring. For insisting on having a dog, our beloved Pipoca, that helped us so much with her partnership and joy. For everything, I thank you.

This thesis has been performed with the support of CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil, and has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness and the European Union (FEDER funds) under grant TIN2017-86727-C2-1-R, and the Generalitat de Catalunya (2017 SGR 786).

Contents

List of Acronyms

AI	artificial intelligence
AIG	and-inverter graph
API	application programming interface
ASIC	application specific integrated circuit
BDD	binary decision diagram
BLIF	Berkeley logic interchange format
CAD	computer-aided design
CDC	cross-domain crossing
CEC	combinational equivalence checking
CMOS	complementary metal-oxide-semiconductor
CP	critical path
CTS	clock tree synthesis
CUDD	Colorado University Decision Diagram
DAG	directed acyclic graph
DC	don't care
DDR	double data rate
DRC	design rule checking
DSD	disjoint-support decomposition
DVFS	dynamic voltage and frequency scaling
EDA	electronic design automation
EMI	electromagnetic interference
EPFL	École Polytechnique Fédérale de Lausanne
ESL	equivalent series inductance
ESR	equivalent series resistance
FPGA	field-programmable gate array

GALS	globally asynchronous locally synchronous
HDL	hardware description language
HLS	high-level synthesis
IC	integrated circuit
IoT	internet of things
ISF	incompletely specified function
ITC	International Test Conference
ITRS	international technology roadmap for semiconductors
LUT	look-up table
MCNC	Microelectronics Center of North Carolina
MFFC	maximum fanout-free cone
PCB	printed circuit board
PDN	power delivery network
PI	primary input
PLL	phase-locked loop
PO	primary output
POS	product-of-sums
PPAC	power, performance, area, and cost
PVT	process, voltage, temperature
QoR	quality of results
ROBDD	reduced-ordered BDD
ROC	ring oscillator clock
RTL	register-transfer level
SDC	satisfiability don't care
SEC	sequential equivalence checking
SoC	system-on-chip
SOP	sum-of-products
SPICE	Simulated Program with Integrated Circuits Emphasis
SR-map	support-reducing remapping tool
STA	static timing analysis

UPC Universitat Politècnica de Catalunya

VLSI very-large-scale integration
VRM voltage regulator module

List of Figures

List of Tables

Chapter 1

Introduction

On September of 2018, the *integrated circuit* (IC) celebrated 60 years of its invention [51]. This tiny electronic component, also known as chip, caused one of most important technological progresses in human history: the digital revolution. All areas of knowledge have taken advantage of the IC, generating remarkable improvements at a much faster rate than ever before. Nowadays, there are far more chips than people on earth. They are present in computers, phones, televisors, medical devices, cars, trains, airplanes, and virtually everywhere with the introduction of the *internet of things* (IoT) [9]. Notwithstanding, ICs are also the underlying reason for the existence of multiple things: laptops, smart phones, self-driving cars, wearable devices, space exploration, the Internet, and consequently, all Internet-related services.

An IC is a miniaturized electronic circuit [51] manufactured with semiconductors, and designed to perform one or more logic functions. The solid-state transistors [108] are the on-off switches which turned out to be the basis for the implementation of integrated circuits. In 1965, Gordon Moore noticed that the number of transistors per chip doubled every year [91] since the invention of the IC in 1958. After reaching the first limitations of the *complementary metal-oxide-semiconductor* (CMOS) technology, Moore predicted that this trend would continue, but at a more conservative rate: the number of transistors on a chip would double every 2 years [92]. This prediction, later on called *Moore's law*, helped to drive the progress of the semiconductor industry ever since. This evolution was made possible by a large number of factors. For instance, the reduction of the transistor size shown in Fig. 1.1 (minimum feature size). In the course of 60 years, the minimum feature size scaled $\approx 3500\times$, reaching 7nm in 2018. Furthermore, ICs enabled the development of workstations, also emerging the industry of *electronic design automation* (EDA), producing a self-reinforcing virtuous cycle which continually pushed forward the state-of-the-art [1].

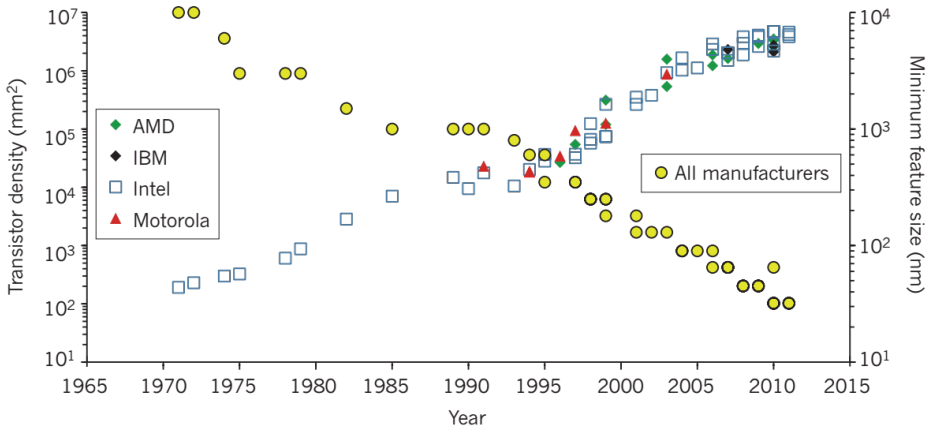


Figure 1.1: Evolution over time of the transistors density and the minimum feature size (Source: [36]).

In the early days, ICs were designed by hand, using engineering paper and color pencils. The masks for lithography were made out of rubylith [74], and manufacturing was performed with primitive planar technology. Several *computer-aided design* (CAD) tools were developed with the first computers: to help with the artwork and the routing of wires connecting transistors, and for circuit simulation (SPICE). During the 1980s, physical synthesis tools emerged, and circuits started to be described at *register-transfer level* (RTL) with the aid of a *hardware description language* (HDL), changing completely the methodology which ICs were designed. Soon after, the system behavior could be described in HDL and transformed into a netlist of logic gates, arising the field of *logic synthesis*.

The *standard cell* methodology shown in Fig. 1.2 is one of the factors that pushed forward Moore's law, reaching *very-large-scale integration* (VLSI) circuits. It is based on a limited set of digital logic gates (a cell library), with a standard height for all cells, and a rigid window of operation: the *clock*. This methodology is also known as *application specific integrated circuit* (ASIC) design flow, and can be divided into *high-level synthesis* (HLS), logic synthesis, and physical synthesis. HLS makes transformations at an architectural level, transforming a C-like algorithmic description into an RTL. Logic synthesis is responsible for the transformation of the circuit behavior description into a netlist of logic gates for a given technology, i.e., a digital mapped circuit [34]. The physical synthesis transforms the netlist derived by logic synthesis into a set of geometric shapes, which represent the different layers to be manufactured. Floor plan, placement, *clock tree synthesis* (CTS), routing and *design rule checking* (DRC) are some steps of physical synthesis [5].

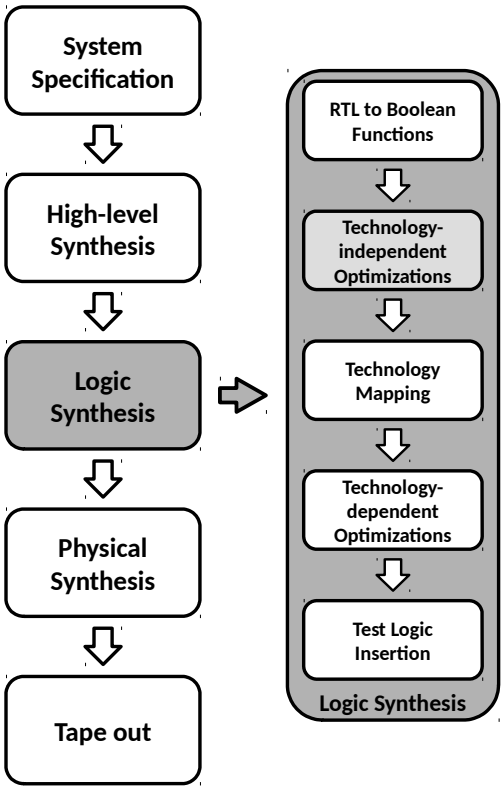


Figure 1.2: Logic synthesis in the standard cell flow of integrated circuits.

Logic synthesis is a key process in order to produce a chip with high *quality of results* (QoR). It can be divided into five steps. The first step consists of transforming the RTL description into a technology-independent representation, e.g., Boolean networks or *and-inverter graphs* (AIGs). Then, several optimizations are performed on this representation, reducing cost functions regarding area and delay. The following step is technology mapping, matching parts of the circuit representation to logic gates of a library. Several optimizations can be performed on the mapped circuit, such as reducing delay in order to meet the constraints, and reducing area and power consumption as low as possible. The final step is the test logic insertion.

Overall, logic synthesis methods try to minimize the number of components, delay and power consumption. Notice that functionally equivalent circuits with fewer components imply fewer transistors and lower costs. Making a circuit faster means to have the same outcome in less execution time, which is also desirable. The autonomy of portable devices based on batteries, the limitation for heat dissipation (and its impact on performance) of many devices are some of the reasons to have low power consumption as a goal.

1.1 Research motivation and goal

The scaling of the transistor size has historically resulted in reduction of costs, higher performance and lower power. Some reasons for this behavior are: the ICs are smaller and more chips can be manufactured in the same wafer, the capacitances are reduced, and lower voltages were required. However, the technology scaling is reaching a physical (and economical) limit [115].

The unavoidable heat generated by millions of devices jammed in the same small piece of semiconductor is a problem, generating issues such as *dark silicon* [35]. It is also worth noting that silicon is the most used semiconductor for manufacturing of ICs, and the feature size of 5nm predicted to 2020 also means that there will be features of ≈ 10 atoms in size [112]. At this point, quantum uncertainties will increase significantly the variability of devices, e.g., resistance, capacitance, and delay. The evolution of manufacturing technologies also has inherently high costs, with more photolithography steps, different machines, and a large amount of design rules. The investment of building a new foundry to scale down the feature size is in the order of billions of dollars, which has a decreasing economical appeal [19].

Many different technologies have been investigated in order to substitute CMOS and continue the exponential improvements witnessed in previous decades. The ideas researched range from quantum [94] and neuromorphic computing [47], to graphene compounds [63] and spintronic materials [118]. Nevertheless, no CMOS substitute has made into production until now.

In the past, when CMOS limits were hit, it did not result in the end of the IC. Improvements will happen a slower rate, and come from different areas instead of transistor scaling. Even if no changes are perceived in transistor density, performance will be increased and costs will be reduced due to better manufacturing productivity, cycle time reduction, defect elimination, and the design of more powerful EDA tools. Note that the semiconductor industry currently has a worldwide revenue of ≈ 450 billion dollars per year, and it continues to reinvent itself, exploring *artificial intelligence* (AI) [59], cloud computing [7], and hardware acceleration [100].

Many experts believe that improvements will continue to happen in the foreseeable future [30]. The *field-programmable gate array* (FPGA) is one of the potential driving forces of the industry. FPGAs emerged in the late 1980s, composed of small memory blocks, which are used to implement combinational and sequential logic, in an array of programmable interconnections. An FPGA with ≈ 30 billion transistors, which implement 5.5 million logic elements, is commercially available since 2016 [48]. Clearly, EDA methods must be updated in order to deal with the size and complexity of such large ICs, while maintaining or improving the QoR of the final implementation.

Logic synthesis is one of the topics of this thesis. The algorithms involved in logic synthesis perform extremely complex tasks, with many variables to be considered, and trying all possibilities is not computationally affordable. The necessity of having reasonable solutions within time-to-market leads to multiple heuristics, generating sub-optimal results. Notice that the results obtained by state-of-the-art logic synthesis tools still have room for improvement, and finding optimal solutions may be feasible only for small circuits. Additionally, for numerous reasons, digital circuits typically operate with rigid clock sources. However, the size and complexity of current ICs lead to excessively conservative timing margins, and consequently, performance degradation and increased costs. Considering these possibilities, the main goal of this thesis is to explore alternatives in order to produce faster and cheaper ICs, even with the established CMOS technology.

1.2 Contributions of this thesis

In this thesis, alternatives to improve *power, performance, area, and cost* (PPAC) in the established CMOS technology are proposed. This work explores advances in EDA for three different topics: area minimization using Boolean methods, area and delay reduction for designs based on *look-up tables* (LUTs), and an alternative clocking scheme in order to improve performance, leakage power and costs. Specifically, the proposed contributions are:

1. A technology-independent method for area minimization of combinational logic, based on a multi-output decomposition using two-literal divisors (see Chapter 3).

2. A functional decomposition which uses the support size as cost function, and a recursive remapping approach targeting LUT-based FPGAs (see Chapter 4).

3. An analysis on dynamic variability mitigation and simplification of *power delivery networks* (PDNs) using an adaptive clocking scheme based on *ring oscillator clocks* (ROCs) (see Chapter 5).

The remaining of this section provides a summary of these contributions.

1.2.1 Boolean decomposition using two-literal divisors

Optimization techniques applied in technology-independent representations are typically limited to single-output transformations. Additionally, these techniques are highly biased by the structure, leading to sub-optimal results.

The work presented in Chapter 3 proposes a method for area minimization, exploring multi-output decomposition and Boolean methods, which offer less structural bias. Small parts of the circuit with multiple outputs are identified, and Boolean division using two-literal divisors is applied in order to increase logic sharing. This contribution presents the following characteristics:

- Boolean decomposition with two-literal divisors [90] is generalized from single-output to multi-output functions.
- The selection of divisors is customized to increase the logic shared among multiple outputs.
- A set of filters is proposed to reduce the search space.
- Area minimization is achieved by iteratively applying Boolean decomposition to KL-cuts [69, 75] of the circuit representation.

These contributions have been published in the following paper:

> [66] Lucas Machado, Jordi Cortadella. Boolean Decomposition for AIG Optimization. In *Proceedings of ACM Great Lakes Symposium on VLSI* (GLSVLSI), pages 143-148, May 2017

1.2.2 Support-reducing logic decomposition and remapping

The cost functions for most decomposition methods were defined due to the high correlation with the area of cell-based designs, e.g., literals, cubes. However, these cost functions have a weaker correlation for FPGAs based on LUTs. Moreover, local optimizations have limited power due to the structural bias of the circuit descriptions, which are typically designed for ASICs. The work presented in Chapter 4 proposes the reduction of the structural bias by remapping the LUT network and decomposing the derived functions using the support size as cost function. The two main contributions are:

1. A functional decomposition, which is guided by the support size, and it is based on simple and fast support-reducing techniques.
2. A recursive remapping approach, that reduces the structural bias of the subject graph, and uses the FPGA mapping metrics as cost function.

The methods are able to improve several best known results of the EPFL benchmarks [6], and obtain significant improvements in comparison with the results of a commercial tool. The reasons for these improvements are the following.

- The mapping result is used to guide the resynthesis algorithm, instead of literals and cubes. This cost function reduces the miscorrelation between intermediate and final results, accepting transformations that will contribute to improve the final solution.

- A recursive collapsing strategy is applied instead of a local *partial collapsing*, which reduces the structural bias of the subject graph.

- Also, additional structures are explored, which are generated by a support-reducing functional decomposition. Notice that the support size as cost function makes sense for FPGAs: a k-input function with *any* number of literals can be implemented with a single LUT of k inputs.

These contributions have been published in the following papers:

[68] Lucas Machado, Jordi Cortadella. Support-Reducing Functional Decomposition for FPGA Technology Mapping. In *Proceedings of International Workshop on Logic & Synthesis* (IWLS), pages 79-86, June 2018

[67] Lucas Machado, Jordi Cortadella. Support-Reducing Decomposition for FPGA Mapping. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (TCAD), 2018 (Accepted for publication)

1.2.3 Voltage noise mitigation using ROCs

Variability, static or dynamic, is one of the biggest challenges in current ICs. Typically, variability is considered by adding guard band margins to the nominal clock period. However, this has led to excessively conservative timing margins, degrading performance. Voltage noise is the main source of dynamic variability and a major concern for the design of PDNs. Lower supply and threshold voltages were made possible with technology scaling, but power density was also increased. Consequently, power integrity became a key factor in the design of reliable high-performance circuits.

ROCs have been proposed as an alternative to mitigate the negative effects of voltage noise. The capability of reacting instantaneously to large voltage variations makes ROCs an attractive solution, which also allows to relax the constraints required for the PDN design. However, the effectiveness highly depends on the design parameters of the PDN, power consumption patterns, and the spatial locality of the ROC within the clock domains.

The work in Chapter 5 presents an analysis on voltage locality for a design using ROCs as clock source. Voltage locality is introduced by multiple activity patterns using an on-chip power distribution model. A trade-off between the number of ROC domains and performance is presented. Also, modifications in the PDN are evaluated, such as removing on-chip decoupling capacitance and changing the number and placement of the power bumps. The goal of this work is to present a conservative analysis of the benefits of using ROCs when dealing with problems related to voltage noise. Robustness to voltage noise is achieved without degrading performance, making possible the simplification of the PDN design. These contributions have been published in the following papers:

> [71] Lucas Machado, Antoni Roca, Jordi Cortadella. Increasing the Robustness of Digital Circuits with Ring Oscillator Clocks. In *Proceedings of the International Workshop on Resiliency in Embedded Electronic Systems* (REES), pages 29-34, March 2017
>
> [72] Lucas Machado, Antoni Roca, Jordi Cortadella. Voltage Noise Analysis with Ring Oscillator Clocks. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI* (ISVLSI), pages 1-6, July 2017
>
> [73] Lucas Machado, Antoni Roca, Jordi Cortadella. Robustness to Voltage Noise with Ring Oscillator Clocks. In *IEEE Transactions on Nanotechnology* (TNANO), 2018 (Under review)

1.3 Manuscript organization

This thesis is structured into 6 chapters. The present chapter constitutes an introduction to the thesis. The remaining of this thesis is organized as follows.

Chapter 2: *Background* - This chapter provides a set of important preliminary information regarding all contributions of the thesis, organized in two main sections: logic synthesis and adaptive clocking.

Chapter 3: *AIG Optimization via Boolean Decomposition* - This chapter investigates area minimization using AIGs, exploring Boolean methods in order to reduce the number of nodes. Boolean division with two-literal divisors is applied to multi-output functions, and AIGs are minimized through local optimization.

Chapter 4: *Support-reducing Decomposition for FPGA Mapping* - This chapter proposes two methods targeting LUT-based FPGAs. A functional decomposition approach which uses the support size as cost function, exploring the inherent characteristics of FPGAs. Also, an recursive remapping method is proposed, which reduces the structural bias of the subject graph and uses the mapping results as cost function, obtaining significant reductions in area and delay.

Chapter 5: *Robustness to Voltage Noise with Ring Oscillator Clocks* - This chapter presents an analysis of dynamic variability mitigation using an adaptive clocking scheme based on ROCs. The impact of the PDN parameters and ROC location on the robustness to voltage noise are investigated. Several PDN simplifications are analyzed, showing that tolerance to voltage noise and related benefits can be increased with multiple ROC domains.

Chapter 6: *Conclusions and Future Work* - The final chapter concludes the thesis, presenting a summary of the contributions and providing ideas for future research based on the present manuscript.

Chapter 2

Background

This chapter presents two main sections that provide important concepts regarding the contributions of the thesis. Section 2.1 presents background for the contributions in logic synthesis, whereas Section 2.2 introduces topics regarding adaptive clocking.

2.1 Logic synthesis

Logic synthesis is an important area of study in the field of *electronic design automation* (EDA), being responsible for the transformation of a circuit behavioral description into a netlist of gates for a technology, i.e. a mapped circuit. Typical objectives of the logic synthesis are to reduce area, delay, power, or a combination of these. This section presents the background for the thesis contributions on logic synthesis, presented on Chapters 3 and 4.

2.1.1 Boolean functions

The Boolean domain is defined as $\mathbb{B} = \{0, 1\}$, where 0 and 1 represent two well-defined logic states, such as *true* (1) and *false* (0). An n-dimensional Boolean space $\mathbb{B}^n$ is composed of 2^n distinct Boolean vectors of length n. For instance, $\mathbb{B}^1 = \{0, 1\}$, $\mathbb{B}^2 = \{00, 01, 10, 11\}$, and so on.

A *completely specified* Boolean function can be described as a mapping between Boolean spaces [34]. A Boolean function with n inputs and m outputs ($n, m \in \mathbb{N}$) can be represented with the mapping: $\mathbb{B}^n \rightarrow \mathbb{B}^m$. It is a single-output function if $m = 1$, and it is a multi-output function if $m > 1$.

An *incompletely specified function* (ISF) is defined over a sub-set of $\mathbb{B}^n$, where there are undefined function points, which are also known as *don't care* (DC) conditions. In another definition, an ISF can be represented as

$\mathbb{B}^n \rightarrow \{0, 1, -\}^m$, where '$-$' denotes a DC value, i.e., it can be either '1' or '0'. The sub-domains of the function F that evaluate to 1, 0 and $-$ are denoted the ON-set, the OFF-set, and the DC-set, and can be represented by the completely specified functions F_{ON}, F_{OFF} and F_{DC}, respectively. If the DC-set is empty, then the function is completely specified.

2.1.1.1 Boolean operations

There are three basic Boolean operations: negation (or complement) (NOT), conjunction (AND), and disjunction (OR). The negation is a unary operation, i.e., it is in the Boolean space $\mathbb{B}^1$, whereas the conjunction and disjunction are operations between two or more Boolean variables. Consider the Boolean variables x and y. The negation of x is denoted by $\bar{x}$, and its result is $\bar{x} = 0$ if $x = 1$, and $\bar{x} = 1$ if $x = 0$. The AND operation can be denoted by $x \cdot y$ (or xy), and its result is $x \cdot y = 1$ if $x = y = 1$, and $x \cdot y = 0$ otherwise. The OR operation can be represented as $x + y$, and its result is $x + y = 0$ if $x = y = 0$, and $x + y = 1$ otherwise.

Another important Boolean operation is the *exclusive-or* (XOR). The XOR operation is denoted by $x \oplus y$, and its result is $x \oplus y = 0$ if $x = y$, and $x \oplus y = 1$ if $x \neq y$. The exclusive-or can also be described in the conjunctive and disjunctive forms: $(\bar{x} \cdot y) + (x \cdot \bar{y})$ and $(\bar{x} + \bar{y}) \cdot (x + y)$, respectively.

2.1.1.2 Cofactors

Consider the Boolean function $F(X) : \mathbb{B}^n \rightarrow \mathbb{B}^1$. The *support* of F is the set of variables $X = (x_1, x_2, \ldots, x_i, \ldots, x_n)$, and the *support size* is denoted by $|F|$. A Boolean variable $x_i \in X$ is considered to be *essential* for the function F if there are at least two elements in $\mathbb{B}^n$ that are different only due to x_i.

The *positive cofactor* of F with respect to variable $x_i \in X$ consists of assigning x_i to '1', i.e., $F_{x_i} = F(x_1, x_2, \ldots, 1, \ldots, x_n)$. Similarly, the *negative cofactor* of F with respect to variable x_i is obtained by assigning x_i to '0', i.e., $F_{\overline{x_i}} = F(x_1, x_2, \ldots, 0, \ldots, x_n)$. If the negative and the positive cofactors with respect a variable x_i are equal, i.e., $F_{\overline{x_i}} = F_{x_i}$, then the variable x_i is not in the support of F. A *cube-cofactor* consists of performing the cofactor operation recursively, e.g., assigning the variables $\{x_i, x_j\} \subseteq X$ in $F(X)$ to $x_i = 0$ and $x_j = 1$, which can be denoted as $F_{\overline{x_i} x_j}$.

Cofactors can be used to extract information from F with respect to a variable in its support. One of the most important operations based on cofactors is the Boole's expansion theorem, also known as the *Shannon expansion* or decomposition, which is described in (2.1).

$$F = \overline{x_i} \cdot F_{\overline{x_i}} + x_i \cdot F_{x_i} \tag{2.1}$$

There are also other important operations based on cofactors: the *Boolean difference* (or derivative) in (2.2), the *existential abstraction* (or smoothing) in (2.3), and the *universal abstraction* (or consensus) in (2.4).

$$\delta F/\delta x_i = F_{\overline{x_i}} \oplus F_{x_i} \tag{2.2}$$
$$\exists x_i F = F_{\overline{x_i}} + F_{x_i} \tag{2.3}$$
$$\forall x_i F = F_{\overline{x_i}} \cdot F_{x_i} \tag{2.4}$$

The *Davio expansion* is another decomposition based on cofactors, using the XOR operation and the Boolean difference. There are two Davio expansions: the positive (2.5), and the negative (2.6) forms.

$$F = (x_i \cdot \delta F/\delta x_i) \oplus F_{\overline{x_i}} \tag{2.5}$$
$$F = (\overline{x_i} \cdot \delta F/\delta x_i) \oplus F_{x_i} \tag{2.6}$$

2.1.1.3 Unateness and containment

A Boolean function F is *positive unate* in the variable x_i if $F_{x_i} \supseteq F_{\overline{x_i}}$, where $\supseteq$ is the set operation of inclusion. Similarly, F is *negative unate* in the variable x_i if $F_{\overline{x_i}} \supseteq F_{x_i}$. Otherwise, x_i is considered a *binate* variable in F. This is the concept of *unateness* [34], intended for completely specified functions. If a function F has only positive and negative unate variables, then F is considered *unate*. If F has one or more binate variables, then it is a binate function.

Containment [116] is a generalization of the concept of unateness for ISFs. There is a *containment* in the function G of the variable x_i in the *positive* polarity if

$$(G_{DC\,x_i} \cup G_{ON\,x_i}) \supseteq G_{ON\,\overline{x_i}},$$

and a containment in the *negative* polarity if

$$(G_{DC\,\overline{x_i}} \cup G_{ON\,\overline{x_i}}) \supseteq G_{ON\,x_i},$$

where the operator $\cup$ is the set operation of union, G_{ON} is the ON-set of G, and G_{DC} is the DC-set of G.

2.1.2 Representation of Boolean functions

There are multiple forms to represent a Boolean function, each of them with a characteristic: canonicity, scalability, expressivity, etc. This section presents the representations used in the work proposed by this thesis: truth tables, Boolean expressions, *binary decision diagrams* (BDDs), Boolean networks, and *and-inverter graphs* (AIGs).

Table 2.1: Example of truth table for a Boolean function $F_1 : \mathbb{B}^3 \rightarrow \mathbb{B}^1$.

x_1	x_2	x_3	F_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

2.1.2.1 Truth tables

Truth tables are a straightforward representation of Boolean functions. A truth table can be partitioned into two parts: on one side, all possible combinations of the input variables are described; on the other side, the values of output variables are set according to the respective input combination. Table 2.1 shows an example of truth table for a Boolean function $F_1 : \mathbb{B}^3 \rightarrow \mathbb{B}^1$. The input vectors that evaluate the function to '1' are the ON-set, e.g., $\{011, 101, 111\}$. Similarly, the input vectors that evaluate F_1 to '0' are the OFF-set, e.g., $\{000, 001, 010, 100, 110\}$. Truth tables are a canonical representation, given the same variable order.

2.1.2.2 Boolean expressions

A single-output Boolean function can also be represented as a Boolean expression. In this case, the Boolean operators are applied to the input variables of the function in order to represent its functionality. Each time a Boolean variable appears in a Boolean expression, negated or not, it is considered as one *literal*. Boolean expressions with fewer literals are preferred, since these will likely require less logic elements to be implemented. Notice that a Boolean expression represents exactly one Boolean function, but a Boolean function can be represented by multiple different Boolean expressions. For example, consider the function F_1 from Table 2.1.

Canonical sum-of-products: Extracting the Boolean vectors that evaluate F_1 to '1', and representing them as Boolean expressions, in order to implement the correct functionality of F_1, the result obtained is described in (2.7), which is a *sum-of-products* (SOP).

$$F_1 = (\overline{x_1} x_2 x_3) + (x_1 \overline{x_2} x_3) + (x_1 x_2 x_3) \tag{2.7}$$

Canonical product-of-sums: Similarly, considering the Boolean vectors that evaluate F_1 to '0' as Boolean expressions, the result obtained is a *product-of-sums* (POS), which is described in (2.8).

$$F_1 = (x_1+x_2+x_3)(x_1+x_2+\overline{x_3})(x_1+\overline{x_2}+x_3)(\overline{x_1}+x_2+x_3)(\overline{x_1}+\overline{x_2}+x_3) \quad (2.8)$$

Such SOP and POS are canonical, as they are translations of the Boolean vectors to expressions, applying logic operations to implement the Boolean function. However, these representations typically have several literals and cubes, with 9 literals (and 3 cubes) in (2.7), and 15 literals (and 5 cubes) in (2.8). SOP and POS are also two-level Boolean expressions, and two-level minimization [15] methods can be applied to reduce the number of literals.
Factored form: Further optimizations can be applied in order to reduce the number of literals, such as factorization [13, 76], generating Boolean expressions with unbounded number of levels. For instance, a factored-form expression of the function F_1 is shown in (2.9), with 3 literals.

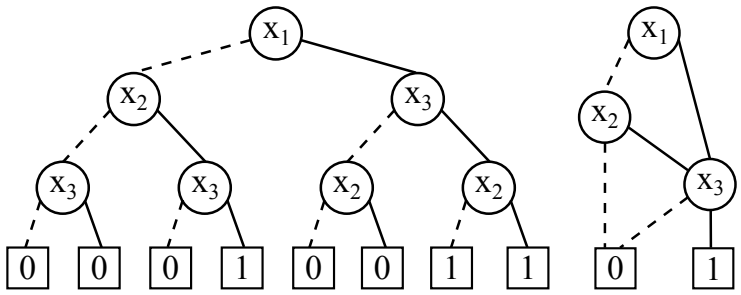
$$(x_1 + x_2)x_3 \quad (2.9)$$

2.1.2.3 Binary decision diagrams

A BDD is another representation of Boolean functions [3]. BDDs are rooted, *directed acyclic graphs* (DAGs) with two terminal nodes (0 and 1), and each nonterminal node represents a Boolean variable with two outgoing edges: the 0-edge and the 1-edge. The BDD representation of the function F_1 from Table 2.1 is shown in Fig. 2.1(a), where the dashed lines are the 0-edges, the non-dashed ones are the 1-edges, the circles represent the nonterminal nodes, and the squares are the terminal nodes. Notice that BDDs are based on the Shannon expansion. A *reduced-ordered BDD* (ROBDD) is a BDD in which the nonterminal nodes are organized in a fixed variable order, and the number of BDD nodes is reduced using minimization rules [17]. ROBDDs are a canonical representation, given the same variable order. In this work, ROBDDs are referred as BDDs. BDDs are an efficient representation (with a few exceptions) and are more scalable than other functional representations, such as truth tables. Also, there are modern software libraries which efficiently implement BDD operations [109].

2.1.2.4 Boolean networks

Graphs are data structures widely used in computer science, due to their high expressivity and the many efficient algorithms for graphs. A graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is a simple structure composed of the set of vertices $\mathcal{V}$ (or nodes),



(a) BDD (b) ROBDD

Figure 2.1: (a) BDD and (b) ROBDD of the function F_1 from Table 2.1.

and the set of edges $\mathcal{E}$ (or arcs), which connect two vertices. However, in order to represent Boolean functions, two constraints are required: (1) the edges must be directed; and (2) cycles are not allowed. These conditions coincide with DAGs, which are used to represent circuits.

A *Boolean network* (or logic network) is a DAG with three types of nodes: the primary inputs (with no incoming edges), the primary outputs (with no outgoing edges), and the internal nodes (or logic gates). The edges are directed from inputs to outputs. The internal nodes can represent any n-input and m-output Boolean function. However, due to the limitation of *application specific integrated circuit* (ASIC) and *field-programmable gate array* (FPGA) technologies, the logic nodes are typically reduced to functions with $n \leq 6$ and $m \leq 2$. This process of breaking a large function into smaller ones is performed by decomposition, which is explained in Section 2.1.3.

2.1.2.5 And-inverter graphs

An AIG [79] is a specific type of Boolean network in which each node has either 0 incoming edges - the *primary inputs* (PIs), 2 incoming edges - the *AND nodes*, or 1 incoming edge - the *primary outputs* (POs). The PI and PO nodes do not have a function associated, whereas the AND nodes perform the Boolean operation AND for two input variables. The edges can implement an NOT operation or not. Sequential elements are considered as PI/PO pairs. An example of AIG is shown in Fig. 2.2: the dashed lines indicate negated edges, the circles are AND nodes, the squares at the bottom are PIs, and the squares at the top are POs.

Using only the NOT and AND operations, AIGs are a simple and powerful data structure, and the state-of-the-art to represent very large circuits, e.g., thousands of inputs, millions of AIG nodes. However, AIGs are not canonical, i.e., the same Boolean function can be represented by different AIGs.

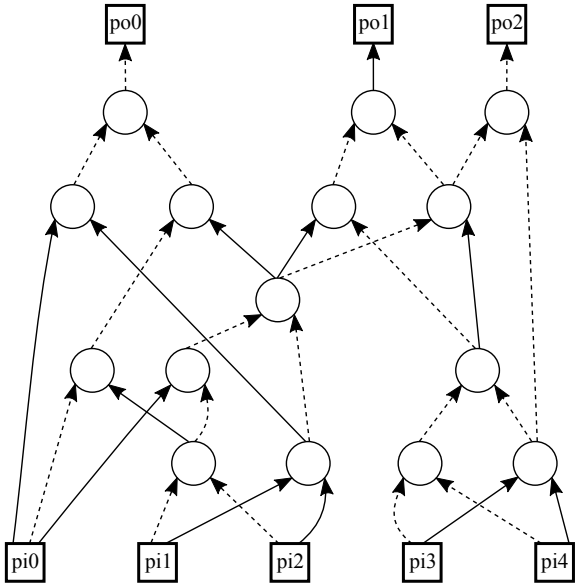


Figure 2.2: Example of an AIG, with 15 nodes and 5 levels.

In technology-independent logic synthesis, it is not known the effective costs in the target technology. Therefore, different cost functions are used to predict the cost of the final circuit, such as literals in Boolean expressions. For AIGs, the area is correlated with the number of AND nodes, whereas the delay is proportional to the logic depth between the PIs and POs.

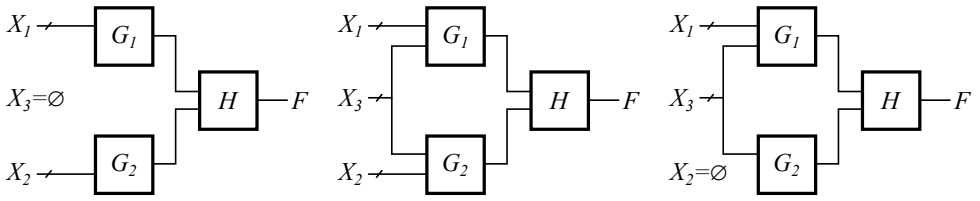
2.1.3 Logic decomposition

Logic (or functional) decomposition is a method of breaking a large, complex Boolean function into a set of smaller, simpler functions. Functional decomposition was introduced by Ashenhurst [8], expressing a Boolean function $F(X)$ in terms of other Boolean functions G and H:

$$F(X) = H(G(X_1), X_2), \tag{2.10}$$

where $X_1 \neq \emptyset$, $X_2 \neq \emptyset$, and $X = X_1 \cup X_2$. The sets X_1 and X_2 are known as bound-set and free-set, respectively. Only single-output functions are considered in the Ashenhurst decomposition [8], and functional decomposition is extended to multiple-output functions in the work proposed by Curtis [31]. An example of decomposition is shown in (2.11).

$$F = (x_1 \cdot x_4) + (x_2 \cdot x_3 \cdot x_4) + x_5 \xrightarrow{decomposition} \begin{cases} G = x_1 + (x_2 \cdot x_3) \\ F = (G \cdot x_4) + x_5 \end{cases} \tag{2.11}$$



(a) Disjoint support (b) Strong (c) Weak

Figure 2.3: Examples of (a) disjoint, (b) strong, and (c) weak bi-decompositions.

2.1.3.1 Bi-decomposition

A bi-decomposition is a special case of functional decomposition in which the derived functions have two or fewer inputs, i.e., $|H| \leq 2$. Given a function H, such that $|H| = 2$, F is bi-decomposable if it can be represented as:

$$F(X) = H(G_1(X_1 \cup X_3), G_2(X_2 \cup X_3)), \tag{2.12}$$

where $X_1 \cap X_2 = \emptyset$, $X_1 \cap X_3 = \emptyset$, $X_2 \cap X_3 = \emptyset$, and $X = X_1 \cup X_2 \cup X_3$. If $X_3 = \emptyset$, then it is a *disjoint-support decomposition* (DSD), and such decompositions [11, 77, 78, 18] are of special interest for their low implementation cost. If $X_1 \neq \emptyset$, $X_2 \neq \emptyset$, and $X_3 \neq \emptyset$, then it is a strong bi-decomposition. Otherwise, if $X_1 = \emptyset$ or $X_2 = \emptyset$, then it is a weak bi-decomposition [87]. Fig. 2.3 shows schematics with examples of these bi-decompositions. A bi-decomposition is also support-reducing if $X_1 \cup X_3 < X$ and $X_2 \cup X_3 < X$.

Bi-decomposition algorithms typically perform decompositions based on the Boolean operations AND, OR and XOR, recursively reducing the size of the sub-functions. Such algorithms are top-down approaches, relying on cost functions to estimate the actual implementation cost. This characteristic may also impact the area results for the cases with potential logic sharing or hierarchy, and a following process for area recovery may be required.

2.1.3.2 Algebraic and Boolean division

Logic synthesis algorithms can be divided into two groups: (1) the algebraic methods, which consider the Boolean functions as polynomial expressions, and (2) the Boolean approaches. Table 2.2 presents all the properties considered in Boolean approaches, whereas algebraic methods only consider properties {1,2,3,4,5,6,8,10} during transformations. On one hand, algebraic methods are very fast, but the quality of results are typically far from optimal. On the other hand, Boolean approaches are able to obtain better results, but also require much more execution time and memory consumption.

Table 2.2: Boolean algebra properties.

#	Property	Expression
1	Associativity	$x + (y \cdot z) = (x + y) \cdot z$
2		$x \cdot (y + z) = (x \cdot y) + z$
3	Commutativity	$x + y = y + x$
4		$x \cdot y = y \cdot x$
5	Identity	$x + 0 = x$
6		$x \cdot 1 = x$
7	Annihilator	$x + 1 = 1$
8		$x \cdot 0 = 0$
9	Distributivity	$x + (y \cdot z) = (x + y) \cdot (x + z)$
10		$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
11	Idempotence	$x + x = x$
12		$x \cdot x = x$
13	Absorption	$x \cdot (x + y) = x$
14		$x + (x \cdot y) = x$
15	Complementation	$x \cdot \bar{x} = 0$
16		$x + \bar{x} = 1$
17	De Morgan	$\bar{x} \cdot \bar{y} = \overline{x + y}$
18		$\bar{x} + \bar{y} = \overline{x \cdot y}$
19	Double negation	$\overline{(\bar{x})} = x$

The concept of *division* of a Boolean function F is given by the expression $F = (D \cdot Q) + R$, where the Boolean functions D, Q and R are the divisor, quotient and remainder, respectively. The function D is called a divisor of F if $R \neq 0$, and a factor if $R = 0$. The division operation can be performed by algebraic or Boolean means.

A common approach to perform Boolean division is by using two-level minimizers that accept don't care information [34]. A new variable x is added and the division is performed by adding the *satisfiability don't care* (SDC) expression $x \oplus d$ to the DC-set of F, where $\oplus$ represent the Boolean exclusive-OR operator, followed by a two-level minimization.

Example: Consider the function $F = (\bar{a}bc) + (a + b)d$ represented as a factored form with 6 literals. It is possible to rewrite F as $F = (\bar{a}bc) + (xd)$ by performing algebraic division, using the divisor $x = a + b$. Boolean division can be performed by incorporating $x \oplus (a + b)$ in the DC-set and running a two-level minimization [15]. This process results in $F = (\bar{a}cx) + (xd)$, which can be represented as the factored form with 4 literals: $F = ((\bar{a}c) + d)x$.

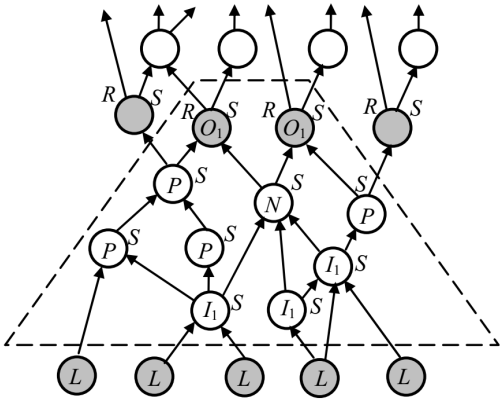


Figure 2.4: Example of a 1×1 window on top of a DAG (Source: [83]).

2.1.4 Local optimization

Boolean methods are more powerful, but also computationally more expensive in comparison with algebraic methods. In order to take advantage of Boolean methods, a known approach is to apply the transformations only to a part of a Boolean network at a time. Limiting the scope of the logic synthesis, also known as *local optimization*, is crucial for the scalability of many logic synthesis algorithms, specially considering the increasingly large Boolean networks used in the semiconductor industry.

2.1.4.1 Windowing

Another method to perform local optimization is *windowing*, which was introduced in [88]. The approach consists of gathering nodes around a node N, given some parameters. An example of a window is shown in Fig. 2.4.

The leaf and the root set are non-overlapping sets of DAG nodes, such that every path from the primary inputs to any node in the root set passes through some node in the leaf set. The window is composed of every node in the paths between the leaf set and the root set, including the root set, and excluding the leaf set. The leaf and root nodes are shaded in the window of Fig. 2.4, denoted L and R, respectively.

A window is typically denoted $n \times m$, where n denotes the number of levels towards the primary inputs, and m define the levels in the outputs direction. For example, the window presented in Fig. 2.4 is 1×1, where the nodes I_1 and O_1 are obtained from the fanin and the fanout of N, respectively.

A reconvergence computation is also typically performed, in order to identify a more significant portion of a DAG. The S nodes comprise the intersection of the fanins of O_1 nodes and the fanouts of I_1 nodes, given a distance

in levels of $n + m$. The leaf nodes (L) do not belong to S, but feed at least one of the nodes in S. The root nodes (R) belong to the set of S nodes, and also feed at least one node not in S. The nodes marked as P are obtained in the reconvergence process, as they are not connected directly to N, and are not leaf nor root nodes.

2.1.4.2 K-cuts

A K-feasible cut (or K-cut) of a node n is a subgraph (or a logic cone) rooted in the node n and with no more than K inputs. It is a useful method in AIG transformation algorithms [84], and in FPGA technology mapping [86].

Formally, a *cut* of a node n in a graph $\mathcal{G}$ is a set of nodes c such that every path between a primary input and n contains a node in c. A cut is irredundant if no subset of it is also a cut. If a cut is composed of one node, then it is a trivial cut. A K-cut [85, 95] of a node n in the graph $\mathcal{G}$ is an irredundant cut c with K or fewer nodes. The region defined by a K-cut is composed of all nodes in the path between c and n, including the node n and excluding the nodes in c.

The enumeration of K-cuts is realized by combining the cuts of the inputs of a node, where each cut is a set of nodes, and a union of these two sets is performed. Notice that the union of two K-cuts does not guarantee that the cut generated is K-feasible, therefore the enumeration process must remove any cut with more than K nodes. Consider the two sets of cuts A and B and the auxiliary set operation $\bowtie$ described in (2.13). The $\bowtie$ set operation removes the redundant cuts, and it is commutative, as the union set operation $\cup$ is also commutative.

$$A \bowtie B \equiv \{a \cup b \mid a \in A, b \in B, \mid a \cup b \mid < K\} \tag{2.13}$$

Given that $\Phi_K(n)$ is the set of K-cuts of $n \in \mathcal{G}$ and, if n is not a primary input or output, that n_1 and n_2 are its inputs. Then, $\Phi_K(n)$ is defined recursively [20], as described in (2.14).

$$\Phi_K(n) = \begin{cases} \{\{n\}\}, & n \text{ is a PI} \\ \{\{n\}\} \cup \{\Phi_K(n_1) \bowtie \Phi_K(n_2)\}, & \text{otherwise} \end{cases} \tag{2.14}$$

K-cuts are considered a method to derive sub-circuits compared to windowing, as the support size is controlled, and the reconvergence paths are identified. However, notice that the number of K-cuts grows exponentially with K, and for this reason different classes of K-cuts have been explored, such as factor cuts [20] and priority cuts [86]. An example of graph covering using K-cuts is shown in Fig 2.5. Given that $K = 4$, the K-cuts enumerated for each node is described in Table 2.3.

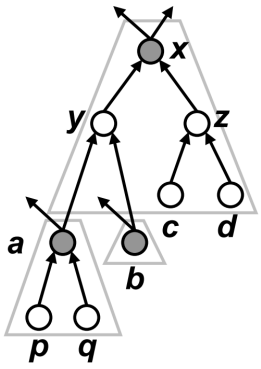


Figure 2.5: Example of a graph covering with K-cuts (Source: [20]).

Table 2.3: K-cuts enumeration for the AIG in Fig. 2.5.

Node	K-cuts
p	$\{p\}$
q	$\{q\}$
a	$\{a\}, \{p, q\}$
b	$\{b\}$
y	$\{y\}, \{a, b\}, \{p, q, b\}$
c	$\{c\}$
d	$\{d\}$
z	$\{z\}, \{c, d\}$
x	$\{x\}, \{y, z\}, \{y, c, d\}, \{a, b, z\}, \{a, b, c, d\}, \{b, p, q, z\}$

2.1.4.3 KL-cuts

K-cuts are an efficient way to represent a region of a graph regarding a single output. However, several K-cuts may be necessary to cover regions with multiple outputs, *duplicating logic*. A *KL-cut* [75, 69] identifies a multiple-output region in order to overcome this issue. A KL-cut is a sub-graph $\mathcal{G}_{KL}$ of a graph $\mathcal{G}$ with K inputs and L outputs. It is represented as two sets of nodes: the inputs $\mathcal{G}_K$, and the outputs $\mathcal{G}_L$.

If a node n belongs to a path between $n_K \in \mathcal{G}_K$ and $n_L \in \mathcal{G}_L$, being $n \notin \mathcal{G}_K$, then n is contained in $\mathcal{G}_{KL}$. Notice that all nodes in $\mathcal{G}_L$ are contained in $\mathcal{G}_{KL}$, and $\mathcal{G}_{KL}$ does not contain any node of G_K. The same algorithms used to enumerate K-cuts can be used to identify L-cuts, controlling both the number of inputs and outputs of a region. However, in the work of Chapter 3, the number of outputs is not restricted in KL-cuts enumeration, as in [70]. Therefore, for every K-cut of a node n, there is a *unique* KL-cut $\mathcal{G}_{KL}$.

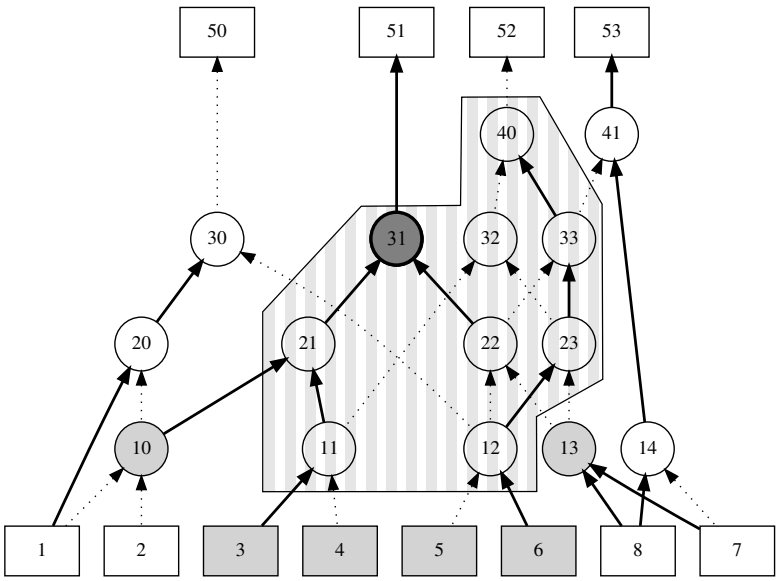


Figure 2.6: Example of KL-cut computation.

The nodes that are part of $\mathcal{G}_{KL}$ are identified by traversing forward the graph $\mathcal{G}$ from $\mathcal{G}_K$. A node n is part of $\mathcal{G}_{KL}$ if at least one of the K-cuts of n is a subset of or equal to $\mathcal{G}_K$. A node of $\mathcal{G}_{KL}$ is contained in $\mathcal{G}_L$ if it has a fanout to a primary output, or to a node not contained in $\mathcal{G}_{KL}$.
Example: Figure 2.6 depicts $\mathcal{G}_K = \{3, 4, 5, 6, 10, 13\}$ with its nodes in light gray, which is one of the K-cuts of the node 31, in dark gray. The KL-cut $\mathcal{G}_{KL}$ is obtained by traversing the AIG forward from $\mathcal{G}_K$, identifying the sub-graph hatched in Fig. 2.6. Nodes 31 and 40 have fanout to POs, and nodes 12 and 33 have fanout to nodes not contained in $\mathcal{G}_{KL}$, therefore $\mathcal{G}_L = \{12, 31, 33, 40\}$ is defined. Note that the logic of $\mathcal{G}_L$ nodes can be described as Boolean functions that depend on the same support, i.e., the $\mathcal{G}_K$ nodes.

2.1.5 Collapsing

Collapsing [27] a Boolean network means to replace it by another network with one node for each output, with each node representing the Boolean function of a primary output based solely on the primary inputs. This approach can be helpful as it removes structural redundancies from the logic network. The process of collapsing a circuit may be performed partially or globally.

Partial collapsing (or elimination) [23, 114] is an iterative process of removing nodes of a Boolean network by merging the function of a node with the ones in its fanout. The result is a Boolean network with a smaller set of

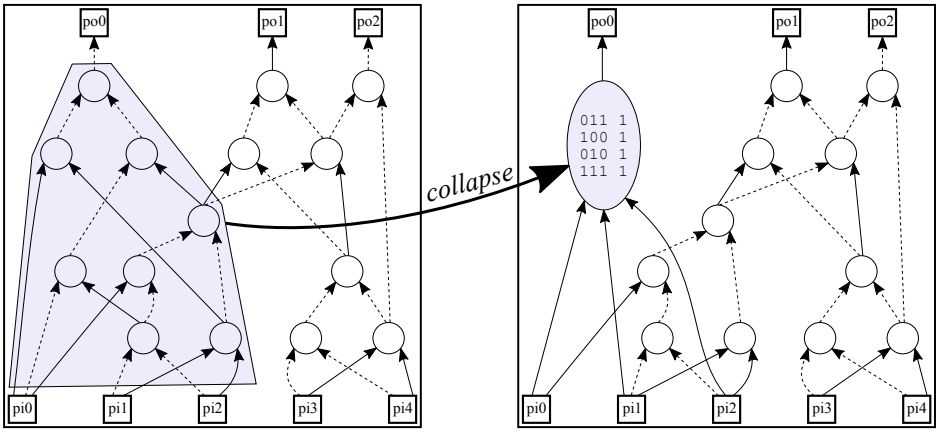


Figure 2.7: Example of the collapsing process for a primary output.

nodes, but with more complex Boolean functions. The number of iterations, or the number of nodes merged, is typically limited by a cost function, e.g., literals, AIG nodes, BDD nodes. Notice that the effectiveness of this process is biased by the structure of the subject graph, and also by the order in which is performed, generating sub-optimal results.

For the work proposed in Chapter 4, collapsing is performed globally and for each output individually. The result of such collapsing process is the logic function of a primary output based on the primary inputs, as shown in the example of Fig. 2.7. The output function obtained is the same regardless of the circuit structure, therefore the structural don't cares are removed [32]. Notice that logic sharing between outputs is potentially lost in this process, as observed in Fig. 2.7. This approach may result in larger area, but the possibilities of reducing circuit delay are increased. There are different approaches for global collapsing, e.g., using a multi-rooted BDD, which perform partial logic sharing.

2.1.6 AIG transformations

AIGs offer an homogeneous data structure, in which simple, fast, and scalable logic synthesis methods can be applied. The scalability is made possible by performing transformations on the AIG structure, using the number of nodes and the logic depth as cost function for area and delay, respectively. These AIG transformations are typically realized with fast local optimizations, which can be repeated multiple times. After several iterations, the optimized AIG is the subject graph for technology mapping. This section reviews the main transformations used for AIG optimization.

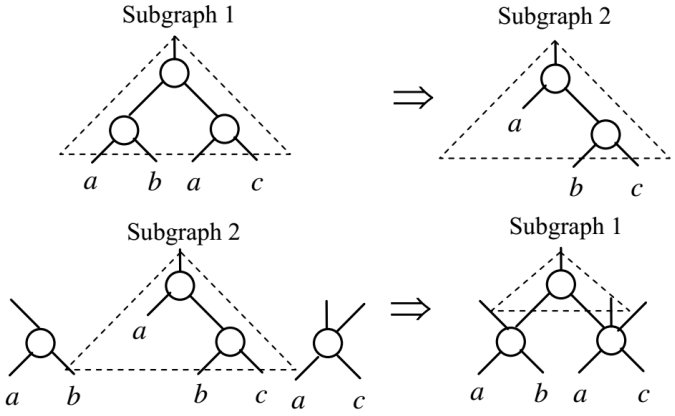


Figure 2.8: Examples of AIG rewriting (Source: [84]).

The first step is to derive the AIG structure from an input description. In ABC [14], this is performed by running the command *strash*, which can be divided in two parts. The first part is to replace each node of the input Boolean network by the equivalent AIG structure, which is derived by the factorization [13] of the Boolean function of the node. The second part is the redundancy removal, which detects and merges isomorphic circuit structures. This process is also known as *structural hashing* [79], which is performed using a hash-table, and ensures that there is only one node having the same pair of AIG nodes as fanins, considering permutation. After this process, it is said that the AIG is structurally hashed.

Refactoring [79, 84] is one of the methods for node count minimization. The method is implemented on the command *refactor* in ABC [14], and consists of iteratively extracting the *maximum fanout-free cone* (MFFC) for each AIG node, with a maximum of 16 variables. The logic function of the MFFC is factored [13], and the result of factorization is translated back to AIG format, replacing the original logic cone if the number of nodes is reduced (or not increased), and the logic depth is not increased.

Rewriting [79, 84] is another algorithm for minimizing the AIG size, enumerating all K-cuts at each AIG node, and replacing them by functionally equivalent and smaller pre-computed cuts. Two examples of AIG rewriting are depicted in Fig. 2.8. The example at the top regards a simple substitution for a smaller structure, whereas the example at the bottom shows a replacement for a larger AIG, but taking advantage of nodes already present in the network. The command *rewrite* in ABC [14] implements this approach for cuts with up to 4 variables, using pre-computed AIGs indexed in a hash table with all 222 NPN-equivalent classes of 4-input functions [84]. This method can also be restricted to transformations that do not increase logic depth.

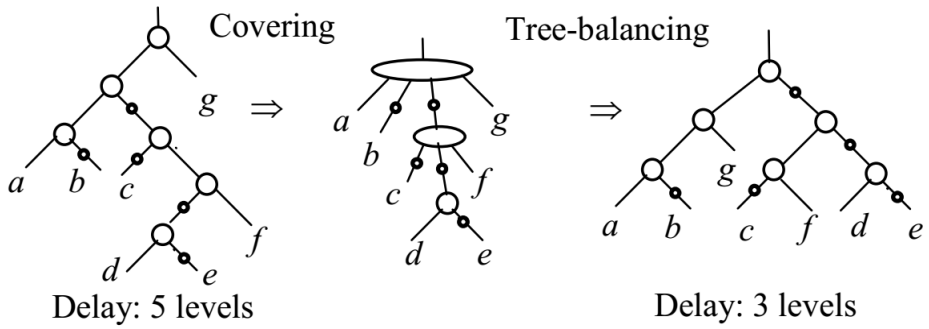


Figure 2.9: Example of a tree-balancing transformation (Source: [81]).

Balancing [27, 81] is a depth-aware transformation using algebraic tree-height reduction, performed by applying Boolean properties such as associativity, commutativity, and distributivity (see Table 2.2). An example of tree-balancing transformation is illustrated in Fig. 2.9. This algorithm [27] is also implemented in ABC [14] on the command *balance*, and it is frequently applied in between AIG node reduction approaches to minimize logic depth.

Another method for AIG optimization is *resubstitution* [79], which tries to re-express the Boolean function of a node by reusing other nodes present in the AIG, also known as divisors. The method is implemented on the command *resub* in ABC [14], and consists of iteratively extracting the MFFC for each AIG node, and rewriting the logic cone using k new nodes and removing l nodes. Similarly to other methods, if the number of nodes is reduced, i.e., $l > k$, then the modification is accepted. This approach can be considered a technology-independent version of the resynthesis methods based on resubstitution [53, 89].

It is also worth mentioning that AIG transformation methods are highly dependent on the input description. Therefore, a higher quality of results can be achieved by applying balancing, rewriting, resubstitution, and refactoring iteratively. For example, the *dc2* command [82, 75] in ABC [14] iterates these methods in order to obtain an optimized AIG. Transformations that increase the number of levels are accepted if *dc2* is executed without the *-l* option, therefore obtaining the minimum number of AIG nodes for this command.

2.1.7 FPGA technology mapping

Technology mapping is an important process in logic synthesis, which transforms a technology-independent circuit description into a network of gates from a given technology, i.e., a mapped circuit. This process can be divided into three steps: decomposition, matching, and covering.

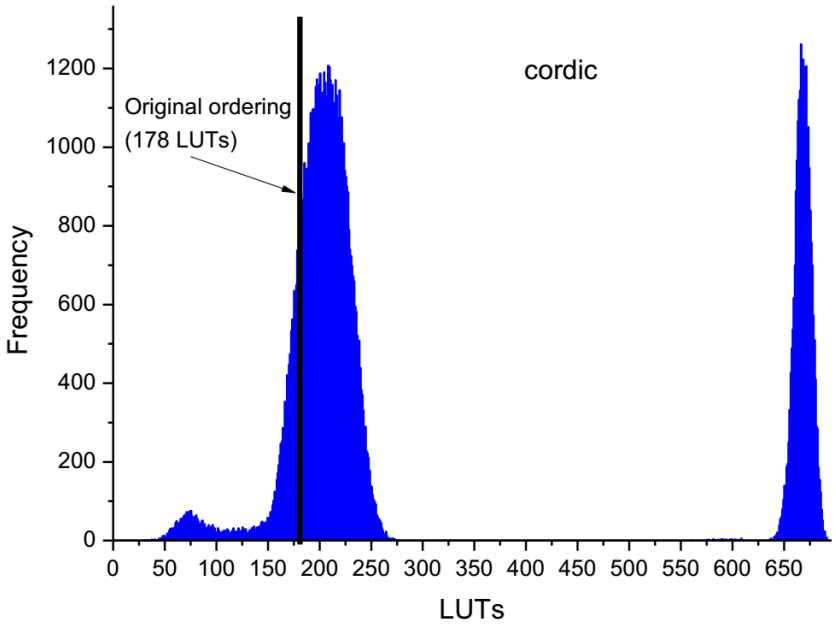


Figure 2.10: Distribution of solutions for the benchmark *cordic* (Source: [38]).

Decomposition transforms the initial description into a simpler, more restricted representation: the *subject graph*. This is important to limit the scope, making technology mapping computationally more tractable. Structural transformations are also applied in this step, such as breaking the subject graph into trees. The *matching* step consists of identifying logic gates from a library that can implement parts of the graph, taking advantage of techniques such as P-signature [45], considering the permutation of the inputs of a logic gate. Finally, the *covering* step chooses a subset of the matching possibilities in such a way that the entire subject graph is covered, while optimizing a cost function, e.g., area, power. Notice that all technology mapping steps are important. Low quality results can be delivered even with optimal algorithms for matching and covering, given that the input circuit is poorly described, or the decomposition step has incorrect metrics. This structure dependence is a problem known as *structural biasing* [21].

FPGAs are integrated circuits consisting of programmable logic blocks and interconnections. FPGAs can be reprogrammed multiple times, and have a much smaller initial cost and production time in comparison with ASICs. For these reasons, FPGAs are largely used for ASIC prototyping and low-volume applications. For FPGAs, the technology is typically consisted of *look-up tables* (LUTs), which are logic blocks that can be configured to implement any logic function of up to k variables.

In ABC [14], the subject graph is a structurally hashed AIG, and the FPGA mapping is performed on top of this structure [86]. Notice that the FPGAs mapping may vary significantly for functionally equivalent but structurally different AIGs. For example, a study on the impact of the variable ordering of the subject graph on the mapping result is performed in [38], with the distribution of solutions for the benchmark *cordic* shown in Fig. 2.10.

2.2 Adaptive clocking

The estimation of the path delays and their variability is critical for the performance and reliability of digital circuits. It is necessary to consider all conditions that may shift and affect the delay of every circuit path, such as the manufacturing process, the supply voltage, and the temperature (PVT corners). Static offsets of these conditions are considered at design phase, but dynamic shifts are hard to predict and conservative margins are added to prevent failures. Voltage noise is the main source of dynamic variability, and adaptive clocking is one of the methods used to mitigate it. This section presents the background for the contributions presented on Chapter 5, regarding the use of ring oscillators as the clock source.

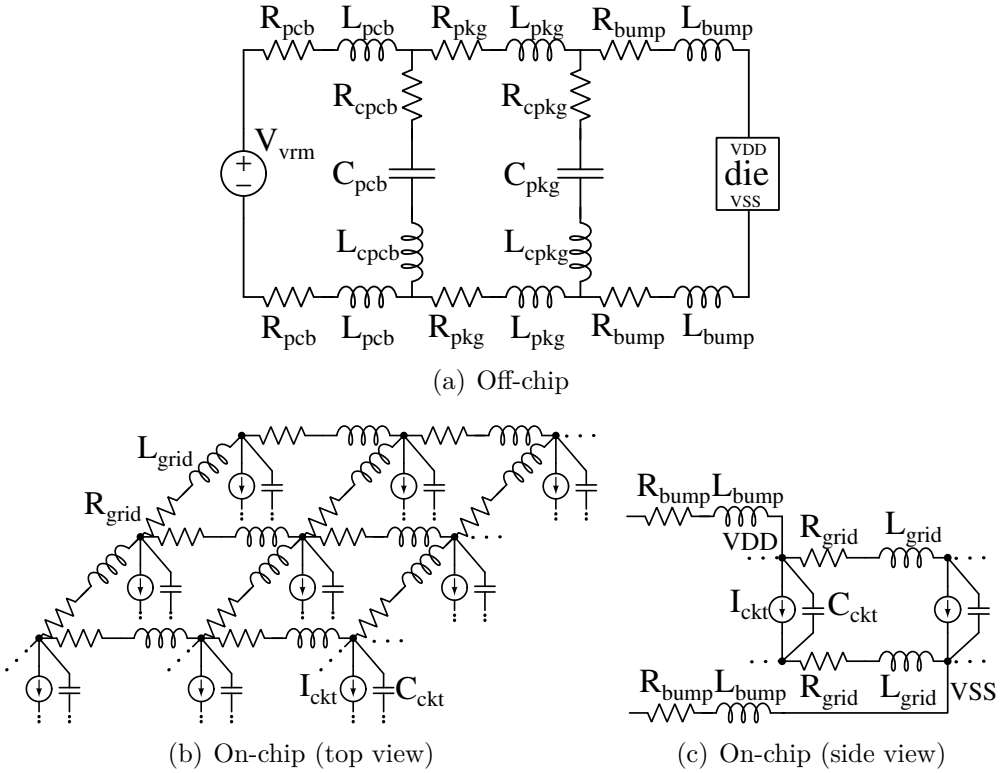
2.2.1 Power Integrity

Power integrity is one of the major challenges in the design of high-performance circuits. All components of the *power delivery network* (PDN) have a direct influence on the voltage fluctuations observed by the on-chip devices. Mitigating this noise is an arduous task that may have a significant impact on all design metrics: power, performance and area. The main components of voltage drops are resistive and inductive [99]:

$$\Delta V = R \cdot i(t) + L \cdot \frac{di}{dt}. \tag{2.15}$$

IR drops (static and dynamic) are produced by the parasitic resistance of the PDN, whereas inductive noise is mainly caused by large current differences, associated with the switching activity of the chip.

Static voltage offsets can be estimated at design time. However, the dynamic variations are hard to predict and this is the reason why overly conservative margins are often added to prevent unexpected failures. Unfortunately, voltage droops that exceed the defined margins cannot be fully eliminated. Clock and power gating are typical low-power techniques that can unintentionally produce large voltage droops. When many devices are



(a) Off-chip

(b) On-chip (top view)

(c) On-chip (side view)

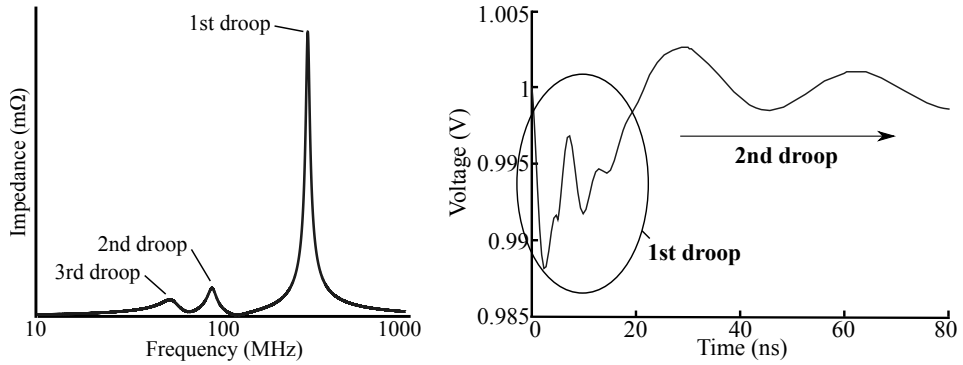
Figure 2.11: PDN model with off-chip and on-chip parasitics.

simultaneously activated, a large di/dt is originated. If that situation is periodically repeated and aligned with a resonant frequency of the PDN, large voltage swings may appear, exceeding the ones tolerated by the system.

A common strategy to mitigate voltage noise is to increase the amount of on/off-chip capacitance by adding decoupling capacitors (decaps) [99]. Unfortunately, the additional decaps imply an increase in area and leakage power consumption. In [120], different amounts of integrated voltage regulators are investigated, analyzing the penalties in area and power for the voltage noise reductions obtained. Other proposals include improving the chip-package impedance, static and dynamic voltage margining, performance throttling and stalling using voltage sensors [49, 12]. All these approaches have important overhead in design cost, area, power or performance.

2.2.2 Voltage noise

The PDN delivers the power and ground voltages to all devices of a chip. Fig. 2.11(a) depicts a PDN model with its off-chip components: *voltage regulator module* (VRM), board (PCB), package (PKG), and the connection



(a) Frequency response (b) Voltage droops

Figure 2.12: (a) The frequency response of a typical PDN, and (b) the voltage droops generated by a single current spike

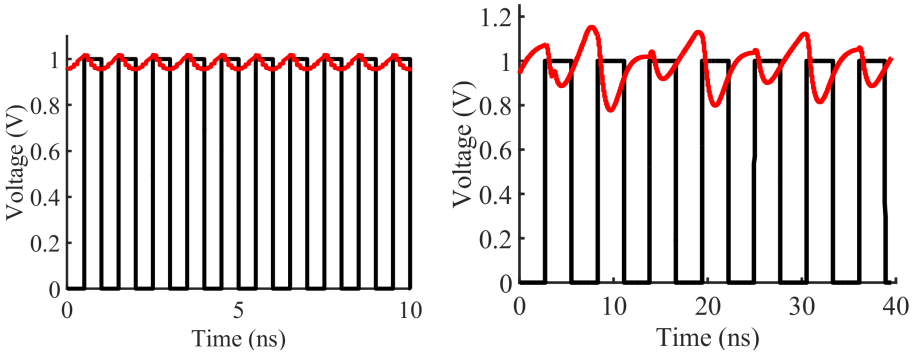
bumps [99]. The on-chip power networks are modeled as a grid, shown in Fig. 2.11(b) and Fig. 2.11(c). Notice that both the power and ground networks are considered in the model, with a V_{DD} or a V_{SS} bump connected at every grid point.

The power distribution has parasitic inductances, resistances and capacitances, which can be modeled as illustrated in Fig. 2.11. Also, decaps are placed at all levels of the PDN in order to reduce voltage fluctuations. The parasitics of the capacitors are also known as *equivalent series inductance* (ESL) and *equivalent series resistance* (ESR). All the PDN parasitics interact with each other, forming LC circuits with different resonance frequencies, which are responsible for the *voltage droops*.

The circuit composed by the on-chip capacitance and the power bumps inductance (L_{bump}) generates the *first droop*, which typically produces the largest voltage noise and has a resonance frequency of 100-400MHz [119]. The second droop is controlled by C_{pkg} and L_{pkg}, and the third droop is dominated by C_{pcb} and L_{pcb}. Note that the second and third droops usually have much lower resonance frequencies and amplitudes than the first droop.

Fig. 2.12(a) shows a frequency response typically generated by the PDN of commercial high-performance integrated circuits, with the impedance and the resonance frequency for the first, second and third droops. The supply voltage behavior illustrated in Fig. 2.12(b) is observed when a current spike is requested for this PDN: the first droop causes fast and large voltage swings in the order of ns; then the voltage continues to fluctuate due to second and third droops, until it becomes stable after a few μs.

Voltage noise is minimized when the activity takes place at frequencies with low impedance associated. Fig. 2.13(a) shows the supply noise and the



(a) Typical voltage noise (b) Worst voltage noise

Figure 2.13: Voltage droops generated by periodical current differences at (a) low and (b) high impedance frequencies.

clock signal for a circuit operating at 1GHz (low impedance), with voltage swings of ±10%. The clock can be set to the frequency of the first droop to emulate the worst-case voltage noise, as seen in Fig. 2.13(b). In this case, the voltage noise amplitude goes up to 20%. Such large fluctuations are also known as *voltage emergencies*.

For designers, it is difficult to anticipate whether voltage emergencies will actually show up in their designs. Very often, they just *conjecture* that these events will not happen, without a full guarantee of safety. Note that a circuit designed for an application can be used for other purposes, with changes in the operating frequency, the submodules activated, the firmware, and the packaging. In this context, it is very difficult to predict the presence of such large voltage fluctuations. Still, if a voltage emergency occurs, then a timing failure may be originated and the circuit operation becomes unpredictable.

2.2.3 Ring Oscillator Clocks

Jitter and other clock uncertainties are generally considered by increasing the timing margins of the clock period, degrading circuit performance. For that reason, the use of *ring oscillator clocks* (ROCs) as clock sources has been discarded, as they have a high jitter caused by their sensitivity to the various sources of variability. Therefore, rigid clock generators with low-jitter, such as *phase-locked loops* (PLLs), became the *de facto* clock source paradigm.

Figure 2.14 shows a synchronous circuit fed by a PLL or by an ROC, depending on the selection of a multiplexer. Figure 2.15 illustrates the clock signals generated by the PLL and the ROC when a voltage droop occurs. Note that the clock period of the PLL is not affected by the changes in the

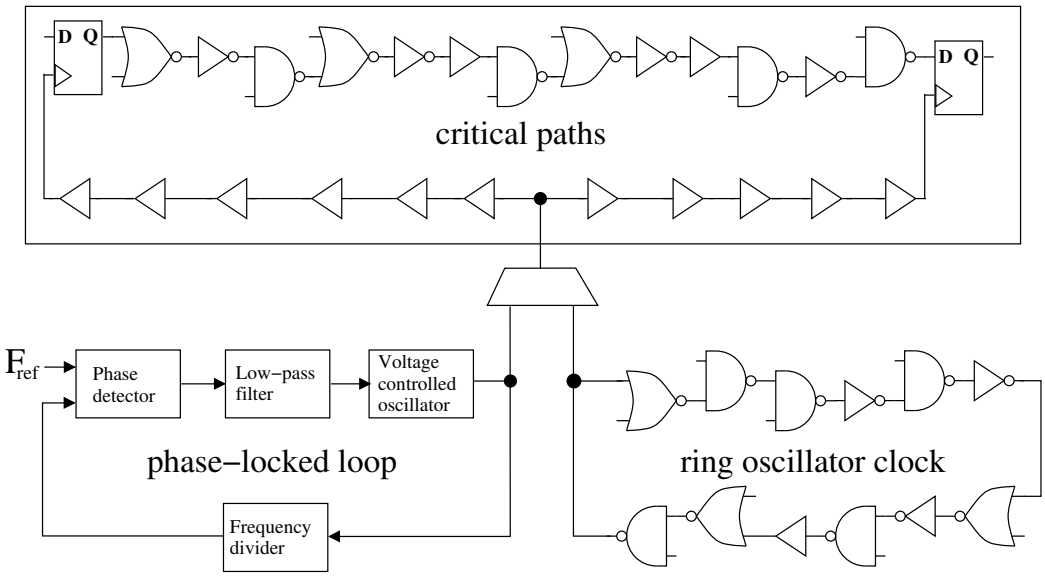


Figure 2.14: Synchronous circuit with a PLL or an ROC as the clock source.

voltage source, as it is designed to support such variations and deliver a low-jitter clock. However, the circuit paths have a different behavior: their delay increases when the voltage decreases. When the PLL is selected as the clock source, timing failures are avoided by adding margins that consider the delay of the critical paths at the *minimum estimated voltage*.

On the other hand, the period of the ROC is affected by the voltage variation, as seen in Fig. 2.15. In [98], it is shown that the power supply is the dominant source of jitter for ring oscillators. Recent studies demonstrate that the jitter of ROCs is highly correlated with the delay variability of the circuit paths [28, 29].

In other words, the *process, voltage, temperature* (PVT) variations suffered by the ROC are perceived by the circuit paths in an analogous way, as they are composed of similar logic gates. For example, when the circuit path becomes slower due to a voltage droop or a temperature increase, the frequency of the ROC slows down as well. This correlation between the jitter of ROCs and the circuit delay variations enables the reduction of timing margins, and hence improve circuit performance or reduce power [28, 29]. Note that the jitter of a PLL does not have similar correlation with the circuit delay.

Obviously, there is not an exact match between the delay of the critical paths and the period of an ROC. Standard cells have different responses to PVT variations. Additionally, there are voltage and temperature differences

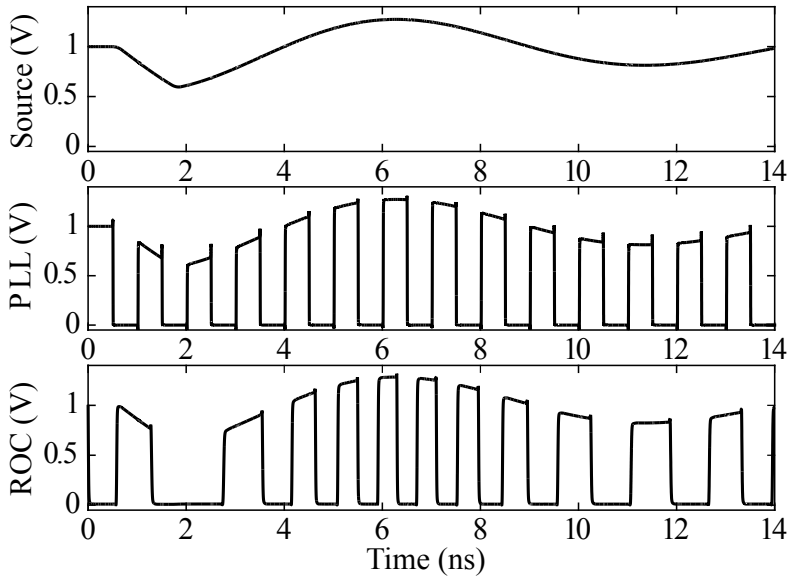


Figure 2.15: PLL and ROC clock generation in the presence of voltage noise.

across the chip, and process variability is not identical throughout the die [2]. Moreover, unknown or not well-understood issues must be covered, such as aging and radiation. Notwithstanding, the most significant variations are strongly correlated between the critical paths and the ROC.

In this work, a rigid clock source (PLL) is compared with an ROC, that is implemented using to the guidelines described in [28]. In summary, the design of an ROC consists of:

- Delay extraction of the critical paths of the circuit.
- Use the delay data extracted to create a path of library gates with similar delay behavior, considering all corners.
- Organize these gates in a ring to generate an oscillating signal.

The delay extraction is performed for all PVT corners available in the technology, using *static timing analysis* (STA) tools. The extracted delays are the input to a path synthesizer tool, which produces a single chain of standard cells that is able to produce an oscillating signal, i.e. a clock. Note that the design of an ROC depends only on the manufacturing technology and the variability behavior. Hence, ROCs are agnostic to the characteristics of the chip or the package.

Chapter 3

AIG Optimization via Boolean Decomposition

Restructuring techniques for *and-inverter graphs* (AIGs), such as rewriting and refactoring, are powerful, scalable and fast, achieving highly optimized AIGs after few iterations. However, these techniques are biased by the original AIG structure and limited by single output optimizations. This chapter investigates AIG optimization for area, exploring how far Boolean methods can reduce AIG nodes through local optimization. Boolean division is applied for multi-output functions using two-literal divisors and Boolean decomposition is applied as a method for AIG optimization [66]. Multi-output blocks are extracted from the AIG and optimized, achieving a further AIG node reduction of 7.8% on average for a subset of ITC99 and MCNC benchmarks.

3.1 Motivation

Recently, technology-independent algorithms based on AIGs have been proposed, enabling efficient and scalable optimizations. Restructuring methods such as refactoring [79], rewriting [84], and balancing [27] are powerful, and obtain highly optimized AIGs after few iterations. Still, these techniques are constrained by single output transformations, and iterations with technology mapping [82, 37] are often used to improve structurally biased results.

AIG rewriting and refactoring perform local transformations, extracting the local context with K-cuts [95], windows or *maximum fanout-free cones* (MFFCs). K-cuts can be considered a superior method to extract local context compared to windowing [79, 37], as it is possible to *control the support* of the Boolean functions to be optimized, while identifying a region of the circuit that *depends on this support*.

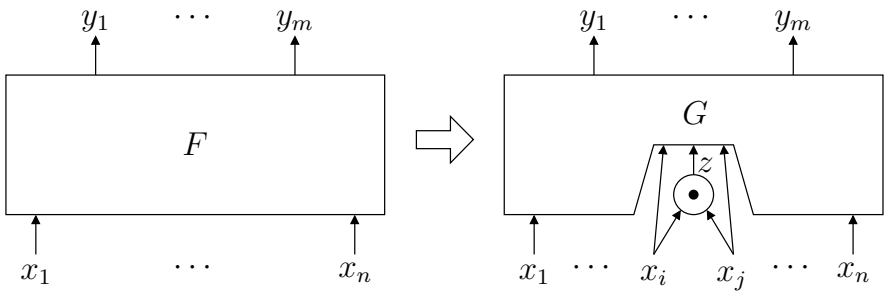


Figure 3.1: Decomposition using a two-literal Boolean divisor.

Algorithms based on K-cut enumeration have been proposed, such as factor cuts [20] and priority cuts [86], reducing the search space and enabling cuts with more nodes and inputs. Also, *multi-output* blocks based on K-cuts were presented [75, 69], extracting the *complete local context*.

This work studies technology-independent transformations that reduce the AIG size by exploring the use of Boolean decomposition. This is done by expanding the idea of two-literal divisors [90] to multi-output functions (see Fig. 3.1). The principle is as follows. A multi-output function

$$(y_1, \ldots, y_m) = F(x_1, \ldots, x_n)$$

can be decomposed into another multi-output function

$$(y_1, \ldots, y_m) = G(x_1, \ldots, x_n, z),$$

with z being a two-literal divisor ($z = x_i \odot x_j$), and $\odot$ being a Boolean operator (such as an AIG node). G is supposed to be a function with a simpler implementation than F, which can be obtained by Boolean division. A multi-output Boolean function can be recursively decomposed using this paradigm, and the result can be represented as an AIG network.

The main purpose of this work is to explore how far Boolean decomposition can optimize beyond the existing AIG rewriting methods. Unfortunately, scalability is an important issue when dealing with Boolean methods. Obviously, collapsing large networks into one-node functions and then decomposing is not computationally affordable.

This work takes a significant leap forward regarding [90], with the following contributions:

- Boolean decomposition with two-literal divisors is generalized to circuits with multiple outputs, instead of single-output functions.
- The selection of divisors is customized to increase the logic shared among multiple outputs.

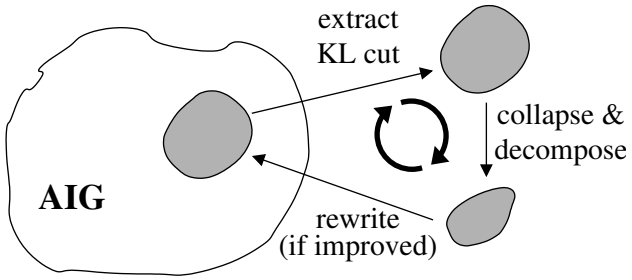


Figure 3.2: Iterative rewriting of KL-cuts on an AIG.

- A set of filters to reduce the search space is proposed.
- Scalability is addressed by iteratively applying Boolean decomposition to KL-cuts [75, 69] of the AIG (see Fig. 3.2).

In [80], a resynthesis method based on Boolean decomposition is performed on *look-up table* (LUT) networks by identifying and decomposing MFFCs. However, [80] only performs a simplified version of *disjoint-support decomposition* (DSD). In [104], windows are enumerated and don't care conditions identified in the network are used to simplify these windows, whereas in [106] the algebraic decomposition method is improved by assigning previously calculated don't cares using a set of rules. In this work, there is neither search nor previous calculation of don't care conditions, as the optimization occurs solely inside the KL-cut logic.

Notice that a KL-cut represents a portion of the circuit that depends on the same set of variables. The *key idea* is to take advantage of this property of KL-cuts in order to identify divisors that are useful for multiple functions, sharing more logic and reducing area. The results reported in this work have been often obtained by applying computationally intensive methods, e.g., many divisors, many cuts. Bear in mind that the goal now is to establish the bounds potentially reachable by future work that could use smart *oracles* to drive the search. Some preliminary criteria are discussed in Section 3.3.3. Still, the method proposed in this work may be interesting for highly repetitive structures or area-critical components.

3.2 Overview

This section introduces an overview of the contributions proposed, with an example of Boolean decomposition for a multiple-output function, and the optimization of a small benchmark. Additionally, a summary of results achieved with existing AIG transformation methods is presented.

3.2.1 Multi-output Boolean decomposition

Boolean decomposition is known to obtain good-quality results at the expense of a high computational cost. Finding good divisors is the most challenging task. Different approaches can be proposed to prune the search, e.g., use two-literal divisors [90], consider polarity information to ignore unpromising divisions, or reuse algebraic factored forms to select divisors. Considering all these simplifications, is it still possible to obtain good results?

To illustrate the decomposition, consider the set of functions in (3.1). Any combination of two literals of F can be selected as a Boolean divisor. The number of literals is defined as the cost function, which is 12 in this case.

$$F = \begin{cases} F_1 = bx + \bar{c}\bar{x} \\ F_2 = ax + d\bar{x} \\ F_3 = ax + be \end{cases} \qquad DC = \bar{a}bx + a\bar{b}x + ab\bar{x} \tag{3.1}$$

A reduction of 3 literals is achieved by performing a multi-output Boolean division using the divisor $y = ab$, generating the new set of functions and DC-set in (3.2). Note that this divisor is not easily extracted from the set of functions F: F_1 does not have the variable a and F_2 does not have the variable b. Still, all functions are reduced in 1 literal due to the effective use of the DC-set. For further decomposition steps, it is possible to perform a DC-set projection, removing variables from the DC-set that are not in the support of F, and decreasing the computational effort of the Boolean division. By projecting DC' to the support of F', $DC_{proj} = \bar{b}y$ is obtained.

$$F' = \begin{cases} F'_1 = y + \bar{c} \cdot \bar{x} \\ F'_2 = y + d\bar{x} \\ F'_3 = y + be \end{cases} \qquad DC' = DC + (y \oplus (ab)) \tag{3.2}$$

3.2.2 AIG optimization example

In order to demonstrate the benefits of the approach introduced in this work, the circuit *b06* of the ITC99 benchmarks suite [4] is optimized. The flow and the results for the different solutions are depicted in Fig. 3.3.

The input circuit is a Boolean network represented in a *Berkeley logic interchange format* (BLIF) file. An AIG with 42 nodes is obtained after decomposing the Boolean network with algebraic factorization and structural hashing (*strash* command in ABC [14]). After iteratively applying algebraic transformations using *dc2* command, the number of nodes is reduced to 35.

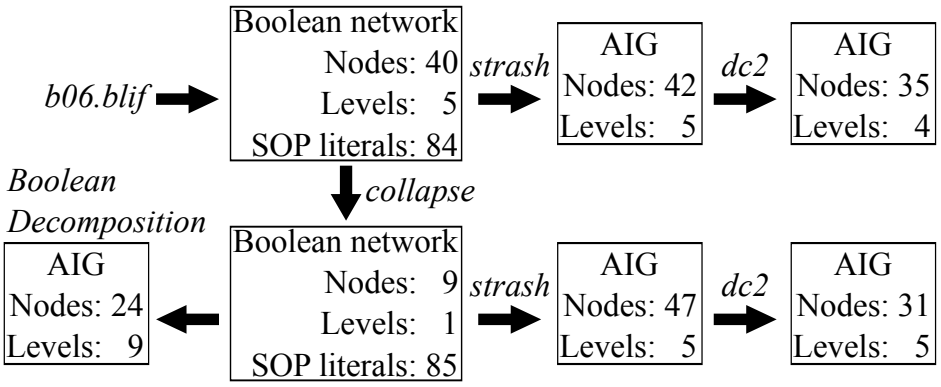


Figure 3.3: Optimization flow using different methods for *b06*.

An alternative approach would start by collapsing the initial network, which results in a Boolean network with one node for each output. Decomposing these nodes results in a larger AIG with 47 nodes, but applying iterative algebraic transformations reduces it to 31 nodes.

The method proposed in Section 3.3 applies Boolean decomposition on top of the collapsed network using an approach inspired by [90]. In this work, multi-output decomposition is used by iteratively selecting the Boolean divisor that minimizes the sum of the literals of the functions factored forms [13]. This approach is able to achieve a better logic sharing, obtaining an AIG with only 24 nodes. Also, a set of filters is applied to reduce the search space of divisors. By reducing the amount of two-level minimizations compared to [90], runtime is reduced 40 times on average, without sacrificing the quality of the results. Note that the results in this work cannot be directly compared with [90], which only presents the decomposition of single-output functions and it is not applied for circuit optimization.

3.2.3 Results obtained via AIG transformations

This section presents the AIG size reduction achieved by AIG transformations for a set of ITC99 and MCNC benchmarks [4]. Each benchmark is read and transformed into an AIG through algebraic factorization. Then structural hashing is performed, obtaining the number of AIG nodes and levels shown in column "*Initial*" of Table 3.1.

In order to obtain a highly optimized AIG, the *dc2* command is executed iteratively until no changes are observed. This reduces the number of nodes for the majority of cases, as seen in column "*After dc2*". The number of levels is usually reduced, but not for all cases. Geometric mean I and the ratios 1 and 2 refer to the complete set of benchmarks.

Table 3.1: Results obtained through AIG transformations.

Name	Initial		(a) After *dc2*		(b) After collapse + *dc2*		Diff nodes (a) and (b)	Best of (a) and (b)	
	Nodes	Levels	Nodes	Levels	Nodes	Levels		Nodes	Levels
b04	546	24	487	21	871	16	78.85%	487	21
b05	830	54	459	23	4095	19	792.16%	459	23
b06	42	5	35	4	31	5	-11.43%	31	5
b07	365	27	331	22	566	22	71.00%	331	22
b08	155	20	131	15	119	9	-9.16%	119	9
b09	136	12	123	10	197	12	60.16%	123	10
b10	180	11	162	9	175	8	8.02%	162	9
b11	611	28	452	21	1085	18	140.04%	452	21
b12	1002	17	947	14	1399	13	47.73%	947	14
b13	261	12	220	11	224	10	1.82%	220	11
b14	6069	60	3924	100	-	-	-	3924	100
b15	8432	65	7030	95	-	-	-	7030	95
alu4	2654	14	1573	15	625	14	-60.27%	625	14
apex2	2960	17	991	17	142	14	-85.67%	142	14
bigkey	3081	10	2847	10	3302	10	15.98%	2847	10
clma	11938	40	4842	38	527	16	-89.12%	527	16
diffeq	2575	40	2137	41	-	-	-	2137	41
ex1010	7681	17	4664	15	2337	14	-49.89%	2337	14
ex5p	1731	15	928	19	204	8	-78.02%	204	8
i10	3675	50	1637	36	-	-	-	1637	36
misex3	2454	13	1267	15	754	14	-40.49%	754	14
pdc	7757	19	3219	19	1717	18	-46.66%	1717	18
seq	2780	14	1373	13	1516	16	10.42%	1373	13
spla	6660	19	2298	16	525	14	-77.15%	525	14
tseng	1927	47	1763	41	-	-	-	1763	41
Geomean I	1409.43	21.26	921.52	19.48	-	-	-	614.49	17.56
Ratio 1	1.00	1.00	0.65	0.92	-	-	-	0.44	0.83
Ratio 2	-	-	1.00	1.00	-	-	-	0.67	0.90
Geomean II	1090.95	17.03	698.38	14.93	550.87	12.78	-	420.83	13.11
Ratio 3	1.00	1.00	0.64	0.88	0.50	0.75	-	0.39	0.77
Ratio 4	-	-	1.00	1.00	0.79	0.86	-	0.60	0.88

An alternative experiment is performed by collapsing the circuit after reading the input file. The collapsing operation does not finish for some benchmarks due to its complexity. In the cases the operation finishes, the AIGs are obtained through algebraic factorization and structural hashing. Then, *dc2* command is run iteratively until no changes are observed, generating the results shown in column "*After collapse + dc2*". Geometric mean II and the corresponding ratios 3 and 4 refer to the benchmarks in which collapsing could finish. The difference in the number of nodes varies from a 89% reduction (*clma*) to a 792% increase (*b05*).

```
1: procedure booleanDecompositionAIG(AIG, cutParams)
   Input: An AIG network, and parameters to enumerate KL-cuts
   Post: The AIG has equal or fewer nodes
2:     for each node N in AIG in topological order do
3:         for each kcut C of node N from kcut enumeration do
4:             obtain the klcut from C in AIG
5:             if the klcut is accepted based on cutParams then
6:                 F = set of klcut functions
7:                 DC = ∅ // DC-set of F functions
8:                 divisors = boolDecompose(F, DC)
9:                 if size of divisors < size of klcut then
10:                    new_klcut = AIG network of divisors
11:                    replace klcut by new_klcut in AIG
12:                    restart kcut enumeration
```

Figure 3.4: AIG optimization using Boolean decomposition.

Collapsing a Boolean network may result in a larger AIG (see Section 3.2.2), and the AIG transformations may not obtain the same results as before collapsing, as these modifications are biased by the structure. In the *b05* benchmark, the initial description has very good logic sharing between outputs, which is lost after collapsing. The shared logic is not recovered due to local optimization limitations, such as single-output transformations, some outputs depending on a large set of inputs. For several other cases, collapsing enables significant AIG reduction by removing redundant structures.

3.3 AIG optimization approach

This section introduces a new AIG optimization approach, based on Boolean decomposition with two-literal divisors [90]. Boolean methods are known to be time-consuming and not scalable, but also to obtain better results when compared with algebraic methods. In this work, the Boolean decomposition method using two-literal divisors [90] is applied to multi-output functions, and the runtime is reduced without losing quality of results. Still, the algorithm does not scale for large circuits, and it is applied via local optimization.

3.3.1 Local optimization using KL-cuts

A pseudo-code of the AIG optimization strategy is presented in Fig. 3.4. The procedure booleanDecompositionAIG receives an AIG and parameters to enumerate the KL-cuts, defining the limits for nodes and inputs, for example. After the execution, the AIG has equal or fewer nodes.

The method traverses the AIG in topological order (from the outputs to the inputs), and all K-cuts are enumerated for each node, based on the parameters *cutParams* (line 3). In line 4, the KL-cut is derived from the K-cut (as described in Section 2.1.4.3) and if it is accepted based on the parameters (line 5), then the Boolean decomposition is performed on the Boolean functions of the KL-cut (line 8). It is important to note that the method called in line 8, and explained in Section 3.3.2, could be replaced by any other method based on the minimization of multiple output functions.

If the result of the decomposition is smaller than the number of nodes in the KL-cut, then the KL-cut logic is replaced on the AIG (line 11). Note that boolDecompose returns the set of two-literal divisors used to decompose the function of the KL-cut, which can be easily translated to an AIG network. Additionally, as the AIG is modified when there is a KL-cut replacement, the previous K-cut enumeration has to be restarted (line 12).

3.3.2 Boolean decomposition

The algorithm for boolDecompose is presented in Fig. 3.5, which recursively performs Boolean decomposition on a set of functions F. The algorithm is divided into four steps: detection of trivial cases (I), generation of candidate Boolean divisors and definition of the cost function (II), selection of the best divisor (III), and preparation of the next recursive call (IV).

At line 3, detection of trivial cases (I) is performed, identifying when all functions in F are decomposed. The algorithm is executed recursively until this condition is satisfied. Step II starts by obtaining the algebraic factored form for each function in F (line 9). The cost function to be minimized is defined as the sum of literals of all functions in the factored form (*numLiterals* at line 10). Then, the two-literal leaves of the factored form trees are selected as candidate divisors for each function in F (see Fig. 3.6).

Boolean division is performed for all functions in F using each divisor in D, in order to calculate the cost function for all divisors. The best divisor (III) is the one that achieves the largest reduction in number of literals. As explained in Section 2.1.3.2, Boolean division is performed by adding the *satisfiability don't care* (SDC) of a divisor to the DC-set of a function and running two-level minimization. Notice that the DC-set may contain variables not relevant to the division, therefore a DC-set projection is performed to the support of the function f and the divisor d (line 20). The projected DC-set is accumulated with the SDC generated by the evaluated divisor (line 21), generating the DC-set used in the two-level minimization (DC_{div}). The division may also be avoided depending on the analysis of the variables polarity (see Section 3.3.3).

```
1: function boolDecompose(F, DC)
   Input: A set of functions F and their respective DC-set DC
   Return: A set of divisors that form an AIG
   // Each recursive call defines one divisor
2:     // Step I - If all functions are decomposed, return
3:     if all functions f ∈ F are trivial then return
4:
5:     // Step II - Generate the candidate Boolean divisors
6:     numLiterals = 0      // Number of literals of all functions in F
7:     D = ∅                // Set of two-literal divisors considered
8:     for each non-trivial f ∈ F do
9:         a = algebraic factored form of f
10:        numLiterals += number of literals in a
11:        D = D ∪ {two-literal leaves of a}
12:
13:    // Step III - Perform Boolean division for each divisor
14:    x = new variable  // Variable of the new divisor to be selected
15:    bestDivisor = ∅
16:    for each divisor d ∈ D do
17:        divLiterals = 0          // Number of literals after division
18:        for each non-trivial f ∈ F do
19:            // DC(f) is the DC-set of f
20:            DC_proj = DC(f) projection onto support of f and d
21:            DC_div = DC_proj + (x ⊕ d)
22:            Flits = Rlits = number of literals in f factored form
23:            if d is accepted based on f vars polarities then
24:                // R(f, d) is the result of the division of f by d
25:                R(f, d) = twoLevelMinimization(f, DC_div)
26:                Rlits = literals in R(f, d) factored form
27:            if Rlits > Flits then
28:                // If R(f, d) has more literals, or division was not
29:                // performed, select f as part of the solution
30:                R(f, d) = f
31:                divLiterals += Flits
32:            else
33:                divLiterals += Rlits
34:        // If division reduced literals, update best result
35:        if divLiterals < numLiterals then
36:            for each non-trivial f ∈ F do
37:                bestR(f) = R(f, d)
38:            bestDivisor = d
39:            numLiterals = divLiterals
40:
41:    // Step IV - Set the functions for next recursive call
42:    for each non-trivial f ∈ F do
43:        newF(f) = bestR(f)
44:        newDC(f) = DC(f) + (x ⊕ bestDivisor)
45:    // Return the best divisor and perform a new recursive call
46:    return {bestDivisor} ∪ boolDecompose(newF, newDC)
```

Figure 3.5: Boolean decomposition procedure.

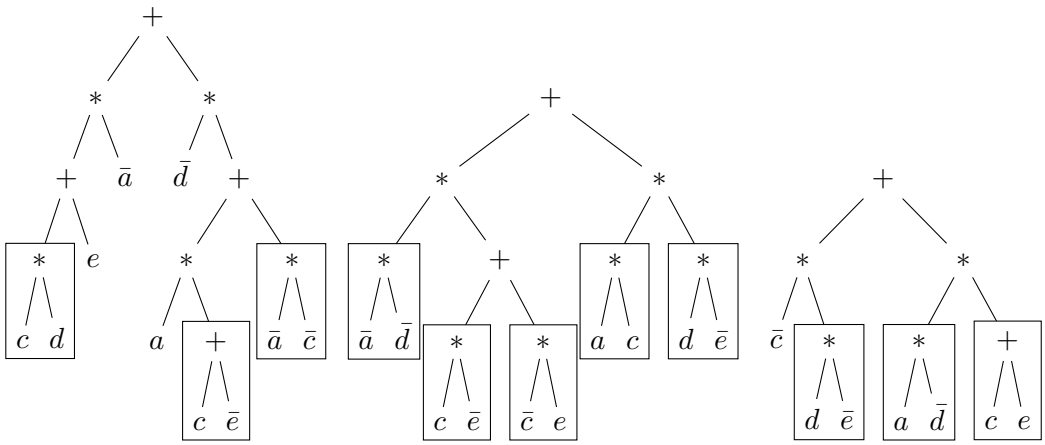


Figure 3.6: Factored form trees from *b06* benchmark.

If the division is avoided, or if the number of literals of the Boolean division result ($R(f, d)$) is larger than the ones for f (line 27), then f is used as part of the current solution (line 31). If the minimum number of literals is reduced after a division (line 35), the best solution is updated: the division result (line 36), the divisor and the number of literals (line 39). The next iteration is prepared at step IV. The ON-set (line 43) and the DC-set (line 44) obtained by the best divisor are used in the next recursive call (line 46).

Notice that the two-level minimization could be performed for the whole multi-output function. However, running single output two-level minimizations is preferred as it is more efficient, divisions can be filtered based on the variables polarities, and the divisions that increase literals can be discarded for each output individually.

3.3.3 Filters to reduce runtime

Select divisors from factored forms. In [90] all possible pairs of variables and polarities are considered for Boolean division. In this work, only pairs of literals obtained from the leaf nodes of the factored form trees are selected as potential divisors. The divisors are derived from all output functions of the KL-cut. For the benchmarks analyzed with our method, the quality of results was not affected by applying this filter, while the optimization runtime was significantly improved. In order to illustrate the divisor selection, factored form trees obtained from the output functions of *b06* benchmark are depicted in Fig. 3.6. The two-literal leaves highlighted in Fig. 3.6 are the divisors selected for Boolean division. Notice that only one polarity is investigated, e.g., if the divisor $c + \bar{e}$ is chosen, its negated version $\bar{c} \cdot e$ is disregarded.

Table 3.2: Divisors accepted based on the divided function f.

Divisor variables a and b w.r.t. f	Divisors accepted
$a \notin f$ support or $b \notin f$ support	None
a is binate or b is binate	$(a \cdot b)$, $(\overline{a} \cdot \overline{b})$, $(a \cdot \overline{b})$, $(\overline{a} \cdot b)$
a and b have the same polarity	$(a \cdot b)$, $(\overline{a} \cdot \overline{b})$
a and b have different polarities	$(a \cdot \overline{b})$, $(\overline{a} \cdot b)$

Use variable polarity information. This filter is applied to avoid exploring divisions with unpromising polarities between the divisor and the divided function. Table 3.2 describes the divisors that are accepted based on its support and the polarities of the variables in the divided function. A total of 849 Boolean divisions are performed during the Boolean decomposition for the benchmark *b06* when using this filter versus 943 without it (and 13829 divisions would be done without any filter). The polarity of the variables can be obtained using the concept of unateness, which is defined for completely specified functions. Unateness can only be used in the first iteration of Boolean decomposition, when the DC-set is empty. After the first iteration, the DC-set contains the SDCs of the previously selected divisors, and the concept of containment [116] must be used (see Section 2.1.1.2).

3.4 Experimental results

Table 3.3 shows the AIGs metrics (nodes and levels) before and after Boolean Decomposition. Column "*Initial*" reports the metrics of the AIGs after input and structural hashing, and column "*ABC smallest*" reports the AIGs with least number of nodes from Table 3.1 (column "*Best of (a) and (b)*"). The experiments were run on an Intel Core i7 processor with 4GB of RAM. All AIGs passed formal verification using ABC command *cec*.

The AIG optimization via Boolean decomposition is applied on the AIGs of column *(a) ABC smallest*. KL-cuts with K=8 and unbounded L are enumerated in order to obtain smaller parts of the AIG with complete local context. Also, the number of nodes of the KL-cuts is restricted to 30, therefore having a very limited scope of optimization. Boolean decomposition with two-literal divisors is performed on the KL-cut outputs functions, only replacing the KL-cut logic if the number of nodes is reduced. The column "*(b) Boolean Decomp.*" reports the results obtained after performing two iterations of the Boolean decomposition method. It is possible to reduce the number of AIG nodes in 7.76% on average, with important results such as 25.81% (*b06*), 16.95% (*spla*), 15.84% (*bigkey*) and 14.8% (*clma*), which

Table 3.3: AIG results of Boolean decomposition.

Name	Initial		(a) ABC smallest		(b) Boolean Decomp.		Runtime (s)	Diff. nodes (a) and (b)
	Nodes	Levels	Nodes	Levels	Nodes	Levels		
b04	546	24	487	21	442	21	222	-9.24%
b05	830	54	459	23	409	29	140	-10.89%
b06	42	5	31	5	23	9	0.55	-25.81%
b07	365	27	331	22	320	27	125	-3.32%
b08	155	20	119	9	113	10	2	-5.04%
b09	136	12	123	10	117	11	4	-4.88%
b10	180	11	162	9	156	10	14	-3.70%
b11	611	28	452	21	427	24	178	-5.53%
b12	1002	17	947	14	920	15	212	-2.85%
b13	261	12	220	11	207	12	3	-5.91%
b14	6069	60	3924	100	3810	120	14421	-2.91%
b15	8432	65	7030	95	6656	107	14592	-5.32%
alu4	2654	14	625	14	570	16	268	-8.80%
apex2	1960	17	142	14	128	15	5	-9.86%
bigkey	3081	10	2847	10	2396	15	2209	-15.84%
clma	11938	40	527	16	449	17	46	-14.80%
diffeq	2575	40	2137	41	2015	51	407	-5.71%
ex1010	7681	17	2337	14	2306	15	2004	-1.33%
ex5p	1731	15	204	8	197	8	127	-3.43%
i10	3675	50	1637	36	1530	37	882	-6.54%
misex3	2454	13	754	14	731	14	288	-3.05%
pdc	7757	19	1717	18	1543	18	1371	-10.13%
seq	2780	14	1373	13	1320	16	626	-3.86%
spla	6660	19	525	14	436	15	211	-16.95%
tseng	1927	47	1763	41	1696	42	445	-3.80%
Geomean	1391.5	21.2	614.4	17.5	566.7	19.9	147.4	-7.76%
Ratio 1	1.000	1.000	0.442	0.826	0.407	0.939	-	-
Ratio 2	-	-	1.000	1.000	0.922	1.137	-	-

have an important level of sharing between primary outputs. Notice that the worst result obtained is for the benchmark *ex1010*, which is consisted of 10 independent output functions with very little logic sharing between outputs.

Our approach is able to identify a better logic sharing, therefore increasing the number of levels, which is not controlled by the method proposed. Still, there is an increase of up to 1 level for 15 out of 25 benchmarks evaluated. Also, the average number of levels is still smaller than the *Initial* results.

Table 3.4 presents technology mapping performed for *field-programmable gate arrays* (FPGAs) and standard cells using the AIGs from Table 3.3. Area reduction was observed simply by changing the input by AIGs with fewer nodes. Mapping to LUTs was performed with the ABC command "*if -K 4*", obtaining 5.5% area reduction on average. Mapping to standard cells was per-

Table 3.4: Technology mapping results.

Name	FPGA (LUT4)				Standard cell (μm^2, ns)			
	(a) ABC		(b) Bool. Dec.		(a) ABC		(b) Bool. Dec.	
	Nodes	Levels	Nodes	Levels	Area	Delay	Area	Delay
b04	183	7	177	8	14041	1879	12735	1833
b05	222	9	201	11	14350	2057	12687	2545
b06	17	2	17	2	1191.9	567	993	754
b07	145	8	146	8	9828	1949	9645	2205
b08	55	4	50	5	3659	873	3460	934
b09	55	4	53	5	3554	876	3716	1060
b10	73	5	75	5	5076	881	4893	957
b11	179	7	175	8	13215	1916	12551	2086
b12	452	6	455	6	28188	1322	27922	1409
b13	95	4	94	4	6890	975	6482	1046
b14	1608	33	1544	37	112586	8639	112277	10068
b15	3021	33	2922	33	204689	8110	195332	9398
alu4	293	7	277	7	18114	1308	16446	1450
apex2	65	6	61	6	4218	1300	3884	1398
bigkey	1254	3	921	4	81867	901	73250	1327
clma	277	6	249	6	15934	1446	13968	1512
diffeq	824	14	754	14	61892	3658	57616	4129
ex1010	1132	7	1118	7	66649	1314	65849	1415
ex5p	129	3	122	4	6356	784	6090	789
i10	724	14	688	14	47974	3222	45120.5	3187
misex3	363	6	363	6	21716	1222.7	21162.1	1218
pdc	904	7	821	7	50301	1585	45429	1592
seq	716	6	673	7	40718	1232	38957	1438
spla	262	5	222	6	15108	1218	12991	1303
tseng	758	13	741	13	50583	3598	49251	3561
Geomean	284.35	6.84	268.64	7.39	18356	1607	17217	1859
Ratio	1.000	1.000	0.945	1.081	1.000	1.000	0.938	1.157

formed with the ABC command "*map*" using the library "*GSCLib_3.0.lib*" from [4], obtaining an area reduction of 6.2% on average.

Chapter 4

Support-reducing Decomposition for FPGA Mapping

The cost functions of two-level or factored-form representations, e.g., literals, are used in most decomposition methods, as they have a high correlation with the area of cell-based designs. However, this correlation is weaker for *field-programmable gate arrays* (FPGAs) based on *look-up tables* (LUTs). Moreover, local optimizations have limited power due to the structural bias of the circuit descriptions. This chapter proposes the reduction of the structural biasing by remapping the LUT network and decomposing the derived functions using the support as cost function [68]. The proposed method improves the FPGA mapping results of a commercial tool for the 20 largest MCNC benchmarks, with gains of 28% in delay plus 18% in area when targeting delay, and a reduction of 28% in area plus 14% in delay with area as cost function. Results with 23% less area and 6% less delay are obtained after physical synthesis (post place-and-route). Furthermore, 12 of the best known results for delay (and 6 for area) of the EPFL benchmarks are improved.

4.1 Motivation

FPGAs are integrated circuits consisting of programmable logic blocks and interconnections. FPGAs can be reprogrammed multiple times, and have an extremely smaller initial cost and production time in comparison with *application specific integrated circuits* (ASICs). For those reasons, FPGAs are largely used for prototyping of ASICs and applications with low volume. However, when compared to ASICs, the flexibility given by FPGAs comes

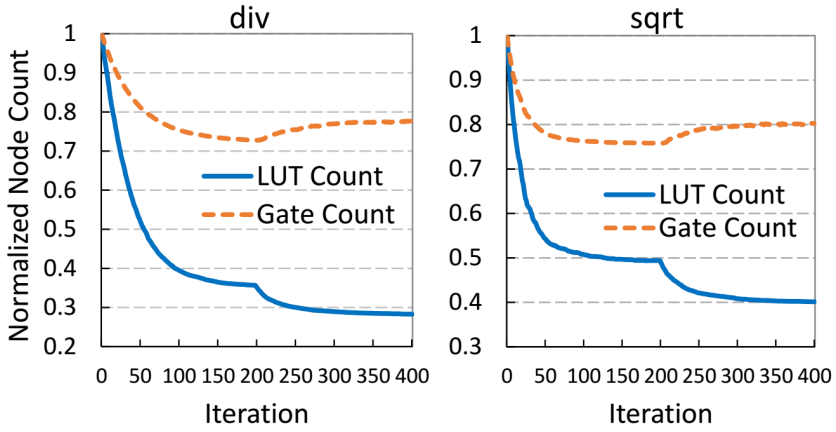


Figure 4.1: Correlation between the number of AIG nodes and LUTs after technology mapping (Source: [65]).

at the expense of larger area and power consumption, and lower performance [57]. Recently, FPGAs started to be employed in the optimization of specific tasks in data centers, with technology leaders making great efforts in hybrid solutions with ASICs and FPGAs [24, 100, 111].

The FPGA implementation process inherited many techniques from the ASIC design flow (see Fig. 1.2). The use of well-established methods enabled the fast growing and wide-usage of FPGAs, but these algorithms generally have cost functions customized for cell-based designs, in which the area is proportional to the number of transistors. Usual cost functions in logic synthesis are cubes in *sum-of-products* (SOP) forms, literals in Boolean function expressions, or nodes and levels of *and-inverter graphs* (AIGs). On the other hand, FPGAs based on LUTs are composed of logic blocks with k inputs (typically 4 to 6), and each LUT can implement any logic function of up to k inputs.

A study on this miscorrelation is presented in [65], showing that the reduction of nodes and levels in AIGs do not necessarily translate to fewer LUTs or less logic depth in the FPGA mapping derived. Two examples of this study are shown in Fig. 4.1, using perturbations in the logic network and accepting changes based on a cost function. In the first 200 iterations the objective is to reduce the number of AIG nodes, and the corresponding FPGA mapping has a reduced LUT count as well. For the following 200 iterations, the cost function is the amount of LUTs. However, the corresponding number of AIG nodes is increased.

There are several works on FPGA technology mapping based on cut-enumeration, performing a covering of the subject graph using k-cuts [26,

22, 86]. FlowMap [26] was the first approach to guarantee minimal depth for a given structure, and the algorithms just improved ever since. These cut-based techniques vary on the algorithms, parameters, and cost functions used for the cut-enumeration and covering. Nevertheless, the quality of the solution heavily depends on the structure of the subject graph.

A second group of works rely on *binary decision diagrams* (BDDs) to perform FPGA mapping [61, 114, 23, 56, 110]. BDDs typically provide *per se* a good starting point for FPGA mapping, as the redundant variables are removed and the structure size is reduced. Also, BDDs enable the application of functional techniques, reducing the structural bias. However, the complexity of BDDs increases significantly with the number of variables, becoming computationally unfeasible for large designs. Thus, BDD-based methods are often applied to portions of the circuit (*partial collapsing*), but these methods are also structurally biased. This work proposes to combine these two groups of strategies, using both functional decomposition and cut-based mapping.

The idea of performing decomposition while reducing the support (and targeting FPGAs) has already been proposed. The support is minimized using don't cares in [105], as explained by [16]. In [62], it is proposed a complex decomposition aiming support minimization, by identifying the compatibility of all variables (or classes) in the bound-set. The decomposition proposed in [62] is applied in BoolMap [61]. Our work proposes the restructuring of the LUT network using the support size as cost function, with the aid of simple and fast decompositions.

The support-reducing techniques presented in this work are well-known methods, with the exception of the abstraction-based decompositions (see Section 4.3.5). Other decomposition methods could be considered, such as [62, 87, 60, 25], which are more time-consuming, but could improve the quality of results. Still, the *key idea* is to consider the support size as the cost function for decomposition, which restructures the subject graph targeting LUT-based FPGAs, and not the techniques incorporated.

In this thesis, a decomposition of a function F is considered *support-reducing* if the decomposing functions have their support size smaller than F. This definition differs from [54], which limits the term *support-reducing* to *disjoint-support decompositions* (DSDs).

This work proposes two main contributions:

1. A functional decomposition, which is guided by the support size, and it is based on simple and fast support-reducing techniques.

2. A recursive remapping approach, that reduces the structural bias of the subject graph, and uses the FPGA mapping metrics as cost function.

The proposed contributions are implemented into an FPGA remapping tool, named **S**upport-**R**educing Re**map**ping (**SR-map**). By remapping the results of a commercial tool for the 20 largest MCNC benchmarks, SR-map is able to reduce delay in 28% (plus 18% in area) when targeting delay, and improve area in 28% (plus 14% in delay) with area as cost function. The main reasons for these improvements are:

1. The FPGA mapping metrics are used to guide the resynthesis algorithm, instead of literals and cubes.

2. A new and aggressive collapsing strategy is applied, instead of a local *partial collapsing*.

3. A *new structure* is generated by a support-reducing decomposition.

The goal of using the mapping result as cost function is to reduce the miscorrelation between intermediate and final results, accepting transformations that will contribute to improve the final solution [65]. This is possible with fast and high-quality FPGA mapping algorithms [86].

BDD-based methods often rely on the partial collapsing of the subject graph [61, 114, 23]. The effectiveness of this process depends on the structure of the subject graph, which can easily reach a local minimum. This work performs a recursive global collapsing on the LUT network (see Section 4.4), with the goal of reducing the structural bias from the subject graph.

The support size as cost function makes sense for FPGAs: a k-input function with *any* number of literals can be implemented with a single LUT of k inputs. This concept is illustrated with an example in Section 4.2.

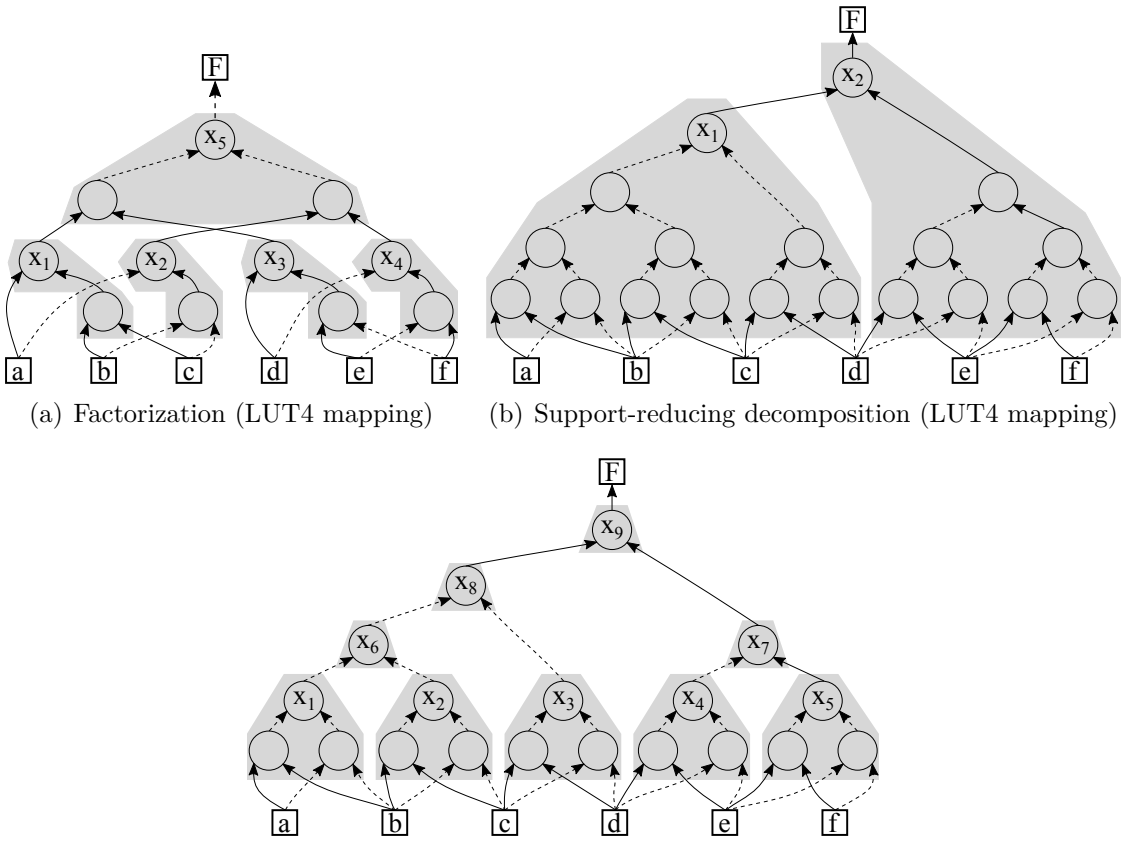
4.2 Motivating Example

In this section, an example is used to illustrate the support-reducing decomposition. Consider the following expression of a 6-input Boolean function:

$$F(a,b,c,d,e,f) = abcde\overline{f} + \bar{a}\bar{b}\bar{c}\bar{d}\bar{e}f \tag{4.1}$$

This expression has 12 literals, and it is also the optimal AND/OR factored form, as there is no other expression with fewer literals. The AIG shown in Fig. 4.2(a) is derived from the expression in (4.1). A structural FPGA mapping targeting LUTs with 4 inputs is also shown in Fig. 4.2(a), with the 5 shadowed regions representing the LUT covering of the AIG. The FPGA mapping derived by [86] is the following:

$$x_1 = abc,\ x_2 = de\overline{f},\ x_3 = \bar{a}\bar{b}\bar{c},\ x_4 = \bar{d}\bar{e}f,\ x_5 = x_1x_2 + x_3x_4 \tag{4.2}$$



(a) Factorization (LUT4 mapping) (b) Support-reducing decomposition (LUT4 mapping)

(c) Support-reducing decomposition (LUT2 mapping)

Figure 4.2: Functionally equivalent and structurally different AIGs, obtained via (a) algebraic factorization, and (b)(c) support-reducing decomposition.

By applying the proposed support-reducing decomposition on (4.1), the following expression (with 20 literals) is obtained:

$$F = (ab + \bar{a}\bar{b})(bc + \bar{b}\bar{c})(cd + \bar{c}\bar{d})(de + \bar{d}\bar{e})\overline{(ef + \bar{e}\bar{f})} \tag{4.3}$$

The AIG presented in Fig. 4.2(b) is derived from expression (4.3). This AIG has 8 more nodes than the one in Fig. 4.2(a). This means that it would likely result in a circuit with larger area, if implemented as a cell-based design. However, its mapping with 4-input LUTs has only 2 LUTs, whereas the one for Fig. 4.2(a) has 5. The FPGA mapping given by [86] for the AIG of Fig. 4.2(b) is the following:

$$x_1 = (ab + \bar{a}\bar{b})(bc + \bar{b}\bar{c})(cd + \bar{c}\bar{d}), \quad x_2 = x_1(de + \bar{d}\bar{e})\overline{(ef + \bar{e}\bar{f})} \tag{4.4}$$

Similarly, the FPGA mapping for LUTs of different sizes would require fewer LUTs for the AIG obtained by the support-reducing decomposition, as

Table 4.1: Comparison of the FPGA mapping for the AIGs obtained via algebraic factorization and support-reducing decomposition.

LUT size	Algebraic factorization		Support-reducing decomposition	
	LUTs	Levels	LUTs	Levels
2 inputs	11	4	9	4
3 inputs	6	3	4	2
4 inputs	5	2	2	2
5 inputs	3	2	2	2

shown in Table 4.1. The derived FPGA mapping is smaller for all cases, even for 2-input LUTs, which is illustrated in Fig. 4.2(c).

Note that a and $\bar{a}$ are different literals. This makes sense for cell-based designs, as each literal generally will result in a transistor. Still, a and $\bar{a}$ are the same in terms of support, as both refer to the variable a. Therefore, the *support* is the key cost function for the proposed decomposition, generating a structure more suitable for LUT-based FPGAs, even with larger AIGs.

4.3 Support-reducing decomposition

This section presents the proposed functional decomposition, based on simple and fast support-reducing techniques. It is a technology-independent decomposition, i.e., it is agnostic to the target FPGA technology. The goal is to generate a structure guided by the support size. The resulting subject graph typically produces a faster or smaller LUT network, but there are corner cases that poor results are obtained, e.g., multiplier. For this reason, the remapping approach in Section 4.4 selects the best result between the existing LUT network and the one derived by the decomposition.

The decomposition input is an *incompletely specified function* (ISF). It is possible to identify external don't care conditions and use them as input to the decomposition. However, in this work, the don't care conditions are identified only in the internal recursions of the method. The output of the decomposition is a Boolean network consisted of logic gates from the set *{AND2, OR2, XOR2, MUX21, AO21, AX21}*, which are required to implement the techniques considered.

Fig. 4.3 presents a pseudo-code of the algorithm. The trivial cases are checked at line 4. If the DC-set is not empty, then minimization is applied (line 7), updating F if the minimized function has a support smaller or equal to F. The minimization can be implemented by any method that accepts an

ISF, e.g., Espresso [15], BDD minimization [46, 109].

The decomposition method described in Fig. 4.4 is invoked at line 12, which receives an ISF as input and returns a solution consisting of a decomposing gate *op* and a set of functions $\langle F_1, ..., F_n \rangle$. If the solution has no disjoint support, then the *satisfiability don't care* (SDC) conditions are calculated by CALCSDC (line 15), which is implemented as in [32].

Each derived function is decomposed recursively, generating a Boolean network (line 17). The network obtained is connected to the related input in the decomposing gate (*op*) at line 18. Notice that the resulting network is also a tree, and the task of sharing logic is postponed to structural hashing and AIG optimizations (see Section 4.4).

The method in Fig. 4.4 performs several support-reducing techniques on the input function F, selecting the one with the lowest sum of support sizes, given that all functions in $\langle F_1, ..., F_n \rangle$ have a support size smaller than $|F|$. If several solutions are found, additional costs are considered (see Section 4.3.1). Other techniques could be incorporated [62, 87, 60, 25], which are slower but could improve the results. Still, the idea is to use simple and fast techniques that reduce the support, obtaining an efficient method that is able to produce good results by using the support size as cost function.

The support-reducing techniques considered are: (1) essential literals (lines 4-5), which is a *simple* and fast DSD; (2) trying to remove one variable from the support (line 9), using Shannon and Davio expansions (and their simplifications); (3) trying to remove two variables (line 11), with additional DSD techniques; and (4) a new bi-decomposition method, based on the universal and existential abstractions, applied to one and two variables.

4.3.1 Cost function

In this work, the cost function is the sum of support sizes of the derived functions, i.e., $\sum_{i=1}^{i=n} |F_i|$. Moreover, a solution is only accepted if all derived functions have a support size smaller than F, i.e., $\forall_i |F_i| < |F|$. Additionally, if there is more than one solution with the smallest sum of support sizes, then the following costs are considered, in this order:

1. The sum of squares of the BDD sizes [10], targeting a balanced solution, which favors delay reduction.

2. The gate implementation cost in CMOS transistors, e.g., an AND2 gate costs less than a MUX21 or an XOR2.

As BDDs are the representation of choice, the cost function exploits their structure to guide the decomposition, but similar costs could be derived for other representations.

```
1: function supportReduceDecomposition(F, DC)
   Input: An ISF, with the ON-set (F) and the DC-set (DC)
   Output: A Boolean network (tree) that implements the ISF
2:     // check for trivial cases (constants, variables)
3:     // the support size of F is denoted as |F|
4:     if |F| ≤ 1 then return F
5:     // if DC-set is not empty, then minimize F
6:     if DC ≠ ∅ then
7:         F_min = minimize(F, DC) // Espresso [15], BDD reduction [46]
8:         // accept F_min if support is reduced or the same
9:         if |F_min| ≤ |F| then F = F_min
10:    // perform decomposition
11:    // op ∈ {AND2, OR2, XOR2, MUX21, AO21, AX21}
12:    (op, ⟨F_1, ..., F_i, ..., F_n⟩) = decomposeFunction(F, DC)
13:    for each function F_i in ⟨F_1, ..., F_i, ..., F_n⟩ do
14:        // calculate SDC as in [32]
15:        DC_i = DC + calcSDC(op, i, ⟨F_1, ..., F_i, ..., F_n⟩)
16:        // decompose F_i recursively
17:        network = supportReduceDecomposition(F_i, DC_i)
18:        op.connect(i, network)          // connect network to gate input i
19:    return op                                              // root of the tree
```

Figure 4.3: Pseudo-code of the proposed support-reducing decomposition.

```
1: function decomposeFunction(F, DC)
   Input: An ISF, with the ON-set (F) and the DC-set (DC)
   Output: A decomposition (op, ⟨F_1, ..., F_i, ..., F_n⟩)
2:     Q = ∅                  // priority queue of potential solutions
3:     // check for essential literals
4:     decomposeEssentials(F, 1, Q)
5:     decomposeEssentials(F̄, 0, Q)
6:     // if essential literals found, return
7:     if Q ≠ ∅ then return best solution ∈ Q
8:     // check one-variable decompositions
9:     decomposeOneVariable(F, DC, Q)
10:    // check two-variable decompositions
11:    decomposeTwoVariables(F, DC, Q)
12:    return a solution with the lowest cost ∈ Q
```

Figure 4.4: Pseudo-code for an step of the support-reducing decomposition.

```
1: procedure DECOMPOSEESSENTIALS(F, P, Q)
   Input: Boolean function F, polarity P, priority queue of decompositions Q
   Post: decompositions added to Q
2:     E = ⟨e_1, ..., e_n⟩                          // set of n essential literals of F
3:     if E == ∅ then return
4:     H = F_{e_1...e_n}                            // cube-cofactor of F w.r.t. E
5:     if H ≠ 1 then
6:         G = (e_1 · ... · e_n)                    // AND of all essential literals
7:         // polarity P ∈ {0,1}
8:         if P == 1 then Q.add(G · H)                                   // AND2
9:         else Q.add(¬G + ¬H)                                           // OR2
10:    else
11:        G_1 = (e_1 · ... · e_{n/2})              // AND of essential literals 1 to n/2
12:        G_2 = (e_{n/2+1} · ... · e_n)            // AND of essential literals n/2+1 to n
13:        if P == 1 then Q.add(G_1 · G_2)                               // AND2
14:        else Q.add(¬G_1 + ¬G_2)                                       // OR2
```

Figure 4.5: Pseudo-code for decomposition using essential literals.

4.3.2 Essential literals

Fig. 4.5 describes the decomposition method using *essential literals*, i.e., literals that are common to all prime implicants. Decomposition with essential literals is checked first and preferred to the other techniques, as it is a fast DSD method which removes the *simple* part of the decomposition. For example, given $F(X)$ and $\{a, b, c\} \subseteq X$, if $\{a, b, \bar{c}\}$ are essential literals of $F(X)$, then F can be rewritten as $F(X) = (ab\bar{c})F_{ab\bar{c}}$. Similarly, given $G(X) = \overline{F(X)}$ and $\{x, y, z\} \subseteq X$, if $\{\bar{x}, y, z\}$ are essential literals of $G(X)$, then F can be decomposed as $F(X) = \overline{(\bar{x}yz)} + \overline{G_{\bar{x}yz}}$.

It is possible to check if a literal is essential by comparing the literals with the function F . For example, if F is smaller or equal to the function $H = a$ ($F \leq H$), then a is an essential literal of F. However, BDD packages [109] have more efficient methods to derive all essential literals of a function.

The essential literals of F are checked at line 8 ($P = 1$), and the one for $\overline{F}$ at line 9 ($P = 0$). If the function is solely composed of essential literals, i.e., the cube-cofactor w.r.t. to the essential literals is the constant 1 (F is a cube), then a balanced decomposition is performed (lines 13-14).

Example: Consider the function $F = \bar{a}\bar{c}(b(d + \overline{f}) + \bar{e})$, which has the essential literals $\{\bar{a}, \bar{c}\}$. By calculating the cube cofactor $F_{\bar{a}\bar{c}} = (b(d + \overline{f}) + \bar{e})$, it is possible to decompose the function $F = (\bar{a}\bar{c})(b(d + \overline{f}) + \bar{e})$. Consider another function $G = (ab + cd) + (e + f)$, which has no essential literals. The complement function $\overline{G} = H = (\bar{a} + \bar{b})(\bar{c} + \bar{d})\bar{e}\overline{f}$ has the essential literals $\{\bar{e}, \overline{f}\}$. The cube cofactor in this case is $H_{\bar{e}\overline{f}} = (\bar{a} + \bar{b})(\bar{c} + \bar{d})$, deriving the decomposition $G = \overline{(\bar{e}\overline{f})} + \overline{(\bar{a} + \bar{b})(\bar{c} + \bar{d})} = (e + f) + (ab + cd)$.

```
1: procedure DECOMPOSEONEVARIABLE(F, DC, Q)
   Input: ON-set (F), DC-set (DC), priority queue of decompositions Q
   Post: decompositions added to Q
2:     for each variable x_i ∈ support of F do
3:         if δF/δx_i == 1 then
4:             Q.add(x_i ⊕ F_{x̄_i})
5:             Q.add(x̄_i ⊕ F_{x_i})
6:             return
7:         if ∃x_i F == F_{x̄_i} then
8:             Q.add((x̄_i · F_{x̄_i}) + F_{x_i})                    // AO21
9:         else if ∃x_i F == F_{x_i} then
10:            Q.add((x_i · F_{x_i}) + F_{x̄_i})                    // AO21
11:        else // full Davio and Shannon expansions
12:            Q.add((x_i · δF/δx_i) ⊕ F_{x̄_i})                    // AX21
13:            Q.add((x̄_i · δF/δx_i) ⊕ F_{x_i})                    // AX21
14:            Q.add(x̄_i · F_{x̄_i} + x · F_{x_i})                  // MUX21
15:        // one-variable abstraction-based bi-decompositions
16:        if ∃x_i F ≠ 1 then
17:            G = ∃x_i F
18:            H = MINIMIZE(F, Ḡ + DC) // Espresso [15], BDD reduction [46]
19:            Q.add(G · H)                                        // AND2
20:        if ∀x_i F ≠ 0 then
21:            G = ∀x_i F
22:            H = MINIMIZE(F, G + DC) // Espresso [15], BDD reduction [46]
23:            Q.add(G + H)                                        // OR2
```

Figure 4.6: Pseudo-code for one-variable decompositions.

4.3.3 One-variable decompositions

The basic one-variable support-reducing decompositions are given by the Shannon expansion (2.1), and the Davio expansions (2.5-2.6). These methods isolate one variable, thus reducing the support size of the derived functions in at least one. Simplifications of these expansions can be obtained given specific conditions, as shown in (4.5). Essential literals cover the cases in which one of the cofactors is a constant.

$$\begin{aligned} F &= x_i \oplus F_{\overline{x_i}}, \quad \text{if } \delta F/\delta x_i = 1 \\ F &= \overline{x_i} \oplus F_{x_i}, \quad \text{if } \delta F/\delta x_i = 1 \\ F &= \overline{x_i} \cdot F_{\overline{x_i}} + F_{x_i}, \quad \text{if } \exists x_i F = F_{\overline{x_i}} \\ F &= x_i \cdot F_{x_i} + F_{\overline{x_i}}, \quad \text{if } \exists x_i F = F_{x_i} \end{aligned} \tag{4.5}$$

The Davio and Shannon expansions are added to the priority queue in the method described in Fig. 4.6 (lines 12-14). The simplifications listed in (4.5) are also checked (lines 4-5, 8 and 10) and preferred to the full Davio and Shannon expansions.

```
1: procedure DECOMPOSETWOVARIABLES(F, DC, Q)
   Input: ON-set (F), DC-set (DC), priority queue of decompositions Q
   Post: decompositions added to Q
2:     for each pair of variables x_i, x_j ∈ support of F do
3:         found = true                          // checks AND DSD condition
4:         if F_{x̄_i} == F_{x̄_j} then
5:             S = (x_i · x_j); G = F_{x_i x_j}; H = F_{x̄_i}
6:         else if F_{x̄_i} == F_{x_j} then
7:             S = (x_i · x̄_j); G = F_{x_i x̄_j}; H = F_{x̄_i}
8:         else if F_{x_i} == F_{x̄_j} then
9:             S = (x̄_i · x_j); G = F_{x̄_i x_j}; H = F_{x_i}
10:        else if F_{x_i} == F_{x_j} then
11:            S = (x̄_i · x̄_j); G = F_{x̄_i x̄_j}; H = F_{x_i}
12:        else found = false
13:        if found then
14:            if G == 0 then Q.add(S̄ · H)                    // AND2
15:            else if G == 1 then Q.add(S + H)               // OR2
16:            else Q.add(S̄ · H + S · G)                      // MUX21
               return
17:        // checks XOR DSD condition
18:        if δF/δx_i == δF/δx_j then
19:            S = (x_i ⊕ x_j), G = F_{x̄ȳ}, H = δF/δx_i
20:            if G == 0 then Q.add(S · H)                    // AND2
21:            else if G == 1 then Q.add(S̄ + H̄)               // OR2
22:            else if H == 1 then Q.add(S ⊕ G)               // XOR2
23:            else Q.add((S · H) ⊕ G)                        // AX21
               return
24:        // two-variable abstraction-based bi-decompositions
25:        if ∃x_i x_j F ≠ 1 then
26:            G = ∃x_i x_j F
27:            H = MINIMIZE(F, Ḡ + DC) // Espresso [15], BDD reduction [46]
28:            Q.add(G · H)                                   // AND2
29:        if ∀x_i x_j F ≠ 0 then
30:            G = ∀x_i x_j F
31:            H = MINIMIZE(F, G + DC) // Espresso [15], BDD reduction [46]
32:            Q.add(G + H)                                   // OR2
```

Figure 4.7: Pseudo-code for two-variable decompositions.

4.3.4 Two-variable decompositions

In [18], it is proposed the use of simple cofactor tests in order to perform disjoint-support decompositions. The cofactor tests and decompositions for AND and XOR are described in (4.6), given $F(X)$ and $\{x, y\} \subseteq X$. These tests are performed in the method of Fig. 4.7 (lines 4-10, and 18).

If one of the cube-cofactors in (4.6) is a constant, then simplifications can be derived (lines 14-15 and 20-22). If this is not possible, then a MUX21

gate is defined for the AND decomposition (line 16), and an AX21 gate for the XOR decomposition (line 23).

$$\begin{aligned}
F &= \overline{(xy)}F_{\bar{x}} + (xy)F_{xy}, && \text{if } F_{\bar{x}} = F_{\bar{y}} \\
F &= \overline{(x\bar{y})}F_{\bar{x}} + (x\bar{y})F_{x\bar{y}}, && \text{if } F_{\bar{x}} = F_{y} \\
F &= \overline{(\bar{x}y)}F_{x} + (\bar{x}y)F_{\bar{x}y}, && \text{if } F_{x} = F_{\bar{y}} \\
F &= \overline{(\bar{x}\bar{y})}F_{x} + (\bar{x}\bar{y})F_{\bar{x}\bar{y}}, && \text{if } F_{x} = F_{y} \\
F &= ((x \oplus y) \cdot \delta F/\delta x) \oplus F_{\bar{x}\bar{y}}, && \text{if } \delta F/\delta x = \delta F/\delta y
\end{aligned} \tag{4.6}$$

4.3.5 Abstraction-based bi-decompositions

A Boolean function F is bi-decomposable if it can be written as $F = G\ op\ H$, where *op* is a Boolean operation and G and H are non-constant functions. This work introduces two methods for bi-decomposition, which are based on the existential and the universal abstractions. As described in [16], these abstractions are related to F as follows:

$$\forall x_i F \leq F \leq \exists x_i F. \tag{4.7}$$

The existential abstraction $\exists x_i F$ is larger than F. Therefore, it implies an AND decomposition, e.g., $F = \exists x_i F \cdot H$. Similarly, the universal abstraction $\forall x_i F$ is smaller than F, and it implies an OR decomposition, e.g., $F = \forall x_i F + H$. Notice that this method can be applied to any number of variables, as long as the abstractions are not constants, i.e., $\exists x_i F \neq 1$, and $\forall x_i F \neq 0$. In this work, the abstraction-based bi-decompositions are applied to one variable (lines 16-23 in Fig. 4.6) and two variables (lines 25-32 in Fig. 4.7).

These abstractions have a characteristic of interest: their support is smaller than F in at least one variable, i.e., $|\exists x_i F| < |F|$ and $|\forall x_i F| < |F|$, given that x_i is in the support of F. Consequently, it is possible to guarantee the support reduction for at least one of the decomposing functions by using these abstractions.

The method proposed differs from other bi-decomposition methods [87, 60, 25], which try to identify variable partitions and the decomposing functions. The abstraction-based bi-decomposition is applied by setting one of the derived functions (G) to an abstraction ($\exists x_i F$ or $\forall x_i F$), and obtaining the other function (H) via don't care minimization.

The following conditions are used to obtain H via don't care minimization. For the AND bi-decomposition $F = G \cdot H$, $F \leq H \leq F + \overline{G}$, given F and G. Considering $G = \exists x_i F$, then $F \leq H \leq F + \overline{\exists x_i F}$. Similarly, the condition for the OR bi-decomposition $F = G + H$ is $F \cdot \overline{G} \leq H \leq F$, given F and G. Considering $G = \forall x_i F$, then $F \cdot \overline{\forall x_i F} \leq H \leq F$.

Example: Consider the function $F = abcde\overline{f} + \bar{a}\bar{b}\bar{c}\bar{d}\bar{e}f$. The universal abstraction w.r.t. any variable is the constant 0. Hence, it is not useful to perform the OR bi-decomposition $F = G + H$, since it degenerates to $G = 0$ and $H = F$. On the other hand, the AND bi-decomposition based on the existential abstraction generates a good support-reducing decomposition, as seen in Section 4.2. Consider the existential abstraction w.r.t. variable a: $G = \exists aF = bcde\overline{f} + \bar{b}\bar{c}\bar{d}\bar{e}f$. Using the conditions for H in an AND bi-decomposition ($F \leq H \leq F + \overline{G}$), the following ISF is defined:

$$abcde\overline{f} + \bar{a}\bar{b}\bar{c}\bar{d}\bar{e}f \leq H \leq \bar{a} + \bar{b} + \bar{c} + \bar{d} + \bar{e} + f$$

By applying Boolean minimization, $H = (ab + \bar{a}\bar{b})$ is obtained, and the following AND bi-decomposition is produced:

$$F = (bcde\overline{f} + \bar{b}\bar{c}\bar{d}\bar{e}f)(ab + \bar{a}\bar{b})$$

Notice that this is not a disjoint-support decomposition.

4.4 Recursive remapping

This section presents the proposed remapping approach. The idea is to collapse the whole LUT network recursively, decompose, and select the best mapping for each circuit part. An overview of the method is illustrated in Fig. 4.8, and a pseudo-code of the proposed approach is shown in Fig. 4.9. Different approaches were considered, such as computing maximum fanout-free cones and performing partial collapsing [114]. However, these methods were computationally more expensive and produced worse results than the approach proposed in this section. Notice that windowing and partial collapsing are biased by the structure and by the order that these processes are applied. On the other hand, the recursive remapping proposed is more aggressive, leading to potential (manageable) time-outs, but also larger gains.

The inputs for the remapping approach are a circuit description (N), the number of LUT inputs (k), and a cost function (COST), e.g., area, delay. The description can also be a valid FPGA mapping (for k-LUTs), indicated by the flag *isMap*. The output is an optimized FPGA mapping regarding COST.

The method can be divided into three sequential steps:

Step 1: Obtain LUT network M_1 by mapping (or remapping) the input description N. This is performed in function FPGAMAP (line 4), which runs structural hashing and AIG algebraic optimization. FPGA mapping is performed for each different structure generated, returning the best mapping for the cost function (COST).

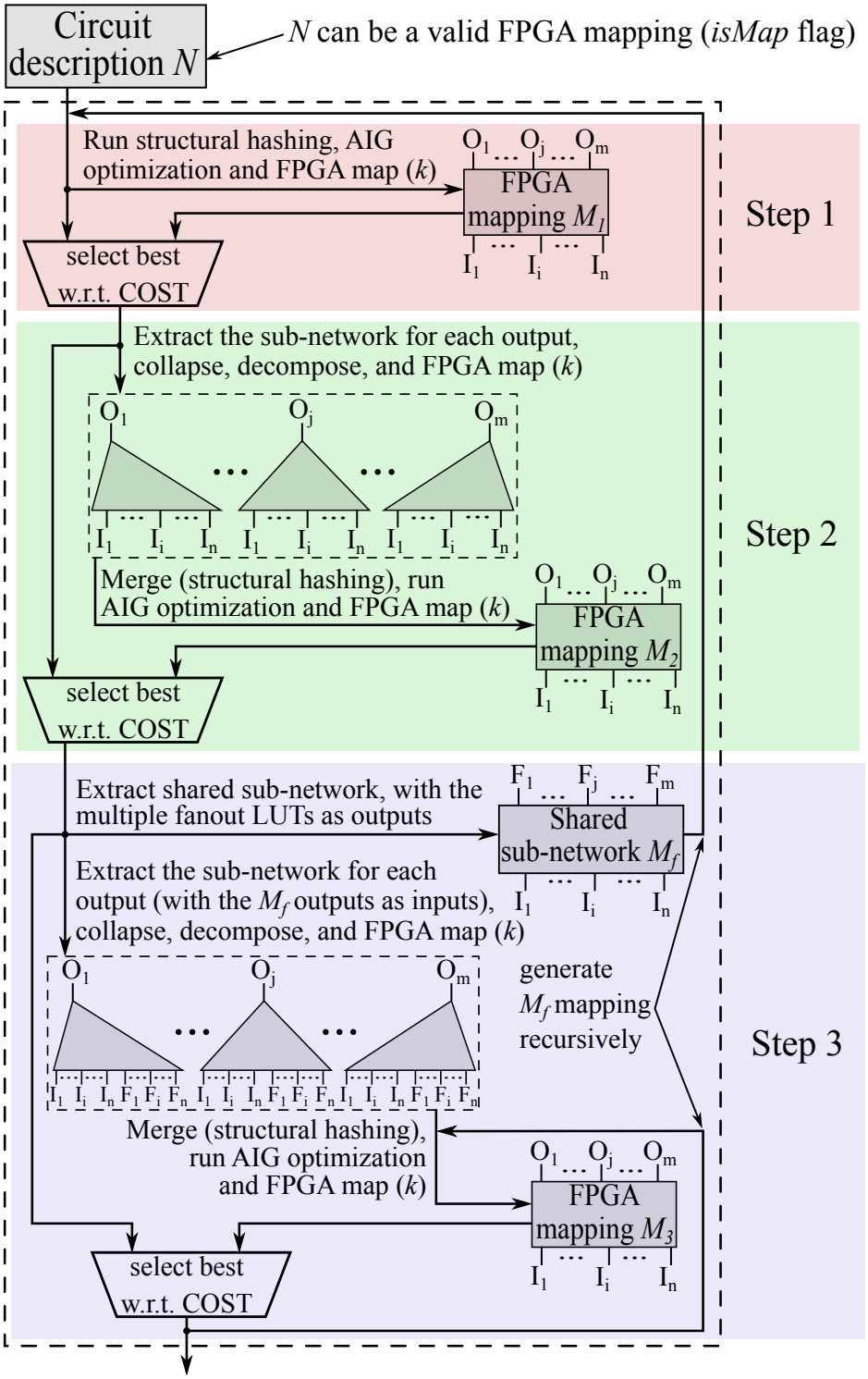


Figure 4.8: The recursive remapping approach.

```
1: function collapseDecomposeMap(N, k, cost, isMap)
   Input: Circuit description (N), LUT size (k), cost function (cost), flag (isMap)
   Output: An FPGA mapping guided by cost (BestMap)
2:     // Step 1 - obtain M1 by mapping (or remapping) input N
3:     // run structural hashing, AIG optimizations, FPGA mapping
4:     M1 = fpgaMap(N, k, cost)
5:     // define best mapping result regarding cost
6:     if isMap and cost(N) < cost(M1) then BestMap = N
7:     else BestMap = M1
8:     // if the number of levels is 1, return
9:     if levels(BestMap) == 1 then return BestMap
10:    // Step 2 - obtain M2 by remapping outputs individually
11:    outputNetworks = ∅                          // best mapping for each output
12:    // map each output individually
13:    for each output i in BestMap do
14:        // extract single output network
15:        Ntk_Oi = extractOutputToPIs(BestMap, i)
16:        // collapse, decompose, optimize, FPGA mapping
17:        Oi = collapseFpgaMap(Ntk_Oi, k, cost)
18:        outputNetworks.insert(Oi, i)
19:    // merge output networks using structural hashing
20:    Ntk_M2 = mergeNetworks(outputNetworks)
21:    // run AIG optimizations, FPGA mapping
22:    M2 = fpgaMap(Ntk_M2, k, cost)
23:    if cost(M2) < cost(BestMap) then BestMap = M2
24:    // Step 3 - obtain M3 by remapping outputs with a shared sub-network
25:    sharedNodes = ∅                                  // set of shared nodes used
26:    // create network with all multiple fanout nodes as primary inputs
27:    tempNtk = multipleFanoutToPI(BestMap)
28:    // map each output individually
29:    for each output i in tempNtk do
30:        // extract single output network
31:        Ntk_Oi = extractOutputToPIs(tempNtk, i)
32:        // collapse, decompose, optimize, FPGA mapping
33:        Oi = collapseFpgaMap(Ntk_Oi, k, cost)
34:        if cost(Oi) < cost(outputNetworks[i]) then
35:            outputNetworks.insert(Oi, i)
36:            sharedNodes.insert(inputs of Oi)
37:    if sharedNodes ≠ ∅ then
38:        // get shared sub-network with sharedNodes as outputs
39:        sharedNtk = getSharedSubNetwork(BestMap, sharedNodes)
40:        Mf = collapseDecomposeMap(sharedNtk, k, cost, true)
41:        // merge output networks and shared sub-network with struct. hashing
42:        Ntk_M3 = mergeNetworks(outputNetworks, Mf)
43:        // run AIG optimizations, FPGA mapping
44:        M3 = fpgaMap(Ntk_M3, k, cost)
45:        if cost(M3) < cost(BestMap) then BestMap = M3
46:    return BestMap
```

Figure 4.9: Pseudo-code of the recursive remapping approach.

Step 2: For each output, extract the single-output cone from the LUT network, optimize and map. The mapping is performed in the function collapseFpgaMap (line 17), which runs collapsing, decomposition, AIG optimization, and FPGA mapping. Collapsing is a computationally expensive process that may be unfeasible for complex networks, so a time-out is set to avoid a long runtime. If collapsing is successful (and a BDD is obtained, for example), then the decomposition presented in Section 4.3 is applied, generating a *new network*. If there is a time-out (or if the support size is too large), then the single output network extracted is the only structure considered. For each different network, structural hashing is performed, followed by a single execution of AIG optimization scripts and FPGA mapping. The best mapping for each output is greedily selected, and the FPGA mapping M_2 is generated by merging these mappings using structural hashing (line 20), followed by the function fpgaMap (line 22).
Step 3: Extract a *shared sub-network* from the best mapping found (N, M_1 or M_2). The outputs of this shared sub-network are the nodes with multiple fanout identified in topological order. These nodes are transformed to primary inputs from the outputs perspective (line 27), and the function collapseFpgaMap is applied for each output. The implementation for the *sub-network* is obtained recursively (line 40), until the number of levels reaches 1 (line 9). The FPGA mapping M_3 is generated by merging the output networks and the mapping of the *shared sub-network* using structural hashing (line 42), followed by the function fpgaMap (line 44). Finally, the method returns the best mapping between N, M_1, M_2 and M_3.

The recursive remapping approach creates different optimization opportunities. Regarding M_1, it is possible to derive better solutions by applying AIG optimization in a shared part of the network, instead of the whole circuit. Regarding M_2, if it is not possible to collapse the primary output function, it may be possible for a less complex function, removing part of the structural bias. Furthermore, an area recovery process is achieved, as common parts shared by more than one output are remapped.

4.5 Experimental results

The support-reducing functional decomposition and the recursive remapping are implemented in C++. BDDs are the representation of choice for the functional decomposition, and the CUDD BDD package [109] is used. CUDD provides an intuitive C++ API, and efficient methods to implement the decompositions proposed, with functions to calculate cofactors and abstractions, to identify essential literals, and to perform don't care minimization. Also,

BDDs are used in the collapsing of the LUT network, and provide an input for the decomposition without the need for a translation. The FPGA mapping based on priority cuts [86] and choices [21] implemented in ABC [14] is the one used in the recursive remapping approach. All results passed formal verification with the ABC command *cec*.

In order to obtain a *delay-oriented* FPGA mapping with ABC, the command '*if -C 12 -K k*' is used, which primarily targets delay, with a configuration of at most 12 priority cuts per node [82]. Alternatively, *area-oriented* FPGA mapping is obtained with the command '*if -a -C 12 -K k*'. The number of LUT inputs varies for the different sets of benchmarks. Regarding the BDD-based methods, k=5 is defined to compare with the published results. For the remaining cases, the LUT input size is k=6. Structural bias is further reduced by identifying *structural choices* [21] using the commands '*&synch2*' and '*&dch*' on top of the best FPGA mapping obtained.

The proposed approach attempts to optimize a given FPGA mapping. In the experiments presented in this section, the *input* FPGA mapping is either the best known mapping result for the EPFL benchmarks [6], or the FPGA mapping obtained by a commercial tool. For all other cases, the circuit description is used as input, and the mapping is produced by ABC.

The FPGA mappings reported are greedily selected based on the cost function. In this work, two cost functions are analyzed: logic levels and LUT count. If the objective is reducing delay, then SR-map greedily selects the circuit parts with less logic depth, using LUT count as a tie breaker. Alternatively, if the goal is to minimize area, LUT count is the main cost function and logic depth is the tie breaker.

4.5.1 BDD-based FPGA mapping tools

This section compares the results obtained with SR-map and the FPGA mappings of the tools BoolMap [61], BDS-pga [114], and ABC [14]. The FPGA mappings (with k=5) of BoolMap and BDS-pga refer to the best delay results reported in [61] and [114], which are presented in Table 4.2. BDS-pga obtains an area reduction of 9% in comparison with BoolMap, at the expense of increasing delay in 9%.

The ABC results are derived by reading the descriptions[1], applying structural hashing, 10 iterations of AIG optimization scripts (*compress*2*rs* and *dc*2), and delay-oriented FPGA mapping (with structural choices). The number of iterations can be tuned to achieve better results or a lower runtime. The ABC results are the same as the M_1 mappings presented in Section 4.4.

[1] `http://www.ecs.umass.edu/ece/tessier/rcg/bds-pga-2.0/`

Table 4.2: FPGA mapping comparison with BDD-based approaches ($k = 5$).

Circuit	BoolMap [61] (delay)		BDS-pga [114] (delay)		ABC (delay)		SR-map (delay) + ABC (delay)		
	LUTs	Lev.	LUTs	Lev.	LUTs	Lev.	LUTs	Lev.	Time(s)
5xp1	**13**	**2**	15	2	21	3	14	2	3
9sym	**7**	**3**	**7**	**3**	60	4	8	3	2
9symml	**7**	**3**	**7**	**3**	58	4	8	3	2
alu2	43	4	41	4	117	7	**35**	**4**	13
alu4	268	7	190	7	219	9	**102**	**4**	26
apex6	188	4	186	4	171	4	**169**	**3**	26
apex7	78	3	71	3	61	3	**54**	**3**	8
b9	41	3	40	3	33	3	**38**	**2**	3
C1355	98	5	**65**	**4**	66	4	66	4	157
C1908	137	7	119	7	95	6	**86**	**6**	217
C499	102	4	**64**	**4**	66	4	66	4	162
C5315	672	9	447	7	365	7	**374**	**6**	217
C880	134	8	108	8	87	7	**92**	**6**	55
clip	**15**	**2**	30	4	71	4	19	3	8
count	**42**	**2**	26	5	36	3	33	3	5
des	**594**	**3**	909	4	623	4	568	4	328
duke2	192	5	169	7	141	4	**120**	**4**	26
misex1	15	2	14	2	15	2	**11**	**2**	2
rd84	**10**	**2**	13	3	109	5	13	3	11
rot	228	6	218	9	203	6	**215**	**5**	43
t481	5	3	**5**	**2**	148	6	**5**	**2**	2
vg2	30	4	**12**	**3**	27	3	21	3	6
z4ml	**5**	**2**	**5**	**2**	**5**	**2**	5	2	1
Geomean	49.63	3.60	44.95	3.91	74.81	4.20	**40.59**	**3.31**	57.52
Ratio	1.00	1.00	0.91	1.09	1.51	1.17	0.82	0.92	-

ABC produces worse results than the BDD-based tools, with mappings 51% larger in area and 17% larger in delay, compared to BoolMap. The difference in LUTs is larger than 90% for benchmarks *rd84* and *t481*. The reason for this behavior lies on the nature of each approach. BoolMap and BDS-pga perform functional transformations using BDDs, which partially removes the structural bias, whereas ABC performs an structural mapping of an AIG, which nodes and levels were minimized. Notice that the ABC minimizations have a limited scope (e.g., cuts, windows), and for some benchmarks the best results are obtained after decomposing the global BDDs.

SR-map improves the ABC results and delivers a final result that outperforms BoolMap [61], even using the networks generated by ABC as starting

point. This work improves BoolMap results in 18% for area and in 8% for delay, with the best delay result for 12 of the 22 benchmarks. The bold values in Table 4.2 highlight the best delay results.

4.5.2 20 largest MCNC benchmarks

This section presents results for the 20 largest MCNC benchmarks, comparing the results of this work with the ones obtained with a commercial tool and ABC. The synthesis in the commercial tool is configured to avoid the use of multiplexers and merging of LUTs, delivering results comparable to the other tools. The reported runtime for the commercial tool regards only the logic synthesis and optimization steps.

Table 4.3 presents the FPGA mappings to LUTs with k=6. The bold numbers in '*Levels*' highlight the best delay results, whereas the bold numbers in '*LUTs*' underline the best results for area. Regarding the methods analyzed, SR-map obtains the best delay result for 19 of 20, and the best area result for 13 of the 20 benchmarks. All results are generated with the same input description. The ABC mapping is obtained with structural hashing, 10 iterations of AIG optimization scripts, and FPGA mapping identifying structural choices. The ABC results are the starting point for the proposed method, showing the difference of using the same mapping algorithm but exploring different structures.

BDS-pga [114] and MFS [82] results are omitted in Table 4.3 due to space, but they are shown in Table 4.4 and their relationship is (> means better results): commercial tool > MFS [82] > BDS-pga [114] > ABC (AIG optimization, and FPGA mapping with priority cuts and structural choices). Notice that BDS-pga did not finish for one of the benchmarks (*elliptic*), and it resulted in segmentation fault for other 4 benchmarks. Therefore, two geometric means and ratios are presented in Table 4.4: one for all benchmarks, and another to compare with BDS-pga results.

4.5.2.1 Delay-oriented mapping

Using the delay-oriented FPGA mapping, ABC produces a result 99% larger in area and 5% larger in delay when compared with the commercial tool. SR-map produces a result with 27% fewer logic levels and 10% fewer LUTs than the commercial tool, with ABC delay-oriented mapping, delay as cost function (COST), and the ABC mapping as start point. Additionally, an area reduction of 16% plus a delay reduction of 12% is achieved when area is defined as the cost function in SR-map.

Table 4.3: FPGA mapping comparison for the 20 largest MCNC benchmarks ($k = 6$).

								Recursive remapping												
Circuit	Commercial tool			ABC (delay)		ABC (area)		Factor (delay) + ABC (delay)		Factor (area) + ABC (delay)		SR-map (delay) + ABC (delay)			SR-map (area) + ABC (delay)			SR-map (area) + ABC (area)	
	LUTs	Lev.	Time (s)	LUTs	Lev.	LUTs	Lev.	LUTs	Lev.	LUTs	Lev.	LUTs	Lev.	Time (s)	LUTs	Lev.	Time (s)	LUTs	Lev.
alu4	320	5	230	456	5	415	10	183	5	186	5	63	**3**	24	58	4	21	**56**	5
apex2	302	13	276	507	6	408	11	40	4	40	4	35	**3**	17	34	4	14	**30**	7
apex4	192	3	290	558	5	529	10	390	4	390	4	155	3	203	155	3	212	**153**	**3**
bigkey	569	3	313	577	3	577	3	577	3	577	3	685	**2**	257	**491**	3	191	577	3
clma	**180**	5	438	2614	8	2221	18	200	4	203	4	203	4	75	190	**4**	84	186	8
des	**436**	4	345	447	4	457	7	447	4	446	4	513	**3**	194	445	4	222	449	5
diffeq	**472**	8	348	559	7	510	13	559	7	532	8	533	**7**	241	527	8	262	502	14
dsip	**690**	3	338	871	3	871	3	869	**2**	869	**2**	869	**2**	190	869	**2**	224	869	**2**
elliptic	**115**	**5**	313	315	6	297	11	315	6	315	6	316	5	36	311	6	51	291	11
ex1010	210	3	335	572	5	550	10	453	4	432	5	208	**3**	107	**207**	4	180	**207**	4
ex5p	100	2	252	326	4	301	9	91	2	85	3	86	2	12	**82**	**2**	16	82	5
frisc	1694	13	292	1725	12	1698	26	1886	10	1692	13	1857	**10**	654	1689	13	853	**1637**	25
i10	557	9	285	535	8	500	22	627	7	522	9	537	**7**	329	505	9	317	**481**	24
misex3	197	5	240	284	5	234	9	205	4	193	4	117	4	37	102	**4**	35	**94**	5
pdc	155	4	286	1385	6	1143	14	154	4	148	4	157	**3**	70	**142**	4	67	144	6
s38417	**1458**	7	437	2557	6	2443	11	2460	6	2458	7	2450	**6**	1304	2396	7	1035	2369	12
s38584	**1946**	8	432	2287	6	2255	12	2463	5	2212	6	2224	**5**	1246	2229	5	729	2198	12
seq	531	7	247	583	5	520	10	560	4	486	5	527	**4**	307	459	5	114	**420**	10
spla	157	4	269	1350	6	1128	15	145	4	137	4	156	**3**	138	135	4	62	**121**	6
tseng	656	8	269	651	6	631	13	647	6	635	8	634	**6**	385	636	8	265	**629**	12
Geomean	374.30	5.24	306.1	744.07	5.51	685.86	10.56	400.08	4.43	383.36	4.92	335.79	**3.84**	141.1	312.67	4.61	128.9	**304.62**	7.21
Ratio	1.00	1.00	-	1.99	1.05	1.83	2.01	1.07	0.84	1.02	0.94	0.90	**0.73**	-	0.84	0.88	-	**0.81**	1.38

Table 4.4: FPGA mapping of 20 largest MCNC benchmarks ($k = 6$) for MFS [82], BDS-pga [114], ABC and SR-map.

Circuit	Commercial tool		BDS-pga [114]		MFS [82]		ABC (delay)		SR-map (delay) + ABC (delay)		SR-map (area) + ABC (delay)	
	LUTs	Levels	LUTs	Levels	LUTs	Levels	LUTs	Levels	LUTs	Levels	LUTs	Levels
alu4	320	5	568	7	433	5	470	5	62	3	62	4
apex2	302	13	759	8	640	6	512	6	39	4	37	4
apex4	192	3	-	-	776	5	569	5	155	3	155	3
bigkey	569	3	872	3	485	3	577	3	685	2	577	3
clma	180	5	-	-	609	7	2641	9	205	4	192	6
des	436	4	596	3	558	4	751	5	576	3	527	4
diffeq	472	8	648	8	645	7	565	7	543	7	534	8
dsip	690	3	885	3	680	3	871	3	869	2	869	2
elliptic	115	5	-	-	1754	10	316	6	313	6	313	6
ex1010	210	3	428	5	1250	6	572	5	208	3	207	4
ex5p	100	2	458	5	105	3	362	4	86	2	86	2
frisc	1694	13	2227	20	1717	10	1725	12	1797	10	1689	13
i10	557	9	812	10	530	8	578	8	525	8	518	10
misex3	197	5	540	6	507	4	345	5	124	4	108	4
pdc	155	4	212	6	166	4	1385	6	162	4	152	5
s38417	1458	7	2739	6	2202	6	2489	7	2618	6	2435	6
s38584	1946	8	-	-	2044	5	2287	6	2244	5	2224	5
seq	531	7	807	6	548	5	591	5	535	5	491	5
spla	157	4	181	5	135	4	1350	6	158	3	121	4
tseng	656	8	-	-	649	6	654	7	632	7	636	8
Geomean (all)	374.30	5.24	662.66	5.93	614.65	5.21	781.06	5.70	341.90	4.09	323.52	4.74
Ratio (all)	1.00	1.00	1.77	1.13	1.64	0.99	2.09	1.09	0.91	0.78	0.86	0.90
Geomean (BDS)	383.65	5.18	662.66	5.93	519.30	4.88	735.90	5.46	317.62	3.89	296.40	4.56
Ratio (BDS)	1.00	1.00	1.73	1.14	1.35	0.94	1.92	1.05	0.83	0.75	0.77	0.88

4.5.2.2 Area-oriented mapping

ABC produces a result 83% larger in area than the commercial tool using area-oriented FPGA mapping. Notice that there is an area recovery post-process in ABC delay-oriented mapping, but area-oriented mapping does not try to improve delay. Therefore, delay is increased significantly, almost doubling the logic levels. Using ABC area-oriented mapping and area as cost function, SR-map obtains a result with 19% fewer LUTs than the commercial tool, but with 38% more logic levels. The results are 3% smaller in area than using ABC delay-oriented mapping, but with much worse delay results, as this is disregarded in ABC area-oriented mapping. For this reason, this configuration is not recommended if delay must be considered.

4.5.2.3 Support-reducing decomposition

The remapping approach proposed in Section 4.4 can be applied regardless of the support-reducing decomposition presented in Section 4.3. For example, the collapsed functions can be decomposed using algebraic factorization [15], instead of the method proposed. The results for the recursive remapping using factorization (derived by ABC command *strash*) instead of the support-reducing decomposition are also presented in Table 4.3, denoted as '*Factor*'. For some benchmarks, e.g., *apex2*, *clma*, *ex5p*, the removal of the structural bias using recursive remapping produces similar results both for factorization and decomposition. However, considering the full set of benchmarks, the results obtained using the support-reducing decomposition are considerably better than the ones using factorization, both for area and delay.

4.5.3 EPFL benchmarks

The set of EPFL benchmarks [6] consists of 20 designs, 10 arithmetic and 10 random/control circuits. Since 2015, the best known FPGA mapping results (with k=6) for delay and for area are recorded[2]. Consequently, these benchmarks have highly optimized results, which are very difficult to improve. For example, the commercial tool used in this work is not able to improve any of the EPFL results, as it provides FPGA mappings with more balanced results in area and delay, and also considers congestion issues.

The proposed method is able to update 12 of the best known results for delay, and 6 for area, as presented in Table 4.5. The most remarkable results are: *cavlc*, with a reduction of 25% in delay plus 36% in area; *int2float*, reducing LUT count in 34%; and the *arbiter*, with an area reduction of 27%.

[2] `https://github.com/lsils/benchmarks/tree/master/best_results`

Table 4.5: Best known results for EPFL benchmarks.

Circuit	Best EPFL (delay)		SR-map (delay) + ABC (delay)		
	LUTs	Levels	LUTs	Levels	Time(s)
adder	470	5	459	5	199
arbiter	2884	5	2099	5	729
cavlc	115	4	73	3	25
dec	270	2	264	2	11
i2c	244	3	240	3	30
int2float	41	3	27	3	5
mem_ctrl	2490	7	2458	6	792
max	882	10	857	10	313
multiplier	7274	27	6514	27	24034
priority	157	4	152	4	15
router	57	4	51	4	5
voter	1469	12	1435	12	4601
Circuit	Best EPFL (area)		SR-map (area) + ABC (area)		
	LUTs	Levels	LUTs	Levels	Time(s)
cavlc	101	6	68	4	25
dec	270	2	264	2	11
i2c	225	7	224	6	29
int2float	28	6	26	4	5
priority	108	26	106	26	25
router	52	6	51	4	5

4.5.4 Remapping of the results from a commercial tool

The results in Table 4.3 are obtained from the original BLIF descriptions. In this section, the results obtained with the commercial tool are remapped by SR-map. The remapping results are presented in Table 4.6. The commercial tool performs sequential optimization, and equivalence checking with the original description is performed with ABC command *dsec*.

A reduction of 4% in area and 18% in delay of the results from a commercial tool is obtained with ABC by performing iterative AIG transformations and FPGA mapping with choices. Using the ABC delay-oriented mapping and delay as cost function, SR-map achieves even better results, with 28% fewer logic levels and 18% fewer LUTs. Also, an area reduction of 28% plus a logic depth reduction of 14% is obtained when area is defined as the cost function for SR-map.

Table 4.6: Remapping of the commercial tool results for the 20 largest MCNC benchmarks ($k = 6$).

Circuit	ABC (delay)		SR-map (delay) + ABC (delay)			SR-map (area) + ABC (delay)		
	LUTs	Levels	LUTs	Levels	Time(s)	LUTs	Levels	Time(s)
alu4	232	5	65	3	28	58	4	19
apex2	179	6	33	4	13	32	4	9
apex4	193	3	155	3	107	155	3	86
bigkey	573	3	681	2	285	459	3	144
clma	193	3	181	3	24	166	4	17
des	446	4	445	3	232	436	4	178
diffeq	470	6	470	6	261	452	7	223
dsip	869	2	869	2	215	681	2	181
elliptic	105	4	110	3	21	99	4	16
ex1010	210	3	208	3	96	207	4	76
ex5p	92	2	86	2	10	82	2	7
frisc	1839	10	1892	9	947	1684	11	792
i10	551	8	533	8	302	512	9	280
misex3	158	4	112	4	28	96	4	21
pdc	126	4	148	3	23	114	4	15
s38417	2493	6	2432	6	967	1458	7	971
s38584	2188	6	2332	5	640	1946	8	641
seq	458	5	417	5	89	446	5	77
spla	131	4	145	3	20	119	4	15
tseng	649	6	636	6	240	625	7	237
Geomean	360.88	4.32	306.47	3.76	94.8	270.47	4.52	74.3
Ratio	0.96	0.82	0.82	0.72	-	0.72	0.86	-

Table 4.7: Results of a commercial tool for different strategies, after physical synthesis (post place-and-route).

Input from:	Synthesis strategy of the tool					
	Default		*Area*		*Delay*	
	Area	Delay	Area	Delay	Area	Delay
Initial description	330.73	4.94	320.32	5.65	375.24	4.92
SR-map (delay) + ABC(delay)	-20%	-9%	-16%	-9%	-20%	-6%
SR-map (area) + ABC(delay)	-23%	-6%	-22%	-11%	-23%	-8%

4.5.5 SR-map result as input to the commercial tool

Previous experiments are presented with results in number of LUTs and levels, but reduction in logic levels often does not translate into improved delay post place-and-route due to congestion issues. In order to evaluate this effect, SR-map results are fed back to the commercial tool, comparing the post place-and-route metrics between using the initial description versus using the SR-map remapping as input.

A summary of the geometric mean results is presented in Table 4.7. Detailed results are shown in Table 4.8, Table 4.9, and Table 4.10, with the "*Ratio*" showing a comparison with the results obtained by the Default strategy using the initial description. Three different synthesis strategies are investigated: (1) with the *Default* parameters, (2) targeting *Area* minimization, and (3) *Delay* reduction. The delay reported is for the critical path, in ns.

In previous experiments, the synthesis of the commercial tool was configured to avoid the use of multiplexers and merging of LUTs, delivering results comparable to the other tools. In this section, the results presented may have multiplexers, and the LUTs may be merged. Additionally, sequential optimization is performed, and the number of registers may vary for the different synthesis strategies and inputs. The area reported considers these characteristics of the commercial tool: logic optimization and merging (LUTs), sequential optimization (FFs), and the use of multiplexers to reduce delay and implement functions with more inputs (7 and 8). Multiplexers occupy much less area than LUTs and flip-flops, so we conservatively consider muxes as half the size of the other elements, resulting in this function to compare area: $Area = LUTs + FFs + 0.5 \times Multiplexers$.

In comparison with the *Default strategy*, and using the initial description, the *Area strategy* improves area in 3% for an increase of 14% in delay, whereas the *Delay strategy* reduces delay by 1% at the expense of an area 14% larger.

As observed in Table 4.8, Table 4.9, and Table 4.10, the post place-and-route results show worse metrics than the ones reported in Table 4.6, as these

are implemented into an actual FPGA. Still, for the *Default strategy*, the results have 20% less area and 9% less delay, by using the SR-map remapping targeting delay as input, and 23% less area and 6% less delay, by using the SR-map remapping targeting area as input. Similar results are achieved for the other strategies. Notice that the commercial tool results in Table 4.3 are for a different synthesis strategy, in which the use of multiplexers and LUT combining is avoided.

Interestingly, the SR-map remapping targeting area as input to the commercial tool generated better post place-and-route results, even for delay. This is because aggressive reduction of logic levels often leads to congestion, which prevents the place-and-route tools to translate this reduction into better performance. A detailed comparison of the benchmark results for this case is provided in the plots of Fig. 4.10.

4.5.6 Scalability analysis

Boolean methods are known to have scalability issues, and this is typically handled by limiting the scope of application, with techniques such as partial collapsing [23, 61, 114], windowing [82], and partitioning [40]. The proposed approaches are no different. The size of BDDs may increase significantly with the number of variables, slowing down even simple BDD operations. For this reason, the collapsing process is only applied to functions with up to a certain limit of input variables. Fig. 4.11 shows the average area-delay product for different limits in the support, presenting a trade-off between runtime and quality of results. The trade-off presented is obtained with SR-map for a subset of the benchmarks considered in previous sections, which have functions of 70 variables or more. The support limit is defined to 50 variables for the results previously presented in this work.

Additionally, a time-out is set to prevent a long runtime. This runtime limit is set for the collapsing process and the decomposition. Notice that the BDD size may be large and the BDD operations may take a long time, even with a limit in the number of variables. As observed in Fig. 4.11, the quality of results obtained by SR-map is similar for a time-out of 5 and 1000 seconds. The runtime with time-out of 1000 seconds exposes the exponential behavior of increasing the number of variables for BDD operations. However, the runtime can be kept under control by defining a smaller time-out, which also presents a linear increase with the support limit. As mentioned in Section 4.4, if there is a time-out (or if the support size is larger than the limit defined), then the output network extracted is the only one considered in the remapping algorithm.

Also, note that the remapping method extracts a sub-network that feeds

Table 4.8: Results of a commercial tool after physical synthesis using the initial description as input.

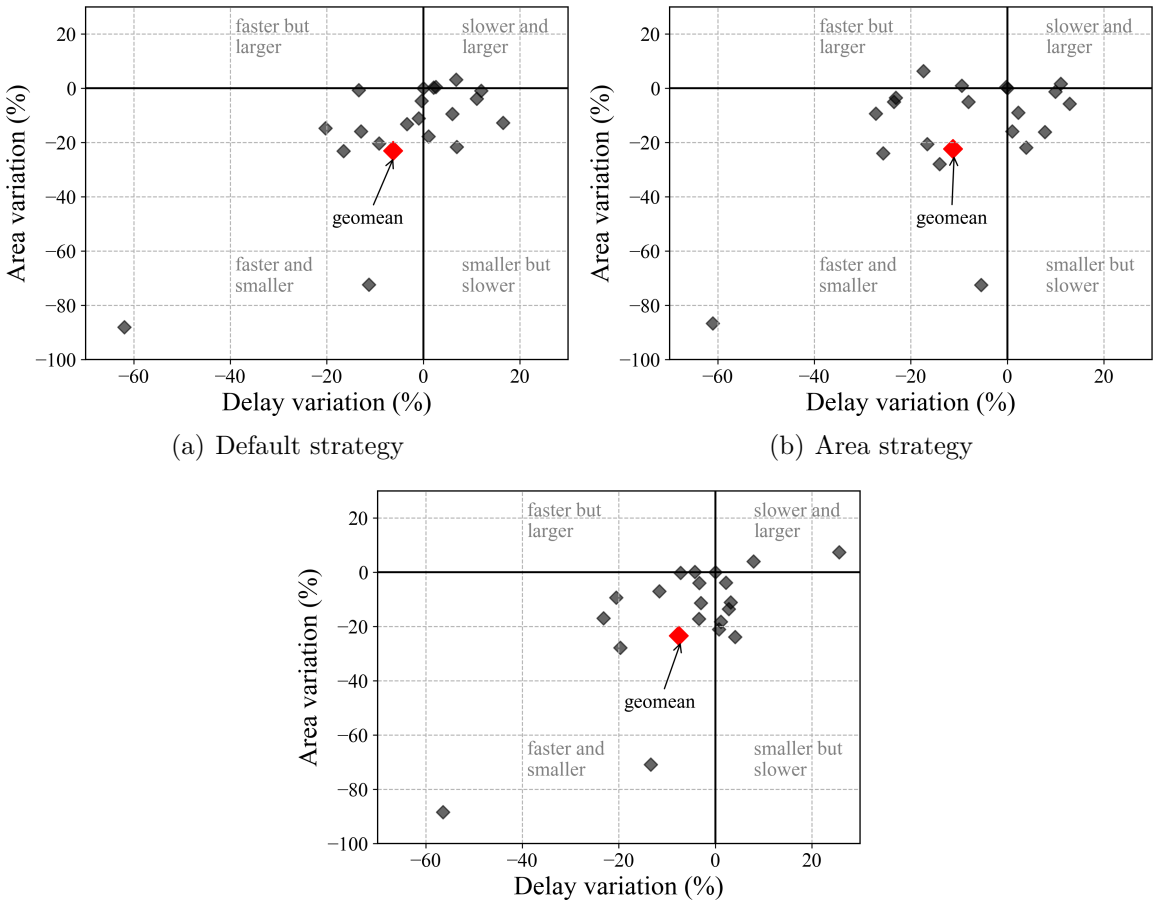
Circuit	Default strategy				Area strategy				Delay strategy			
	LUTs	Area	Delay(ns)	Time(s)	LUTs	Area	Delay(ns)	Time(s)	LUTs	Area	Delay(ns)	Time(s)
alu4	261	275.5	4.055	230	227	244.5	4.775	261	305	319.5	4.084	185
apex2	263	268	8.747	276	224	226.5	9.264	249	300	305	7.87	246
apex4	156	186	2.399	290	161	192.5	2.908	258	157	187	2.398	250
bigkey	573	573	5.511	313	459	459	5.3	266	569	569	4.494	255
clma	147	147	3.803	438	150	150	4.057	571	180	180	3.751	362
des	425	425	4.748	345	457	457	4.853	264	436	436	5.197	262
diffeq	433	433	6.846	348	427	427	9.142	262	472	472	7.111	271
dsip	577	577	4.28	338	580	580	3.865	256	690	690	4.827	249
elliptic	85	85	3.516	313	80	80	4.043	245	115	115	3.57	251
ex1010	170	230	2.412	335	170	230	2.412	263	170	230	2.412	256
ex5p	87	87	2.485	252	86	86	3.715	257	100	100	2.528	244
frisc	1564	1564	12.901	292	1532	1532	14.718	316	1703	1703	12.753	284
i10	497	497.5	8.597	285	499	499.5	12.409	338	549	549.5	8.347	296
misex3	159	173	3.761	240	130	141	3.68	275	178	192	3.614	252
pdc	128	128	4.588	286	124	124	4.578	381	155	155	3.516	287
s38417	1227	1228	7.863	437	1217	1217.5	9.772	973	1469	1470	9.261	399
s38584	1628	1629.5	6.471	432	1624	1625.5	6.91	502	1938	1939.5	6.795	349
seq	460	485.5	6.013	247	513	532	8.639	320	509	534.5	7.189	185
spla	132	132	4.183	269	132	132	4.562	334	157	157	3.716	204
tseng	531	531	5.966	269	529	529	7.802	278	656	656	5.952	197
Geomean	319.48	330.73	4.94	306.10	309.51	320.32	5.65	319.29	362.93	375.24	4.92	258.91
Ratio	1.00	1.00	1.00		0.97	0.97	1.14		1.14	1.13	0.99	

Table 4.9: Results of a commercial tool after physical synthesis using the SR-map (area) + ABC (delay) remapping as input.

Circuit	Default strategy				Area strategy				Delay strategy			
	LUTs	Area	Delay(ns)	Time(s)	LUTs	Area	Delay(ns)	Time(s)	LUTs	Area	Delay(ns)	Time(s)
alu4	76	76	3.597	259	67	67	4.516	256	93	93	3.536	254
apex2	32	32	3.327	242	30	30	3.611	237	35	35	3.426	244
apex4	125	153	2.425	258	125	153	2.425	257	125	153	2.425	264
bigkey	489	489	4.393	255	416	416	3.857	276	611	611	5.647	255
clma	113	113	3.174	243	108	108	3.488	247	130	130	3.013	249
des	426	426	4.85	269	416	416	4.963	286	435	435	4.821	264
diffeq	447	447	7.311	267	454	454	7.55	305	491	491	7.672	283
dsip	461	573	3.708	262	461	573	4.249	275	461	573	3.708	276
elliptic	77	77	3.728	244	76	76	3.093	243	102	102	3.463	251
ex1010	170	230	2.412	267	170	230	2.412	259	170	230	2.412	264
ex5p	83	83	2.476	246	83	83	2.855	248	93	93	2.235	248
frisc	1225	1225	13.792	298	1197	1197	15.291	329	1344	1344	12.847	297
i10	477	478.5	9.547	291	473	474.5	11.408	276	526	527.5	8.067	292
misex3	140	151	4.383	251	123	133	4.154	261	155	166	3.715	270
pdc	102	102	4.167	245	104	104	4.933	251	118	118	3.66	249
s38417	1217	1217.5	8.803	515	1238	1238.5	10.853	1713	1414	1414.5	9.466	520
s38584	1638	1638.5	6.638	417	1633	1633.5	6.892	532	1941	1941.5	6.508	391
seq	421	421.5	5.807	269	404	404.5	6.413	209	484	484.5	5.71	268
spla	111	111	3.641	247	111	111	4.607	246	130	130	3.591	233
tseng	472	472	5.908	300	534	534	7.067	266	583	583	6.143	264
Geomean	244.60	254.62	4.63	276.65	238.99	248.81	5.01	297.51	276.21	287.41	4.54	276.62
Ratio	0.77	0.77	0.94		0.75	0.75	1.01		0.86	0.87	0.92	

Table 4.10: Results of a commercial tool after physical synthesis using the SR-map (delay) + ABC (delay) remapping as input.

Circuit	Default strategy				Area strategy				Delay strategy			
	LUTs	Area	Delay(ns)	Time(s)	LUTs	Area	Delay(ns)	Time(s)	LUTs	Area	Delay(ns)	Time(s)
alu4	84	84	4.157	248	77	77	5.051	262	102	102	3.791	251
apex2	31	31	3.565	250	31	31	4.099	277	34	34	3.508	252
apex4	125	153	2.425	247	125	153	2.425	291	125	153	2.425	260
bigkey	457	569	5.825	299	457	568	3.723	289	457	569	5.825	291
clma	117	117	3.352	286	116	116	3.311	263	139	139	3.532	285
des	460	460	4.171	283	478	478	5.067	281	497	497	5.231	290
diffeq	435	435	6.329	294	392	392	9.063	279	477	477	6.844	284
dsip	469	469	3.585	387	469	469	3.585	387	469	469	3.585	387
elliptic	83	83	3.469	262	80	80	3.506	339	111	111	3.781	244
ex1010	170	230	2.412	294	316	345.5	4.122	353	170	230	2.412	289
ex5p	85	85	2.671	319	88	88	2.994	249	99	99	2.314	324
frisc	1283	1283	11.666	413	1263	1263	14.732	347	1461	1461	13.066	419
i10	525	525.5	8.051	356	518	518.5	10.716	329	577	577.5	7.458	368
misex3	129	138.5	3.994	265	157	167	4.295	262	141	150.5	3.528	267
pdc	145	145	4.168	254	138	138	4.587	262	173	173	3.992	257
s38417	1251	1253.5	8.029	348	1221	1223	8.968	1298	1433	1435.5	8.335	353
s38584	1645	1650.5	5.364	339	1592	1597	5.225	300	1946	1951.5	6.104	341
seq	419	419	5.216	262	409	409	6.384	209	494	494	6.484	277
spla	131	131	4.097	269	135	135	4.526	201	156	156	4.275	256
tseng	490	490	5.526	278	469	469	8.506	226	610	610	5.748	278
Geomean	255.14	265.56	4.50	294.26	261.33	268.97	5.13	302.82	288.01	299.67	4.62	295.01
Ratio	0.80	0.80	0.91		0.82	0.81	1.04		0.90	0.91	0.94	



(a) Default strategy

(b) Area strategy

(c) Delay strategy

Figure 4.10: Comparison of commercial tool results with the SR-map result as input vs. the initial description, for different synthesis strategies.

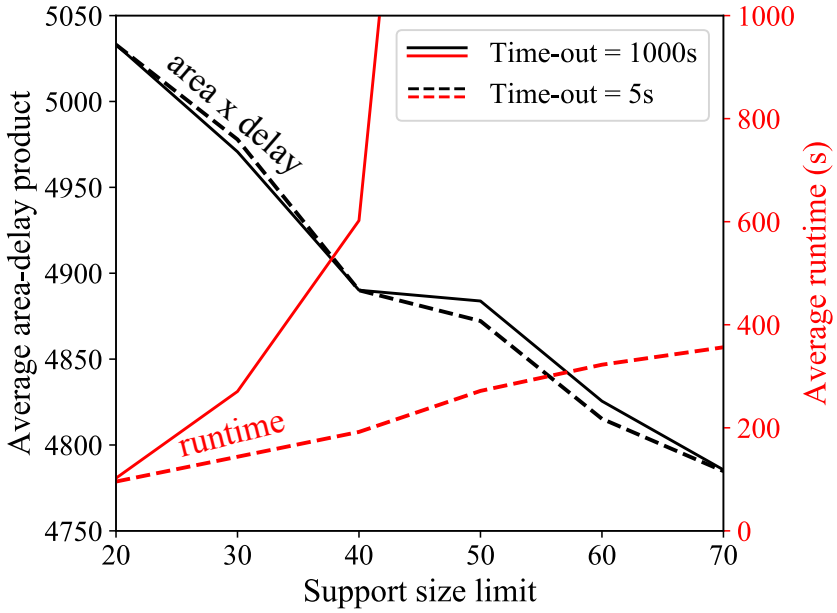


Figure 4.11: Runtime and quality analysis, considering different limits for the support size and time-outs. The area-delay product is the result of number of LUTs times the number of levels.

all outputs, therefore a larger number of levels in the FPGA mapping may also result in a larger runtime. The ABC tool presents a fraction of the runtime obtained with SR-map, as it is part of the proposed method, and it is repeatedly used. Nevertheless, the average execution time observed in Table 4.3 for SR-map (141 and 129 seconds) is comparable with the one for the commercial tool (306 seconds).

Chapter 5

Robustness to Voltage Noise with Ring Oscillator Clocks

Voltage noise is the main source of dynamic variability in integrated circuits and a major concern for the design of *power delivery networks* (PDNs). Lower supply voltages were made possible with technology scaling, but power density was also increased. Consequently, power integrity became a key factor in the design of reliable high-performance circuits.

ring oscillator clocks (ROCs) have been proposed as an alternative to mitigate the negative effects of voltage noise [71]. The capability of reacting instantaneously to large voltage droops makes ROCs an attractive solution, which also allows to relax the constraints required for the PDN design. However, the effectiveness highly depends on the design parameters of the PDN, power consumption patterns, and the spatial locality of the ROC within the clock domain [72]. This chapter analyzes the impact of the PDN parameters and ROC location on the robustness to voltage noise.

The experiments show that up to 83% of the margins for voltage noise can be reduced by using ROCs instead of rigid clocks. Also, up to 27% of the total leakage power can be reduced by using ROCs instead of decoupling capacitors. Additional PDN simplifications are possible, with fewer power interconnections or package decaps of lower quality. Tolerance to voltage noise and related benefits can also be increased with multiple ROCs.

5.1 Motivation

The estimation of the path delays and their variability is critical for the reliability of digital circuits. In order to define a robust clock period, it is necessary to consider all conditions that may shift and affect the delay of

every circuit path, such as the manufacturing process, the supply voltage, and the temperature (PVT conditions). Static offsets of these conditions are estimated at design time and taken into account by adding guard band margins to the nominal clock period. Nevertheless, dynamic shifts are hard to predict and excessively conservative margins are often added to prevent failures.

Augmenting the clock period offers more robustness against these changes in the operating conditions, but this comes at the expense of reducing performance. Another solution is to increase the amount of decoupling capacitors (decaps) [96, 99]. Voltage noise is mitigated when the system has a larger on-chip and off-chip capacitance. Unfortunately, the additional decaps imply an increase in area and leakage power, and variations that exceed the defined margins cannot be fully eliminated.

Several works in the literature suggest software-based solutions that do not have negative impacts on power and performance. A voltage noise prediction based on software signatures was proposed in [101], throttling the processor during specific cases. Thread balancing also contributes to reduce di/dt and smooth voltage swings [102]. Code sequences that may generate large voltage variations were identified in [42], and software guidelines were proposed to avoid the loops that may potentially produce such large voltage swings. The work in [43] proposes mechanisms to monitor the microarchitectural events that may generate large voltage oscillations. All these *software* solutions [42, 101, 102] are effective, but only applicable to microprocessors. They cannot be used to address the problem in a general context, for any kind of circuit or application.

In [120], the use of integrated voltage regulators is investigated, quantifying the penalties in area and power for the voltage noise reductions obtained. Other proposals include: improving the PDN impedance, which requires adaptations for each particular circuit; static and dynamic voltage margining, which result in higher power consumption; and performance throttling and stalling [49, 12], which require high-quality voltage sensors, with additional area and power. All these solutions have important overheads in design cost, area, power or performance. Adaptive clocking [113, 58, 93, 50, 117] seems to be a promising solution with low overhead, but with its efficiency limited by the characteristics of its voltage sensors and clock generators.

ROCs [28, 29] are an alternative proposal for adaptive clocking, which takes into account all sources of variability, voltage noise included. If the ROC is correctly designed, then a strong correlation can be achieved between the clock period and the delay of the critical paths. Considering that the ROC and the critical paths are exposed to the same sources of variability, the clock generator adapts immediately to the circuit demands.

Unfortunately, voltage fluctuations are not uniform across the die. Two distant points in the same die may have different voltage levels. This unsteady behavior raises some questions:

- How the global and local portions of voltage noise affect the performance when using ROCs?
- Is it possible to relax the PDN design by using ROCs?
- What is the relation between the required timing margins for an ROC and the size of its clock domain?
- Where to locate the ROC within a clock domain?

Voltage noise analysis has been focused on estimating the global worst-case and deriving the timing margins required [97]. For example, if the nominal voltage is 1V and the minimum voltage estimated is 0.85V, then a circuit with a rigid clock must consider a variation of 150mV for the clock period. For ROCs, the key value is the largest *differential voltage* between the ROC and the critical path [72]. If the voltage at the ROC is 0.9V when it is 0.85V at the critical path, then the clock period margins must cover only the difference: 50mV.

5.2 Models and metrics

The goal of this work is to present a conservative analysis of the benefits of using ROCs when dealing with problems related to voltage noise. Robustness to voltage noise is achieved without degrading performance, making possible the simplification of the PDN design. In order to provide a flexible and easily reproducible method, but also illustrative enough to cover a broad range of potential applications, the models presented in this section are used.

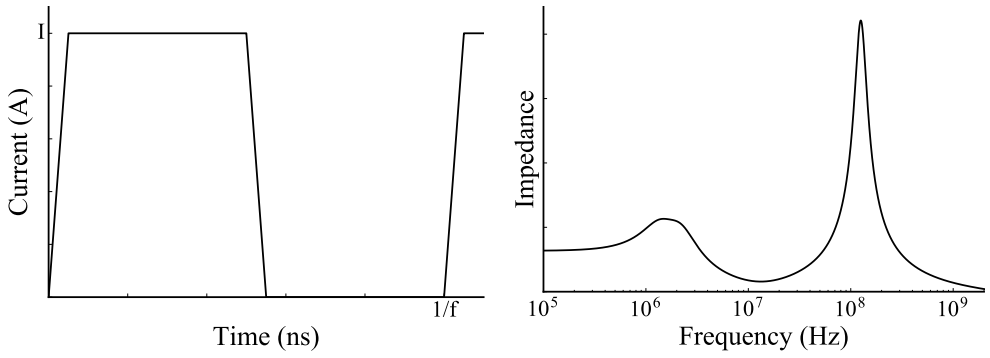
5.2.1 PDN model

The chip-grid presented in [41] represents a *system-on-chip* (SoC) with four cores of Pentium 4 and it is used as the PDN model in this work. The components illustrated in Fig. 2.11 and their values presented in Table 5.1 are obtained from [41] and described using SPICE netlists. Notice that the values in Table 5.1 were derived via extensive measurements, being an excellent representation of the design selected. However, the analysis presented in this chapter would have similar results for different designs and models, as all PDNs presents some level of voltage noise and voltage droops.

As external regulators typically do not regulate high frequency variations, the *voltage regulator module* (VRM) is modeled as a fixed voltage source delivering 1V at the power bumps. The on-chip power distribution is modeled with a 12×12 grid [97], as seen in Fig. 2.11(b). Both the power and ground networks are considered in the model, with a V_{DD} or a V_{SS} bump connected at every grid point.

Table 5.1: PDN parameters

Param.	Value	Param.	Value	Param.	Value
R_{pcb}	0.094 mΩ	L_{pcb}	21 pH	V_{vrm}	1 V
R_{cpcb}	0.17 mΩ	L_{cpcb}	1 pH	C_{pcb}	240 μF
R_{pkg}	1 mΩ	L_{pkg}	120 pH	C_{pkg}	26 μF
R_{cpkg}	0.54 mΩ	L_{cpkg}	5.61 pH	C_{ckt}	120 pF
R_{bump}	40 mΩ	L_{bump}	72 pH	I_{ckt}	195 mA
R_{grid}	50 mΩ	L_{grid}	5.6 fH	-	-



(a) Current source waveform (b) Impedance response

Figure 5.1: Current source waveform and impedance response for a PDN with a total of 200nF of on-chip decaps.

Each point in the grid models a portion of the circuit, with an *intrinsic decoupling capacitance* and a current source emulating the circuit power consumption considering the clock period, with rise, high and fall times set to 5%, 45%, and 5%, respectively (see Fig. 5.1(a)). Additionally, a decoupling capacitor is added at each point. Note that spreading the decaps uniformly is the best placement in order to reduce voltage fluctuations, considering a similar power consumption throughout the die [99, 97]. The frequency response of Fig. 5.1(b) is observed at any point of chip grid, considering a total of 200nF of on-chip decoupling capacitance. For comparison, the frequency

response changes significantly across the grid points by reducing the number of power bumps between the package and the chip (see Fig. 5.15).

5.2.2 Delay model

A simplification of the gate delay formulation was proposed in [103], which is still widely accepted. This model defines the delay (td) for a given voltage based on the threshold voltage (V_{th}) and a technology fitting value α in the range of 1-2. Notice that the model was defined for a single gate, but the relationship between delay and voltage holds for a path composed of several gates. Considering that V_{th}, α and k have small variation with the voltage, then it is possible to calculate the constant k in (5.1) and have the path delay based on the supply voltage.

$$td(V_{\text{DD}}) = \frac{k \cdot V_{\text{DD}}}{(V_{\text{DD}} - V_{\text{th}})^{\alpha}} \tag{5.1}$$

A 65nm commercial library with nominal voltage of 1V is used as reference. The average V_{th} of all combinational cells of the library is 0.36V for 75^oC, and 0.4V for 125^oC. A typical value of α is 1.3 [103], and this parameter is closer to 1 for more advanced technologies. Generally, $\pm 10\%$ offsets are defined for the voltage swings during STA. Therefore, the critical path at $V_{\text{DD}} = 0.9V$ must have a maximum delay of 1ns, considering a clock source of 1GHz. Fig. 5.2 shows the path delay curves with the k values calculated using (5.1), with $V_{\text{DD}} = 0.9V$, $td = 1ns$, $\alpha = [1.0, 1.3]$ and $V_{\text{th}} = [0.36, 0.4]$. For a *conservative analysis*, $V_{\text{th}} = 0.4V$ and $\alpha = 1.3$ are selected, indicating larger delay variations for smaller voltage differences, with $k = 0.45$.

5.2.3 Performance Metric

In this work, the required timing margin is used to compare the performance of the ROC and the *phase-locked loop* (PLL). For the PLL, the margin is the difference between the critical path delay at the nominal voltage (V_{nom}) and at the minimum voltage (V_{min}):

$$margin_{\text{PLL}} \geq td(V_{\text{nom}}) - td(V_{\text{min}}). \tag{5.2}$$

The design of an ROC must consider the delay behavior of Fig. 5.2 in order to keep the clock period larger than the delay of the critical paths for any given voltage. For the simplification of this analysis, the delay behavior of the ROC and the critical paths are both given by (5.1) with the same parameters. Still, if the ROC has a larger V_{th} than the critical path, then margins may also be smaller.

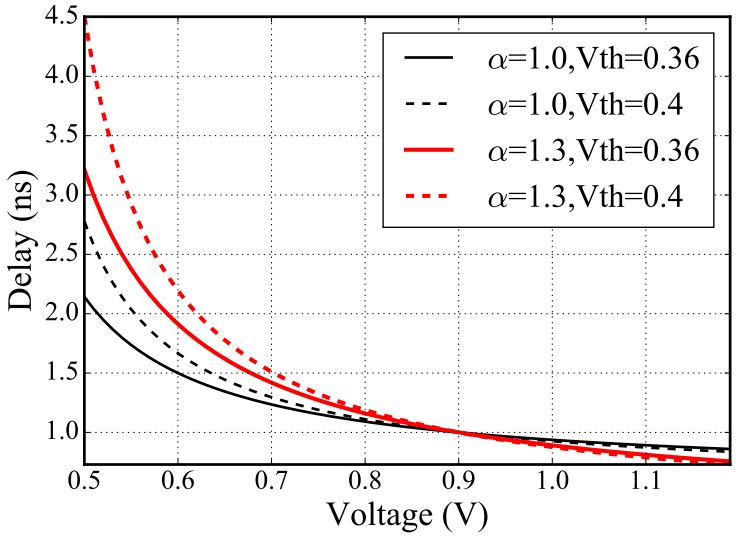


Figure 5.2: Path delay given by (5.1), with $td = 1ns$ and V_{DD}=0.9V.

In order to perform a *conservative analysis* of the required timing margins for the ROC, the following claims are made:

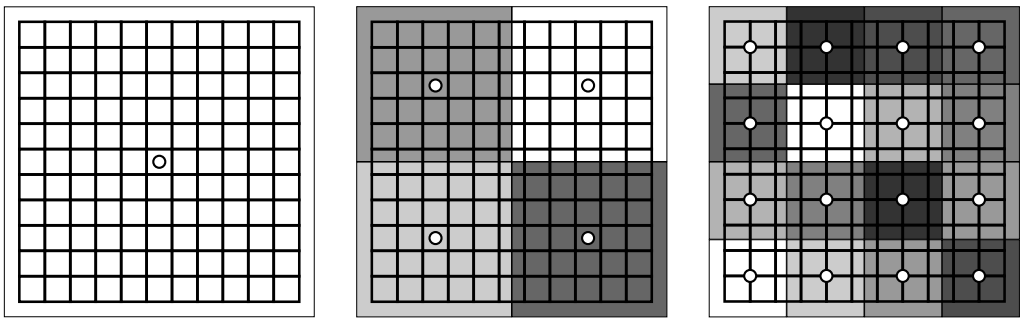
- The voltage at the ROC is always higher than at the critical path.
- The critical path is placed at the point with the largest voltage difference with respect to the ROC.
- The largest voltage difference happens at the minimum voltage, as delay variations are larger for lower voltages.
- Positive effects due to the clock distribution are not taken into account, such as clock-data compensation [119].

Thus, the margin for the ROC is given by (5.3), which is the difference between the critical path delay at the minimum voltage and the ROC period at the largest voltage difference.

$$margin_{\mathrm{ROC}} \geq td(V_{\mathrm{min}} + \max(\Delta V_{\mathrm{DD}})) - td(V_{\mathrm{min}}) \tag{5.3}$$

The PLL margin is required regardless of its placement, as the clock period must consider the critical path delay at V_{min}. But the ROC margin varies with its location, as the voltage difference is smaller between points closer to each other.

Fig. 5.3 depicts the three placement strategies analyzed, with circles at ROC locations and squares around the grid points on the same clock domain:



(a) Center (b) 4 ROCs (c) 16 ROCs

Figure 5.3: Placement of ROCs for different number of clock domains.

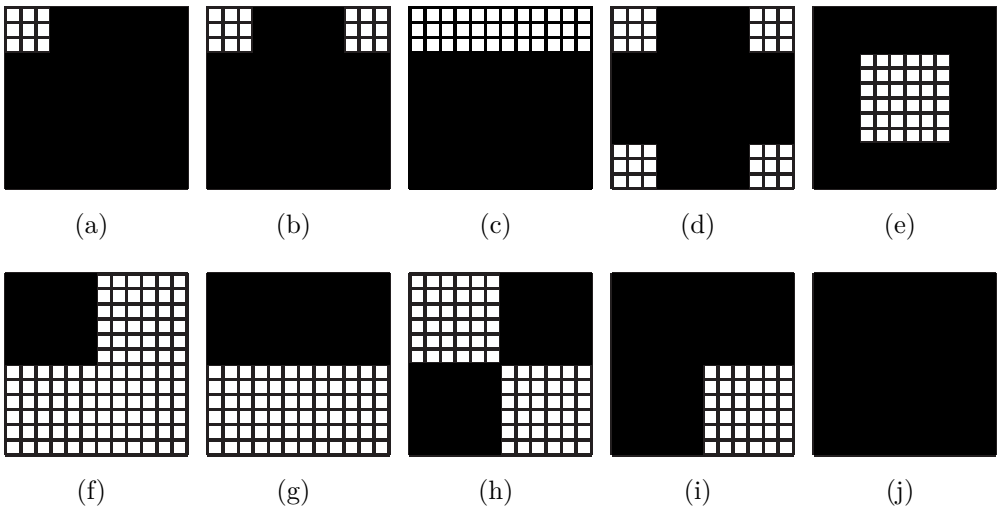


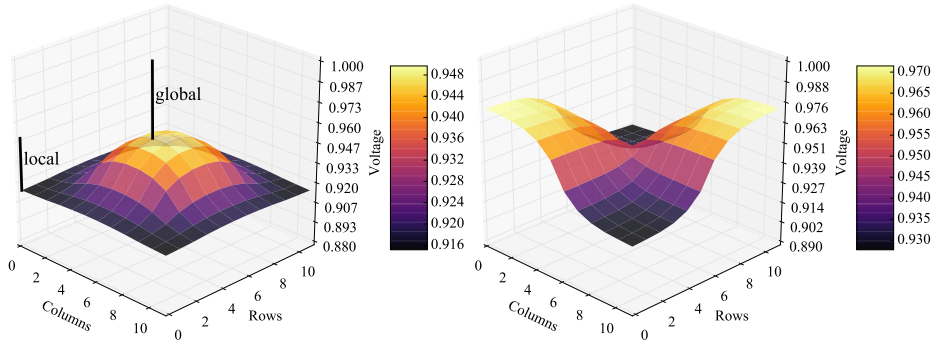
Figure 5.4: Patterns determining the grid points that are active.

one ROC at the center of the chip; 4 ROCs, with one at the center of each processor core; and 16 ROCs uniformly distributed. Additionally, one ROC placed at an arbitrary grid point is analyzed, reporting the placement that requires the largest margin. Notice that 16 ROCs would require additional synchronization between the clock domains, with an overhead in performance and power not investigated. Therefore, this case is reported but its results are not compared with the PLL.

5.3 Voltage locality analysis

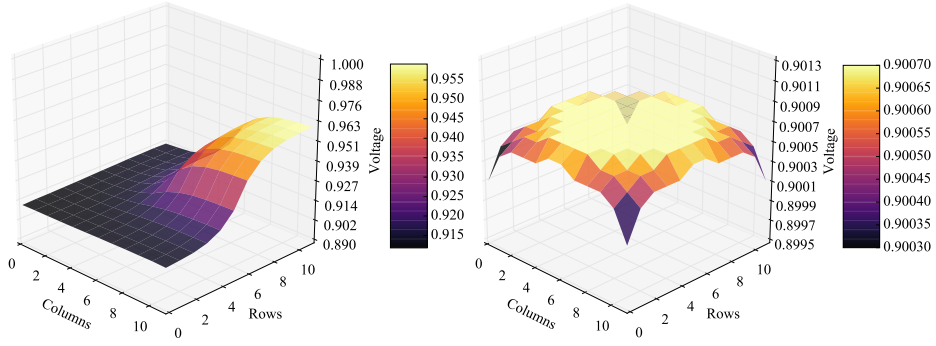
The different patterns depicted in Fig. 5.4 are proposed to stimulate voltage variations across the die. The dark areas represent the portions of the chip that are active, i.e., with a current source of 195mA active at every clock cycle, following the waveform in Fig. 5.1(a). The parts of the die that are not active are modeled with constant current sources of 3mA.

Fig. 5.5 illustrates the global and local voltage variations due to some of the proposed patterns in Fig. 5.4. These images show the voltage levels at each grid point when the minimum voltage is reached during the simulation. The pattern in Fig. 5.4(j) generates the lowest voltage, reaching a maximum current of 28A. An on-chip decoupling capacitance of 200nF is necessary to keep the voltage swings within ±10% for this activity pattern, considering an activity frequency of 1GHz.



(a) Pattern of Fig. 5.4(e)

(b) Pattern of Fig. 5.4(h)



(c) Pattern of Fig. 5.4(i)

(d) Pattern of Fig. 5.4(j)

Figure 5.5: Voltage distribution for some of the activity patterns in Fig. 5.4.

The grid model with 200nF of total on-chip decoupling capacitance is selected. The activity patterns of Fig. 5.4 are simulated with Synopsys

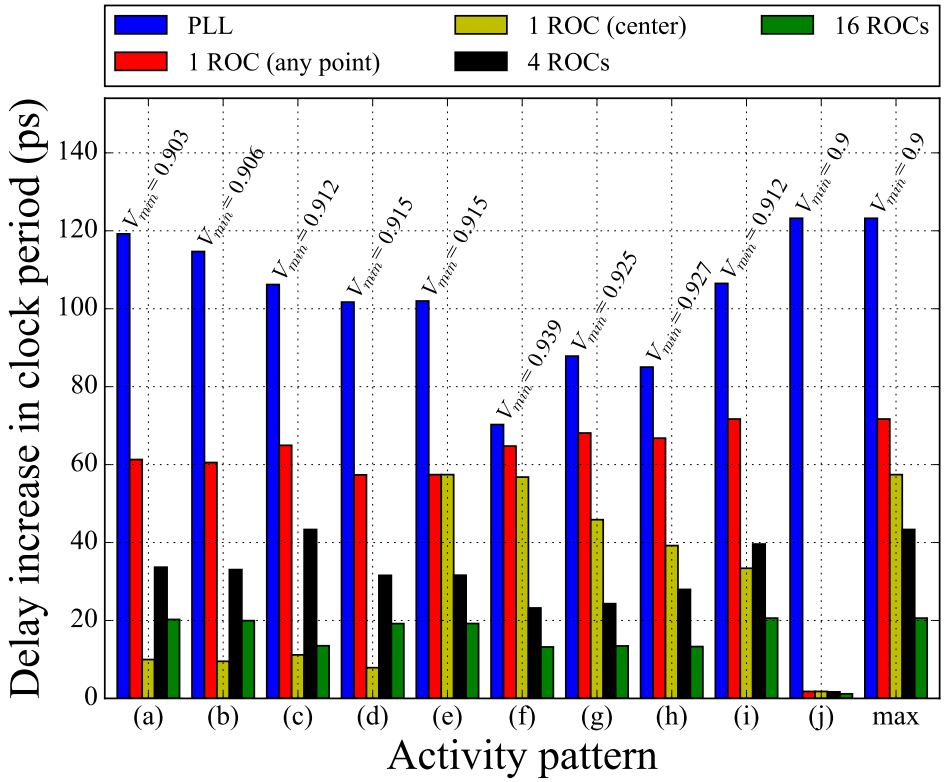


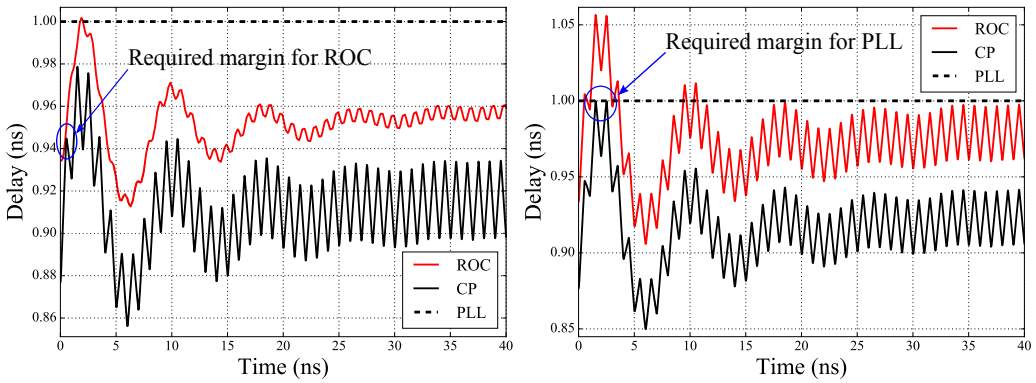
Figure 5.6: Delay increase in the clock period for each activity pattern (200nF of on-chip decaps, activity at 1GHz).

HSPICE® for 50 clock cycles at 125^{o}C, gathering the minimum voltage ($V_{\min}$) at all grid points, and the maximum voltage difference between any two points in the grid (ΔV_{DD}). Two different cases are analyzed: the typical voltage noise, generated by the designed clock period, which has low impedance (1GHz); and the worst-case voltage noise, caused by an activity frequency with very high impedance (first droop).

5.3.1 Typical voltage noise

Fig. 5.6 is generated with the voltage data gathered, using (5.2) and (5.3) to derive the required timing margins. The delay increase for the PLL is proportional to the number of active points, which is related with the total current and the minimum voltage. In the worst case for the PLL, $V_{\min} = 0.9V$ and the delay increase is 123ps.

For the ROC, the delay increase is related with the voltage difference between the ROC and the *critical path* (CP). Considering all activity patterns,



(a) Fig. 5.4(e) (b) Fig. 5.4(j)

Figure 5.7: Critical path delay, and the clock period of the PLL and the ROC, for the activity patterns of Fig. 5.4(e) and Fig. 5.4(j).

delay increase is 57ps if the ROC is placed at the center of the die and 71ps if it is placed at any grid point, i.e., the two points with the largest voltage difference.

In Fig. 5.7, the activity patterns of Fig. 5.4(e) and Fig. 5.4(j) are simulated, keeping track of the voltage at the center of the grid and at the point with the largest voltage difference. A 57ps margin is added to the ROC period, as it is considered to be placed at the grid center.

Fig. 5.7(a) depicts the worst case for the ROC, whereas Fig. 5.7(b) shows the largest delay of the critical path. Notice that the first and second voltage droops are present. As these effects are global, they affect the critical path and the ROC similarly. Therefore, ROCs enable a 53% better average performance for the same level of robustness against voltage noise.

Fig. 5.8 depicts the largest delay increase for each distance between any two grid points, considering all activity patterns. As expected, the delay is smaller if the critical path is closer to the ROC. This graph shows that a *trade-off* is possible between performance and the number of clock domains. The required delay is reduced to 43ps with 4 ROCs, and to 20ps with 16 ROC domains.

5.3.2 Worst-case voltage noise

The delay increase shown in Fig. 5.6 is required for a typical voltage noise, but larger voltage droops may happen if the activity frequency has a high impedance associated, as seen in Section 2.2.2.

The first droop frequency of the grid model with 200nF of on-chip decaps

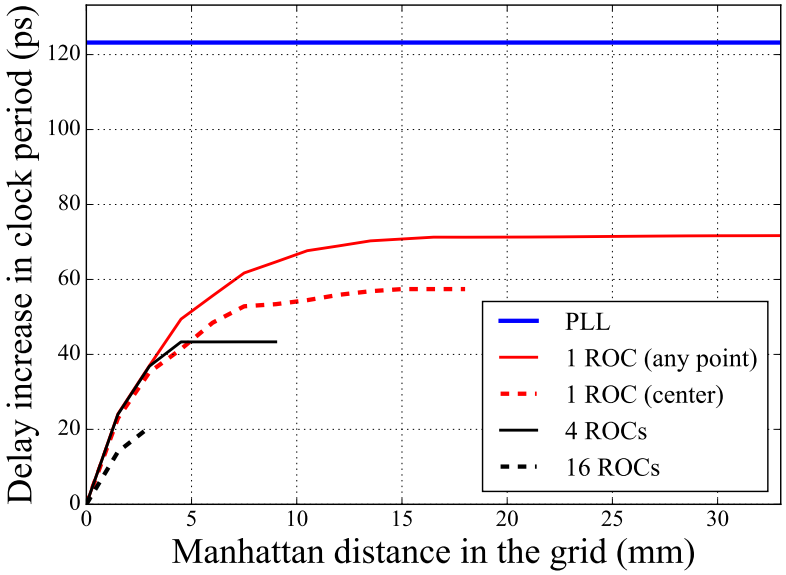


Figure 5.8: Largest delay increase vs. the distance between the ROC and the critical path (200nF decaps, activity at 1GHz).

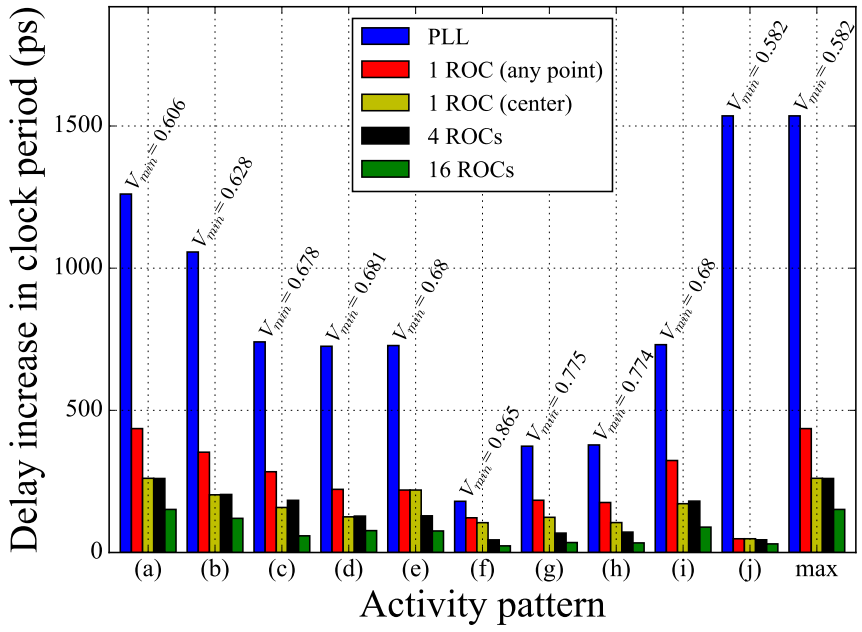


Figure 5.9: Delay increase in the clock period for each activity pattern, for the PLL and the ROC (200nF of on-chip decaps, activity at first droop).

is 125MHz. As a result, the voltage noise is amplified if a large current difference happens every 8 clock cycles, considering a clock source of 1GHz. In order to evaluate this phenomenon, the previous experiment is repeated with the current sources operating at 125MHz. Fig. 5.9 depicts the delay increase for each activity pattern in this case. As expected, the voltage noise is boosted due to the high impedance, and the delay increase required for the PLL is 1.5ns. Therefore, if worst-case voltage noise is considered, a design with a PLL *cannot* operate at 1GHz with this PDN.

The ROC takes advantage of the global characteristic of voltage droops, and the delay increase is 435ps if it is placed at an arbitrary point, and 260ps if placed at the center. Hence, it is possible to reduce the delay in 83%, without increasing the number of clock domains. Also, it is possible to reduce margins by increasing ROC domains, with a delay increase of only 151ps with 16 ROCs, which is comparable to the delay increase of the PLL for a typical voltage noise.

5.4 Relaxing PDN parameters

The design of the PDN is a difficult task that must take into account the circuit specification, the decaps, and the parasitics. It is necessary to adjust the characteristics of the PDN in order to avoid undesired voltage droops, which may happen when the switching activity is aligned with a resonance frequency of the PDN. This section shows how the robustness of ROCs contributes to *relax* the PDN design constraints, given the tolerance to handle global voltage variations. Three parameters are analyzed: on-chip decoupling capacitance, the number and placement of power bumps, and the parasitics of the package decaps.

5.4.1 On-chip decoupling capacitance

Fig. 5.10 depicts the impedance response of the PDN with 200nF, 300nF, 400nF, and 500nF of on-chip decoupling capacitance. Notice that adding decaps to the chip has a linear increase in area and power, whereas the impedance reduction is important, but a power-law function.

The reduction in voltage noise obtained by increasing the on-chip decaps has a direct impact in the performance, as seen in Fig. 5.11(a), where all activity patterns of Fig. 5.4 are simulated for different amounts of on-chip decaps, with activity at 1GHz. The behavior is similar with the activity aligned with the first droop, with significant margin reductions in Fig. 5.11(b). Notice that Fig. 5.11 presents the worst case result considering all activity patterns,

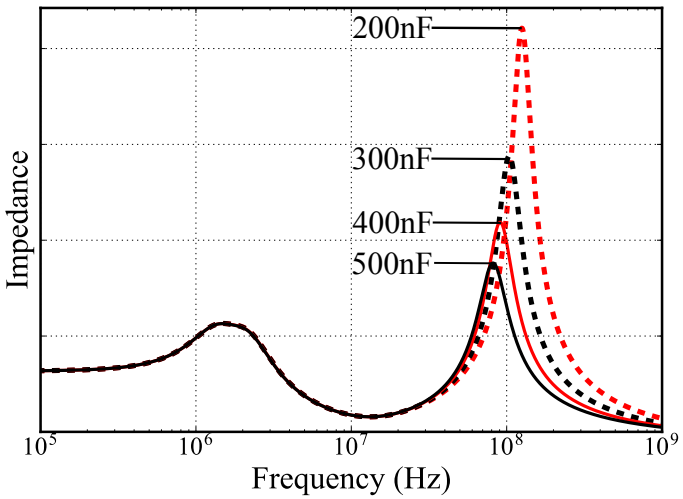


Figure 5.10: Impedance response of the PDN with 200nF, 300nF, 400nF and 500nF of on-chip decoupling capacitance.

as the "*max*" result on Fig. 5.6 and Fig. 5.9.

Notice that the first droop frequency varies with the amount of on-chip capacitance (see Fig. 5.10). The lower impedance is one of the reasons for the performance improvements seen in Fig. 5.11. Still, there is a saturation on the positive effect of adding decaps.

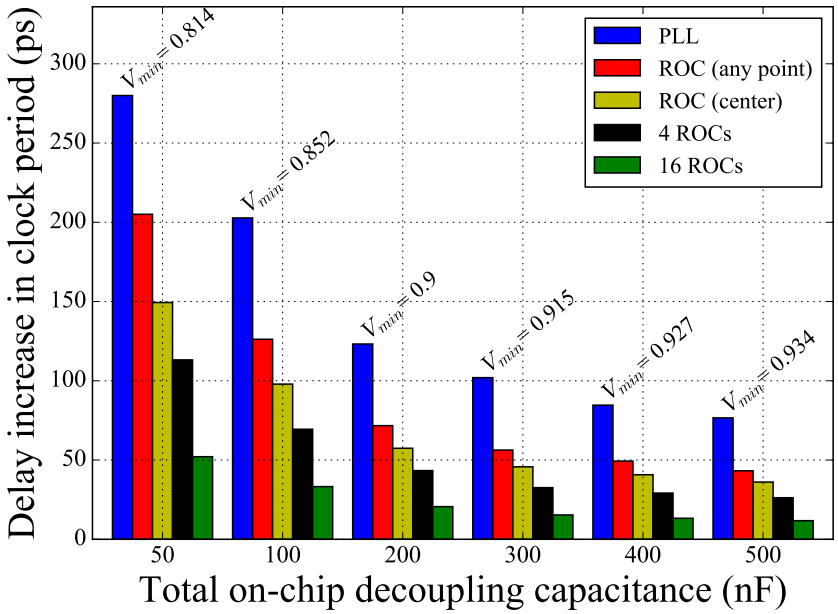
Generally, on-chip decaps do not imply an increase in area, given that the core utilization for standard cells is typically 70-90%, and decaps are placed in the white space. Still, the leakage power consumption of the decaps is important. As ROCs support larger voltage fluctuations with lower margins than static clocks, it is possible to reduce the amount of decaps and leakage power without degrading performance.

Leakage power can be modeled by expression (5.4), where P^{sq}_{std} and P^{sq}_{dec} are the leakage power per area of the standard cells and the decoupling capacitors, respectively. The area occupied by standard cells and decaps are A_{std} and A_{dec}, respectively.

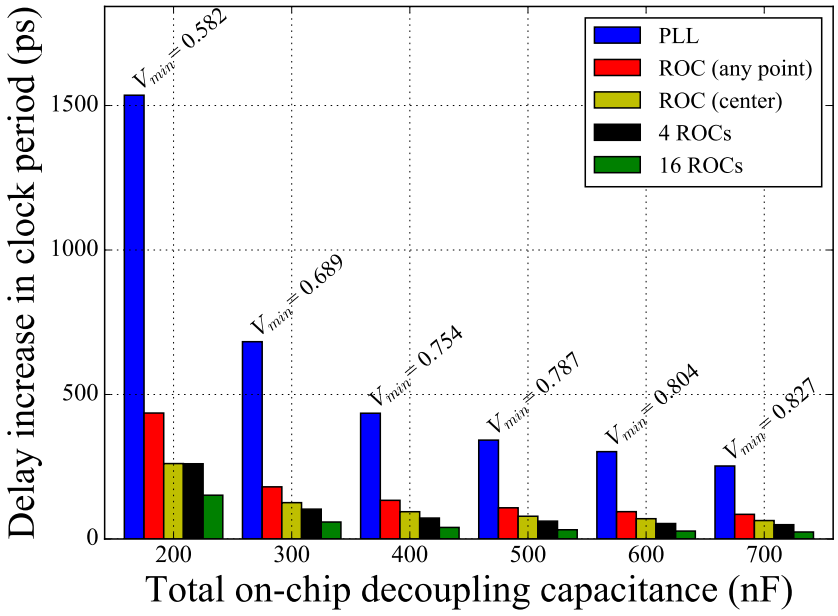
$$P_{leak} = P^{sq}_{std} \cdot A_{std} + P^{sq}_{dec} \cdot A_{dec} \tag{5.4}$$

The leakage savings are estimated by using the parameters of a commercial 65nm library. The least leaky decap cell is selected, with a capacitance per area of 6nF/mm^2 and leakage power consumption of 2.5mW/nF. Hence, the leakage power per area of decaps is defined as 15mW/mm^2.

For standard cells, leakage per area is estimated based on a design with a representative mix of combinational gates and flip-flops [64], obtaining



(a) Activity at 1GHZ



(b) Activity at first droop frequency

Figure 5.11: Delay increase for the PLL and ROC, with different amounts of on-chip decoupling capacitance.

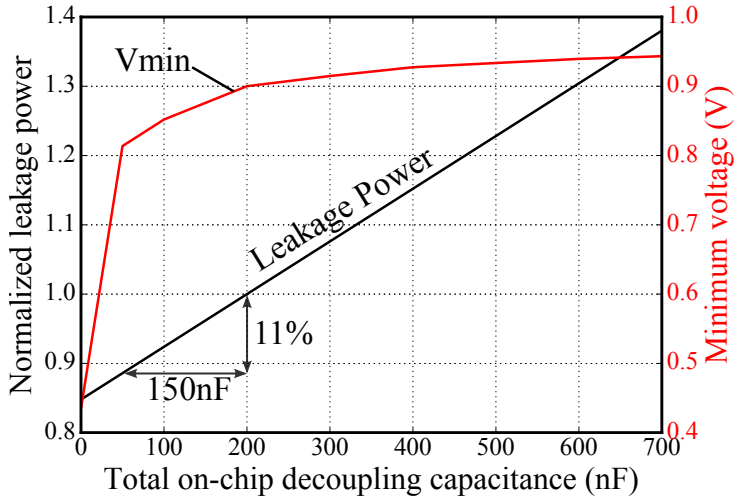


Figure 5.12: Normalized leakage power and minimum voltage for different amounts of on-chip decoupling capacitance (activity at 1GHZ).

20.9mW/mm^2. These values are *conservative*, as decaps typically have a larger average leakage power than standard cells. For the area ratio, it is assumed that 200nF represent 20% of the core area (utilization of 80%).

Fig. 5.12 shows the leakage power and the minimum voltage for different amounts of on-chip decaps, for typical voltage noise. Leakage power is normalized with respect to 200nF. Considering the margins seen in Fig. 5.11(a), it is possible to reduce up to 150nF in decaps without degrading performance, by using ROCs. This reduction represents 11% of the total leakage power.

Similarly, Fig. 5.13 depicts the leakage and minimum voltage, but for the worst-case voltage noise produced by activity aligned with first droop frequency. In this case, leakage power is normalized with respect to 700nF. Considering the data in Fig. 5.11(b), it is possible to have 200nF decaps instead of 700nF, without degrading average performance, with ROCs. Removing 500nF means a reduction of 27% in the total design leakage power consumption. Furthermore, if 200nF occupy all the white space, then 700nF entail a non-negligible area increase that can be avoided by using ROCs.

5.4.2 Power interconnections

The amount (and placement) of power bumps is another characteristic that influences voltage locality. The experiments in previous sections were performed with 72 pairs of V_{DD}/V_{SS} bumps uniformly distributed (Fig. 5.14(a)). This placement minimizes the impedance between the chip and the package [99], and any grid point has practically the same impedance response.

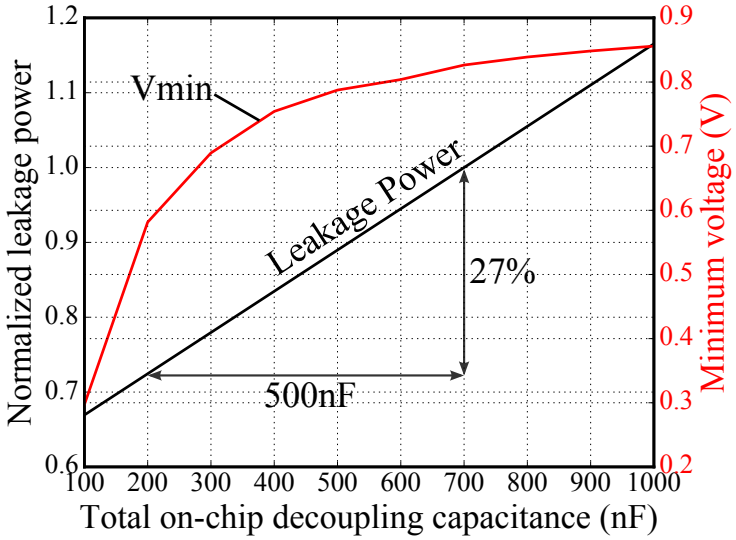
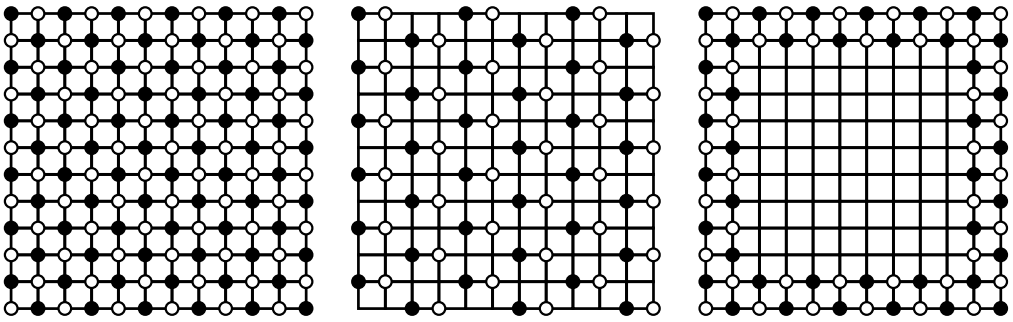


Figure 5.13: Normalized leakage power and minimum voltage for different amounts of on-chip decoupling capacitance (activity at first droop frequency).

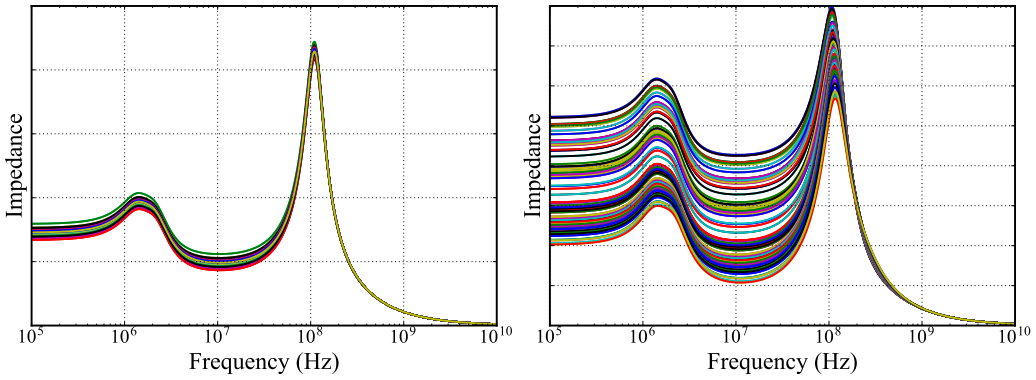


(a) All points (b) Distributed (c) Border

Figure 5.14: Different power bumps placement strategies (a V_{DD} connection is a black circle, and V_{SS} connection is a white circle).

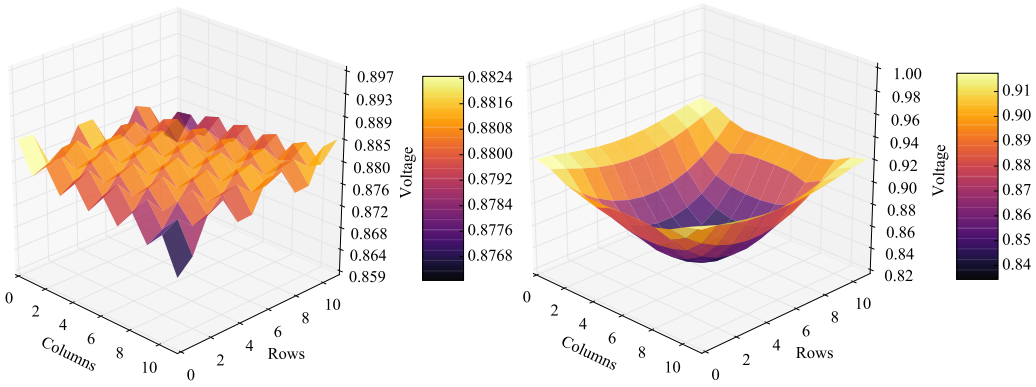
As seen in Fig. 5.5, such placement reduces significantly the voltage differences across the die.

This section considers different bump placements in the grid model with 200nF, for typical voltage noise (activity at 1GHz). Two additional placements are analyzed: 36 V_{DD}/V_{SS} pairs uniformly distributed, illustrated in Fig. 5.14(b); and 40 V_{DD}/V_{SS} pairs placed in the borders (similar to wire bonding), depicted in Fig. 5.14(c). These placements affect the impedance response across the die, as observed in Fig. 5.15. Such configurations also have a huge impact in the power distribution (see Fig. 5.16).



(a) Distributed (b) Border

Figure 5.15: Impedance response of all grid points (200nF of on-chip capacitance) with (a) 36 bumps distributed and (b) 40 bumps in the borders.



(a) Distributed (b) Border

Figure 5.16: Voltage distribution for activity pattern of Fig. 5.4(j) with (a) 36 V_{DD}/V_{SS} bumps distributed ($V_{min} = 0.872V$), and (b) 40 V_{DD}/V_{SS} bumps in the borders ($V_{min} = 0.837V$).

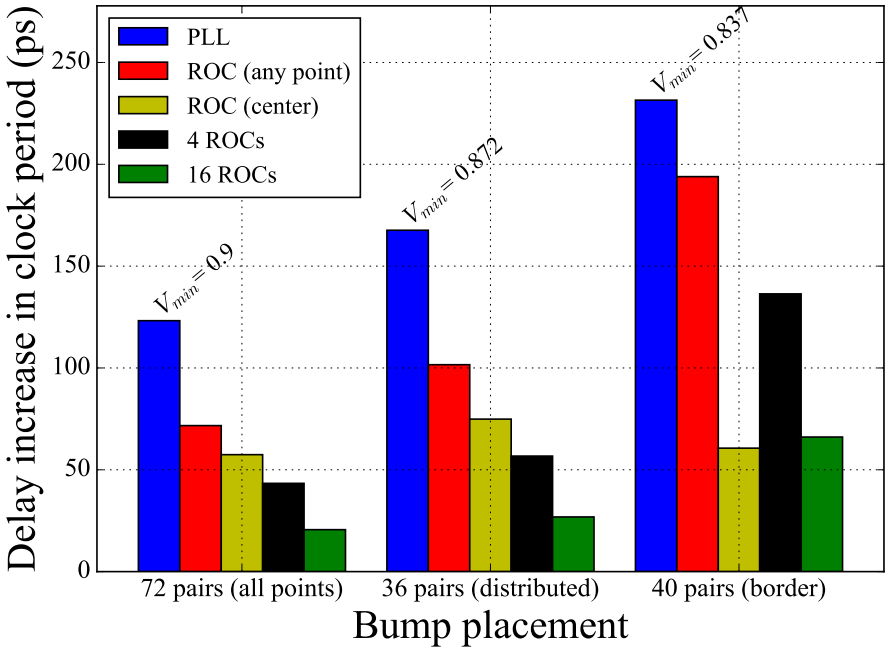


Figure 5.17: Required margins for the PLL and ROC with different bump placements (200nF of decoupling capacitance, activity at 1GHz).

All activity patterns are simulated, producing the results of Fig. 5.17. As the impedance is higher, the minimum voltage is lower, indicating larger margins. Also, ROC margins have an important increase, due to larger voltage differences. Still, it is possible to reduce the bumps configuration using ROCs, with same or better performance of a PLL. With bumps placed in the border, it is possible to take further advantage of ROC characteristics by placing it at the center. In this case, the ROC will have the lowest voltage in the die, producing a clock slower than the critical paths during the periods of voltage noise, and therefore enabling a higher average performance.

5.4.3 Package decoupling capacitance parasitics

The design of the PDN is a key factor in the quality of the supply voltage at the chip devices, with the board and package playing important roles in the solution. Small parasitics in the off-chip PDN may have a great impact in the global voltage variations. This section proposes an analysis with different parasitics in the package decoupling capacitance:

- Package 1 (*PKG1*): the same used in previous sections, with typical *equivalent series inductance* (ESL): $L_{\text{cpkg}} = 5.61pH$.

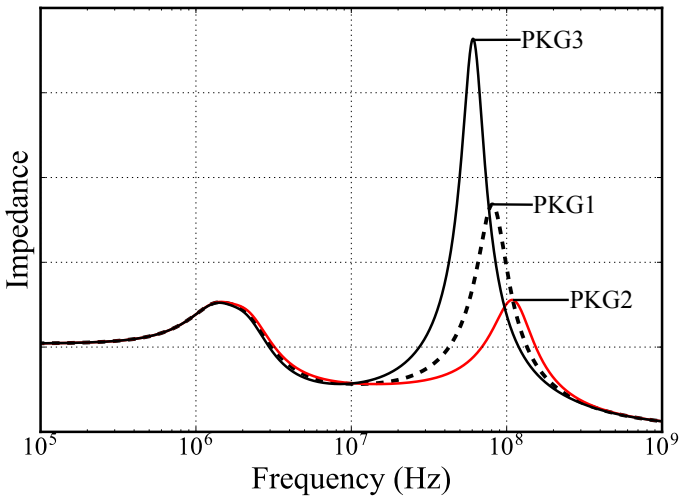


Figure 5.18: Impedance responses with 500nF of on-chip capacitance and different package decap parasitics.

- Package 2 (*PKG2*): with almost ideal decoupling capacitance, maximizing voltage noise reduction: $L_{\text{cpkg}} = 2pH$.
- Package 3 (*PKG3*): using decaps with higher inductive parasitics, increasing the equivalent inductance that forms the LC circuit with the die capacitance: $L_{\text{cpkg}} = 12pH$.

In order to enforce a voltage variation of >10% for all cases and compare their impact on the reference performance, all current sources are active and aligned with the first droop frequency, which is different for each package (see Fig. 5.18), with a total on-chip decoupling capacitance of 500nF. This configuration generates voltage swings large enough to provoke a voltage emergency for *PKG2*, and to keep the delay increase less than 1ns for *PKG3*.

Fig. 5.18 depicts the impedance responses of the 3 distinct packages. Notice that the ESL parasitics in the package decaps have a massive influence in the quality of the PDN. The very low inductance of the *PKG2* decaps results in a lower impedance at the first droop and a great voltage noise mitigation. On the opposite side, the decaps with higher ESL of *PKG3* increase the equivalent inductance connected to the chip, resulting in a higher peak impedance. In practice, *PKG3* can be used as a reference in terms of impedance as if the flip chip interconnection would be replaced by a wire bonding, which it is known to have higher impedance and to be cheaper [39].

Fig. 5.19 shows the delay increase for the PLL and the different ROC configurations, taking into account all activity patterns of Fig. 5.4. For the PLL, it is necessary to cover the deepest droops and to ensure that the delay

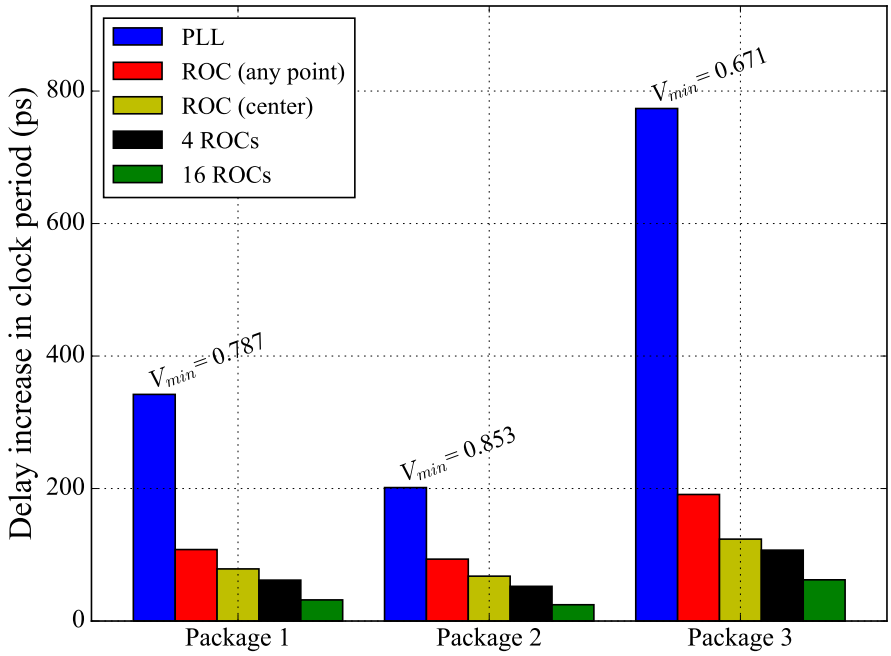


Figure 5.19: Required margins for the PLL and ROC with the different package decap parasitics (500nF of on-chip decaps, activity at first droop).

of the critical paths are always shorter than the clock period. The largest generated droop was -329mV, leading to a performance degradation of up to 84% with *PKG3*, comparing the delay increase of the PLL (773ns) with the ROC in the center of the grid (123ns). As seen in Section 5.3.2, ROCs take advantage of the global characteristics of voltage droops, requiring smaller margins and achieving higher average performance.

5.5 Discussion

Voltage droops have a great impact in the performance when using rigid clocks. For this reason, a significant effort must be invested in designing high-quality PDNs: adding decaps at all levels, reducing the impedance at each interconnection, considering the frequency response w.r.t. the activity of the circuit, and using elements with low parasitics and *low variability*.

Section 5.4 presented different and illustrative configurations of the PDN, demonstrating how harmful low-quality PDNs can be for PLLs. ROCs provide a better alternative to tackle power integrity problems without degrading performance. This section presents a summary of advantages and disadvantages of using ROCs as the clock source.

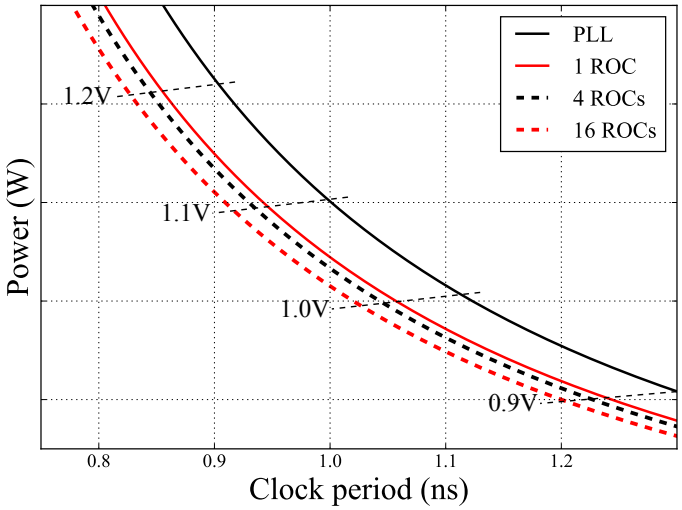


Figure 5.20: Power/Performance trade-off for ±10% voltage noise.

5.5.1 Simpler voltage/frequency scaling

ROCs offer instantaneous adaptation to static and dynamic variability. Such characteristic can be used for a simpler version of *dynamic voltage and frequency scaling* (DVFS). Differently from the DVFS techniques currently used, in which both frequency and voltage must be controlled, with ROCs it is possible to define the performance only with the voltage.

Furthermore, voltage scaling can be used for an improved trade-off of power/performance [28]. Fig. 5.20 depicts the trade-off between power and performance for the PLL and the different ROC strategies, with iso-voltage curves. Notice that ROCs naturally adapt to the process variability, and voltage scaling can be used after fabrication in order to find the *minimum energy point* for the performance required.

5.5.2 EMI reduction

Electromagnetic interference (EMI) is an aspect that must be considered to comply with the regulations in the application domain. In digital systems, EMI is mostly produced by the periodic current differences generated by clock edges.

Several techniques can be used to mitigate electromagnetic radiations. Shielding is often used to isolate the product from the external world, but this method has a significant cost.

A less costly approach is the use of spread-spectrum clock generators, that outspread the energy over a wider bandwidth, reducing peak ampli-

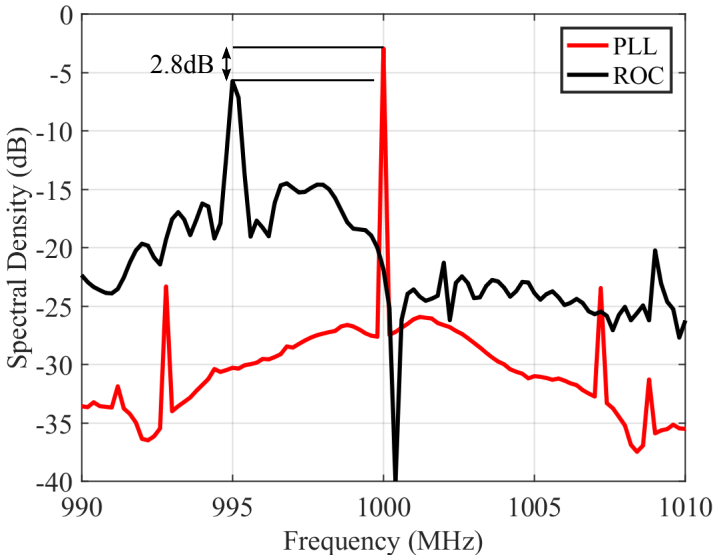


Figure 5.21: Frequency spectrum comparison of ROC vs. PLL.

tude [44, 52]. This technique consists of inserting intentional jitter to the clock generator, which implies additional timing margins. The presence of dynamic variations implicitly injects jitter to the clock period of ROCs. Fortunately, this jitter does not need to be margined since the period variability is correlated with the circuit delays. Therefore, a *natural* spread-spectrum effect is produced without affecting performance.

The frequency spectrum of a ROC and a PLL are compared in Fig. 5.21. As expected, the jitter introduced by the ROC produces a spread spectrum effect, reducing the peak amplitude in 2.8dB. For comparison, [33] reports a 13dB peak reduction for a ±3% spread in their design, which implies a 3% performance degradation. It is important to bear in mind that these results are not measured, but simulated. Also, the only source of variability analyzed is the supply voltage, in a very simple and periodic current profile. Therefore, the results shown in this section indicate that the EMI reduction can be larger for real designs using ROCs, with both switching activity randomness and variability increased. Furthermore, any EMI reduction produced by an ROC comes *for free*, i.e., does not degrade performance and does not require any explicit mechanism to address EMI.

5.5.3 Benefits of multiple ROC domains

In a *globally asynchronous locally synchronous* (GALS) design methodology with multiple ROC domains, the period of each ROC is defined based on the critical path within the local clock domain, and not on the global worst-case. Thus, EMI reduction benefits can be boosted [55], while side-channel security is also improved. In addition, clock tree synthesis is simpler with smaller clock domains, whereas power consumption can be minimized with lower clock frequencies.

5.5.4 Disadvantages

ROCs can *surf* over deep voltage fluctuations while sustaining an average performance. This comes at the expense of a clock period with high jitter and potentially large frequency variations. Systems operating with ROCs must tolerate these characteristics along the executing time of the applications.

It is difficult to design an ROC with a stable duty cycle, and the duty cycle cannot be guaranteed. Therefore, this may be a limitation for applications that require both clock edges, such as *double data rate* (DDR) memories. However, a simple solution is to use multiple clock sources, e.g., a PLL with 50% duty cycle for the memory interface, and ROCs for the random logic.

The GALS methodology has an important characteristic: it requires *cross-domain crossing* (CDC) techniques to be applied between the different ROC regions. There are several known techniques that perform CDC [55]. Each technique has its pros and cons, but there is an overhead in area, power and throughput, independently of the approach defined. Still, for multi-core or very large chips, the use of multiple clocks is already required [107], and the use of multiple ROC domains could be applied without additional costs.

Chapter 6

Conclusions and Future Work

In this thesis, contributions were proposed for improving *power, performance, area, and cost* (PPAC) using established *integrated circuit* (IC) manufacturing technologies. Advances in *electronic design automation* (EDA) were investigated in three distinct topics: technology-independent area minimization, decomposition and remapping methods for *field-programmable gate arrays* (FPGAs) based on *look-up tables* (LUTs), and an adaptive clocking scheme based on ROCs in order to improve performance and power consumption of digital circuits, and costs on the PDN design. This chapter presents a summary of the contributions, and provides ideas for future research that can use the present thesis as basis.

6.1 Summary of the thesis contributions

The *first contribution* is a technology-independent method for area minimization of combinational logic, based on a multi-output decomposition using two-literal divisors. An *and-inverter graph* (AIG) local optimization approach is implemented, applying multi-output Boolean division on KL-cuts. The experiments show promising results, with an average node reduction of 7.8% in comparison with highly optimized AIGs. Similar reductions are observed after technology mapping, with area improvements of 5.5% for FPGA mapping, and 6.2% for standard cells.

The *second contribution* regards the proposition of two methods targeting LUT-based FPGAs: a functional decomposition which uses the support size as cost function, and a recursive remapping. The support-reducing decomposition produces a subject graph with a structure more suitable to FPGA mapping. The recursive remapping approach reduces the structural bias of the circuit, using the actual FPGA mapping result as cost function. The

experiments show very promising results. The combination of the proposed methods improve the FPGA mapping results of a commercial tool for the MCNC benchmarks, with gains of 28% in delay plus 18% in area when targeting delay, and 28% in area plus 14% in delay with area as cost function. Average results after physical synthesis show 23% less area and 6% less delay (or 20% less area and 9% less delay) by using the remapping results as input to the commercial tool instead of the initial descriptions. In comparison with BoolMap [61], a BDD-based method which takes advantage of another support-reducing decomposition approach, the present work is able to obtain 18% fewer LUTs and 8% fewer logic levels. Moreover, 12 of the best known results for delay (and 6 for area) of the EPFL benchmarks are updated.

Power integrity is a major concern due to low supply voltages and high power density in high-performance circuits. The *third contribution* consists of an extensive analysis on the dynamic variability mitigation and simplification of PDNs using an adaptive clocking scheme based on ROCs. The analysis shows that ROCs provide a robust clock scheme that tolerates large fluctuations in the supply voltages. ROCs are a competitive alternative to the rigid clocks generated by PLLs, with reductions of up to 83% in performance margins and up to 27% in leakage power. Notice that the design of a PDN is an arduous task that must consider the circuit characteristics in order to deliver high-quality supply voltages. It was shown that the PDN design constraints can be relaxed, without performance loss, by using an ROC as the clock source. Tolerance to voltage noise and related benefits can also be increased with multiple ROC domains. Additionally, with the increasing importance of the *internet of things* (IoT), we are facing a future in which many devices will have to operate in environments with scarce energy in which scavenging mechanisms will be essential to survive. Providing reliable supply voltages under these scenarios may be difficult and costly. ROCs emerge as a potential solution to operate robustly in hostile environments with low-cost PDNs. Furthermore, considering the use of integrated circuits in safety-critical applications, the ROCs characteristic of adapting to undesirable operating conditions may be crucial to support situations of limited energy or large voltage noise.

6.2 Future work

Scalability is one of the aspects that requires more investigation in the first contribution. We envision a synthesis system in which smart oracles could guide the search for divisors based on simple correlation metrics between functions and divisors. As future work, different types of cuts and combina-

tions of divisors could be studied. Using other models of flexibility (Boolean relations) could also be considered. Delay is another important aspect that is not considered in this work, but could be incorporated by controlling the number of levels and reducing the resulting circuit delay.

Regarding the second contribution, there are some directions that could be explored. A partial collapsing approach that uses the FPGA mapping result as cost function could be investigated. Additional support-reducing techniques could be incorporated, such as the bi-decomposition methods proposed in [87, 25, 60]. For the recursive collapsing approach, the propagation of the don't care conditions could potentially improve the results. Also, keeping track of the critical paths may allow further area reduction while obtaining similar delay results.

For the third contribution, actual measurements on an implementation in FPGA or *application specific integrated circuit* (ASIC) would be an interesting future research. In that sense, the result of voltage measurements varying the location and number of ROCs would provide more precise timing margins, indicating if our simulation-based model analysis is too conservative or not. Moreover, the analysis of EMI reduction could be performed with different benchmarks, likely presenting better results.

Bibliography

[1] ITRS: International Technology Roadmap for Semiconductors, 2015. Available at `http://www.itrs2.net/itrs-reports.html`.

[2] K. Agarwal and S. Nassif. Characterizing process variation in nanometer CMOS. In *Proceedings of the Design Automation Conference*, pages 396–399. ACM, 2007.

[3] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.

[4] C. Albrecht. IWLS 2005 benchmarks, 2005. Available at `http://iwls.org/iwls2005/benchmarks.html`.

[5] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar. *Handbook of algorithms for physical design automation*. CRC Press, Boca Raton, FL, USA, 2008.

[6] L. Amarú, P.-E. Gaillardon, and G. De Micheli. The EPFL combinational benchmark suite. In *Proceedings of the International Workshop on Logic & Synthesis*, pages 57–61, 2015. Available at `https://github.com/lsils/benchmarks`.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[8] R. L. Ashenhurst. The decomposition of switching functions. In *Proceedings of the International Symposium on the Theory of Switching*, pages 74–116, 1957.

[9] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[10] D. Baneres, J. Cortadella, and M. Kishinevsky. Timing-driven N-way decomposition. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 363–368. ACM, 2009.

[11] V. Bertacco and M. Damiani. The disjunctive decomposition of logic functions. In *Proceedings of the International Conference on Computer-Aided Design*, pages 78–82. IEEE Computer Society, 1997.

[12] K. A. Bowman, C. Tokunaga, T. Karnik, V. K. De, and J. W. Tschanz. A 22 nm all-digital dynamically adaptive clock distribution for supply voltage droop tolerance. *IEEE Journal of Solid-State Circuits*, 48(4):907–916, 2013.

[13] R. Brayton and C. McMullen. The decomposition and factorization of Boolean expressions. In *Proceedings of the International Symposium on Circuits and Systems*, pages 49–54, 1982.

[14] R. Brayton and A. Mishchenko. ABC: An academic industrial-strength verification tool. In *Proceedings of the International Conference on Computer-Aided Verification*, pages 24–40, 2010. Available at `https://github.com/berkeley-abc/abc`.

[15] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel. *Logic minimization algorithms for VLSI synthesis*. Kluwer, Norwell, MA, USA, 1984.

[16] F. M. Brown. *Boolean reasoning: the logic of Boolean equations*. Springer, New York, NY, USA, 2012.

[17] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 100(8):677–691, 1986.

[18] V. Callegaro, F. S. Marranghello, M. G. Martins, R. P. Ribas, and A. I. Reis. Bottom-up disjoint-support decomposition based on cofactor and Boolean difference analysis. In *Proceedings of the IEEE International Conference on Computer Design*, pages 680–687. IEEE, 2015.

[19] R. K. Cavin, P. Lugli, and V. V. Zhirnov. Science and engineering beyond Moore's law. *Proceedings of the IEEE*, 100:1720–1749, 2012.

[20] S. Chatterjee, A. Mishchenko, and R. Brayton. Factor cuts. In *Proceedings of the International Conference on Computer-Aided Design*, pages 143–150. IEEE, 2006.

[21] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam. Reducing structural bias in technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2894–2903, 2006.

[22] D. Chen and J. Cong. DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 752–759, 2004.

[23] L. Cheng, D. Chen, and M. D. Wong. DDBDD: Delay-driven BDD synthesis for FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1203–1213, 2008.

[24] Y.-k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei. A quantitative analysis on microarchitectures of modern CPU-FPGA platforms. In *Proceedings of the Design Automation Conference*, pages 109:1–109:6. ACM, 2016.

[25] M. Choudhury and K. Mohanram. Bi-decomposition of large Boolean functions using blocking edge graphs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 586–591. IEEE, 2010.

[26] J. Cong and Y. Ding. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(1):1–12, 1994.

[27] J. Cortadella. Timing-driven logic bi-decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):675–685, 2003.

[28] J. Cortadella, L. Lavagno, P. López, M. Lupon, A. Moreno, A. Roca, and S. S. Sapatnekar. Reactive clocks with variability-tracking jitter. In *Proceedings of the International Conference on Computer Design*, pages 511–518. IEEE, 2015.

[29] J. Cortadella, M. Lupon, A. Moreno, A. Roca, and S. S. Sapatnekar. Ring oscillator clocks and margins. In *Proceedings of the International Symposium on Asynchronous Circuits and Systemshesis*, pages 19–26, 2016.

[30] R. Courtland. Gordon Moore: The man whose name means progress. *IEEE Spectrum*, 30, 2015.

[31] H. A. Curtis. *A New Approach to the Design of Switching Circuits*. D. Van Nostrand, Boston, MA, USA, 1962.

[32] M. Damiani and G. De Micheli. Don't care set specifications in combinational and synchronous logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(3):365–388, 1993.

[33] S. Damphousse, K. Ouici, A. Rizki, and M. Mallinson. All digital spread spectrum clock generator for EMI reduction. *IEEE Journal of Solid-State Circuits*, 42(1):145–150, 2007.

[34] G. De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill, Inc., New York, NY, USA, 1994.

[35] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of International Symposium on Computer Architecture*, pages 365–376. IEEE, 2011.

[36] I. Ferain, C. A. Colinge, and J.-P. Colinge. Multigate transistors as the future of classical metal–oxide–semiconductor field-effect transistors. *Nature*, 479:310–316, 2011.

[37] P. Fišer and J. Schmidt. Improving the iterative power of resynthesis. In *Proceedings of the IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pages 30–33. IEEE, 2012.

[38] P. Fišer, J. Schmidt, and J. Balcárek. Sources of bias in EDA tools and its influence. In *Proceedings of the International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pages 258–261, 2014.

[39] A. Fontanelli. System-in-package technology: opportunities and challenges. In *Proceedings of the International Symposium on Quality Electronic Design*, pages 589–593. IEEE, 2008.

[40] M. Fujita, Y. Matsunaga, Y. Tamiya, and K.-C. Chen. Multi-level logic minimization of large combinational circuits by partitioning. In *Logic Synthesis and Optimization*, pages 109–126. Springer, 1993.

[41] M. S. Gupta, J. L. Oatley, R. Joseph, G.-Y. Wei, and D. M. Brooks. Understanding voltage variations in chip multiprocessors using a distributed power-delivery network. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 624–629, 2007.

[42] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks. Towards a software approach to mitigate voltage emergencies. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 123–128, 2007.

[43] M. S. Gupta, V. J. Reddi, G. Holloway, G.-Y. Wei, and D. M. Brooks. An event-guided approach to reducing voltage noise in processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 160–165, 2009.

[44] K. B. Hardin, J. T. Fessler, and D. R. Bush. Spread spectrum clock generation for the reduction of radiated emissions. In *Proceedings of the IEEE International Symposium on Electromagnetic Compatibility*, pages 227–231, 1994.

[45] U. Hinsberger and R. Kolla. Boolean matching for large libraries. In *Proceedings of the Design Automation Conference*, pages 206–211. ACM, 1998.

[46] Y. Hong, P. A. Beerel, J. R. Burch, and K. L. McMillan. Safe BDD minimization using don't cares. In *Proceedings of the Design Automation Conference*, pages 208–213. ACM, 1997.

[47] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology*, 24(38):384010, 2013.

[48] Intel Corp. Stratix 10 GX/SX device overview, 2017. Available at `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10-overview.pdf`.

[49] R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 79–90. IEEE, 2003.

[50] D. A. Kamakshi, M. Fojtik, B. Khailany, S. Kudva, Y. Zhou, and B. H. Calhoun. Modeling and analysis of power supply noise tolerance with fine-grained gals adaptive clocks. In *Proceedings of the International Symposium on Asynchronous Circuits and Systemshesis*, pages 75–82, 2016.

[51] J. S. Kilby. Miniaturized electronic circuits, 1964. US Patent 3,138,743.

[52] J. Kim, D. G. Kam, P. J. Jun, and J. Kim. Spread spectrum clock generator with delay cell array to reduce electromagnetic interference. *IEEE Transactions on Electromagnetic Compatibility*, 47(4):908–920, 2005.

[53] V. N. Kravets and P. Kudva. Implicit enumeration of structural changes in circuit optimization. In *Proceedings of the Design Automation Conference*, pages 438–441. ACM, 2004.

[54] V. N. Kravets and K. A. Sakallah. Constructive library-aware synthesis using symmetries. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 208–215. ACM, 2000.

[55] M. Krstic, E. Grass, F. K. Gürkaynak, and P. Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design & Test of Computers*, 24(5):430–441, 2007.

[56] M. Kubica, A. Opara, and D. Kania. Logic synthesis for FPGAs based on cutting of BDD. *Microprocessors and Microsystems*, 52:173–187, 2017.

[57] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, 2007.

[58] N. Kurd, P. Mosalikanti, M. Neidengard, J. Douglas, and R. Kumar. Next generation Intel core^{TM} micro-architecture clocking. *IEEE Journal of Solid-State Circuits*, 44(4):1121–1129, 2009.

[59] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[60] R.-R. Lee, J.-H. R. Jiang, and W.-L. Hung. Bi-decomposing large Boolean functions via interpolation and satisfiability solving. In *Proceedings of the Design Automation Conference*, pages 636–641, 2008.

[61] C. Legl, B. Wurth, and K. Eckl. A Boolean approach to performance-directed technology mapping for LUT-based FPGA designs. In *Proceedings of the Design Automation Conference*, pages 730–733, 1996.

[62] C. Legl, B. Wurth, and K. Eckl. An implicit algorithm for support minimization during functional decomposition. In *Proceedings of the European conference on Design and Test*, pages 412–417. IEEE, 1996.

[63] Y.-M. Lin, A. Valdes-Garcia, S.-J. Han, D. B. Farmer, I. Meric, Y. Sun, Y. Wu, C. Dimitrakopoulos, A. Grill, P. Avouris, et al. Wafer-scale graphene integrated circuit. *Science*, 332(6035):1294–1297, 2011.

[64] M. Litochevski and L. Dongjun. High throughput and low area AES, 2012. Available at `https://opencores.org/project/aes_highthroughput_lowarea`.

[65] G. Liu and Z. Zhang. A parallelized iterative improvement approach to area optimization for LUT-based technology mapping. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 147–156. ACM, 2017.

[66] L. Machado and J. Cortadella. Boolean decomposition for aig optimization. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 143–148. ACM, 2017.

[67] L. Machado and J. Cortadella. Support-reducing decomposition for FPGA mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[68] L. Machado and J. Cortadella. Support-reducing functional decomposition for FPGA technology mapping. In *Proceedings of the International Workshop on Logic and Synthesis*, pages 79–86, 2018.

[69] L. Machado, M. G. A. Martins, V. Callegaro, R. P. Ribas, and A. I. Reis. KL-cut based digital circuit remapping. In *Proceedings of the IEEE Nordic Microelectronics Event*, pages 1–4. IEEE, 2012.

[70] L. Machado, M. G. A. Martins, V. Callegaro, R. P. Ribas, and A. I. Reis. Iterative remapping respecting timing constraints. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, pages 236–241. IEEE, 2013.

[71] L. Machado, A. Roca, and J. Cortadella. Increasing the robustness of digital circuits with ring oscillator clocks. In *Proceedings of the International Workshop on Resiliency in Embedded Electronic Systems*, pages 29–34, 2017.

[72] L. Machado, A. Roca, and J. Cortadella. Voltage noise analysis with ring oscillator clocks. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 1–6. IEEE, 2017.

[73] L. Machado, A. Roca, and J. Cortadella. Robustness to voltage noise with ring oscillator clocks. *IEEE Transactions on Nanotechnology*, 2018. (Submitted).

[74] D. MacMillen, R. Camposano, D. Hill, and T. W. Williams. An industrial view of electronic design automation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1428–1448, 2000.

[75] O. Martinello Jr, F. S. Marques, R. P. Ribas, and A. I. Reis. KL-cuts: a new approach for logic synthesis targeting multiple output blocks. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 777–782. EDAA, 2010.

[76] M. G. Martins, L. Rosa, A. B. Rasmussen, R. P. Ribas, and A. I. Reis. Boolean factoring with multi-objective goals. In *Proceedings of the International Conference on Computer Design*, pages 229–234. IEEE, 2010.

[77] A. Mishchenko. An approach to disjoint-support decomposition of logic functions, 2001. Technical Report. Portland State University.

[78] A. Mishchenko. Enumeration of irredundant circuit structures. In *Proceedings of the International Workshop on Logic and Synthesis*, pages 1–7, 2014.

[79] A. Mishchenko and R. Brayton. Scalable logic synthesis using a simple circuit structure. In *Proceedings of the International Workshop on Logic and Synthesis*, pages 15–22, 2006.

[80] A. Mishchenko, R. Brayton, and S. Chatterjee. Boolean factoring and decomposition of logic networks. In *Proceedings of the International Conference on Computer-Aided Design*, pages 38–44. IEEE, 2008.

[81] A. Mishchenko, R. Brayton, S. Jang, and V. Kravets. Delay optimization using SOP balancing. In *Proceedings of the International Conference on Computer-Aided Design*, pages 375–382. IEEE, 2011.

[82] A. Mishchenko, R. Brayton, J. R. Jiang, and S. Jang. Scalable don't-care-based logic optimization and resynthesis. *ACM Transactions on Reconfigurable Technology and Systems*, 4(4):34, 2011.

[83] A. Mishchenko and R. K. Brayton. SAT-based complete don't-care computation for network optimization. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 412–417. IEEE Computer Society, 2005.

[84] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In *Proceedings of the Design Automation Conference*, pages 532–535. ACM, 2006.

[85] A. Mishchenko, S. Chatterjee, and R. K. Brayton. Improvements to technology mapping for LUT-based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):240–253, 2007.

[86] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton. Combinational and sequential mapping with priority cuts. In *Proceedings of the International Conference on Computer-Aided Design*, pages 354–361, 2007.

[87] A. Mishchenko, B. Steinbach, and M. Perkowski. An algorithm for bi-decomposition of logic functions. In *Proceedings of the Design Automation Conference*, pages 103–108. ACM, 2001.

[88] A. Mishchenko, X. Wang, and T. Kam. A new enhanced constructive decomposition and mapping algorithm. In *Proceedings of the Design Automation Conference*, pages 143–148. ACM, 2003.

[89] A. Mishchenko, J. S. Zhang, S. Sinha, J. R. Burch, R. Brayton, and M. Chrzanowska-Jeske. Using simulation and satisfiability to compute flexibilities in boolean networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(5):743–755, 2006.

[90] N. Modi and J. Cortadella. Boolean decomposition using two-literal divisors. In *Proceedings of the International Conference on VLSI Design*, pages 765–768. IEEE, 2004.

[91] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[92] G. E. Moore. Progress in digital integrated electronics. In *International Electron Devices Meeting*, pages 11–13, 1975.

[93] S. B. Nasir, S. Gangopadhyay, and A. Raychowdhury. All-digital low-dropout regulator with adaptive control and reduced dynamic stability for digital load circuits. *IEEE Transactions on Power Electronics*, 31(12):8293–8302, 2016.

[94] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, New York, NY, USA, 2010.

[95] P. Pan and C. Lin. A new retiming-based technology mapping algorithm for LUT-based FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 35–42. ACM, 1998.

[96] M. D. Pant, P. Pant, and D. S. Wills. On-chip decoupling capacitor optimization using architectural level prediction. *IEEE Transactions on VLSI Systems*, 10(3):319–326, 2002.

[97] S. Pant and E. Chiprout. Power grid physics and implications for CAD. In *Proceedings of the Design Automation Conference*, pages 199–204. ACM, 2006.

[98] T. Pialis and K. Phang. Analysis of timing jitter in ring oscillators due to power supply noise. In *Proceedings of the International Symposium on Circuits and Systems*, pages 685–688. IEEE, 2003.

[99] M. Popovich, A. V. Mezhiba, and E. G. Friedman. *Power Distribution Networks with On-Chip Decoupling Capacitors*. Springer, New York, NY, USA, 1st edition, 2008.

[100] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News*, 42(3):13–24, 2014.

[101] V. J. Reddi, M. S. Gupta, G. Holloway, G.-Y. Wei, M. D. Smith, and D. Brooks. Voltage emergency prediction: Using signatures to reduce operating margins. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 18–29, 2009.

[102] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-Y. Wei, and D. Brooks. Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling. In *Proceedings of the International Symposium on Microarchitecture*, pages 77–88, 2010.

[103] T. Sakurai. A JSSC classic paper: the simple model of CMOS drain current. *IEEE Solid State Circuits Society Newsletter*, 9(4):4–5, 2004.

[104] H. Savoj and R. Brayton. The use of observability and external don't cares for the simplification of multi-level networks. In *Proceedings of the Design Automation Conference*, pages 297–301. IEEE, 1990.

[105] H. Sawada, T. Suyama, and A. Nagoya. Logic synthesis for look-up table based FPGAs using functional decomposition and support minimization. In *Proceedings of the International Conference on Computer-Aided Design*, pages 353–358. IEEE, 1995.

[106] C. Scholl. Multi-output functional decomposition with exploitation of don't cares. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 743–748. IEEE, 1998.

[107] K.-D. Schubert, W. Roesner, J. M. Ludden, J. Jackson, J. Buchert, V. Paruthi, M. Behm, A. Ziv, J. Schumann, C. Meissner, et al. Functional verification of the IBM POWER7 microprocessor and POWER7 multiprocessor systems. *IBM Journal of Research and Development*, 55(3):10–1, 2011.

[108] W. Shockley. The theory of p-n junctions in semiconductors and p-n junction transistors. *Bell System Technical Journal*, 28(3):435–489, 1949.

[109] F. Somenzi. CUDD: CU decision diagram package release 3.0.0. *University of Colorado at Boulder*, 2015. Available at `http://vlsi.colorado.edu/~fabio/CUDD/`.

[110] T. Stanion and C. Sechen. A method for finding good ashenhurst decompositions and its application to FPGA synthesis. In *Proceedings of the Design Automation Conference*, pages 60–64. ACM, 1995.

[111] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel. Capi: A coherent accelerator processor interface. *IBM Journal of Research and Development*, 59(1):7–1, 2015.

[112] S. M. Sze and K. K. Ng. *Physics of semiconductor devices*. John Wiley & Sons, Hoboken, NJ, USA, 3rd edition, 2006.

[113] J. Tschanz, N. S. Kim, S. Dighe, J. Howard, G. Ruhl, S. Vangal, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, et al. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *Proceedings of the IEEE International Conference on Solid-State Circuits*, pages 292–604, 2007.

[114] N. Vemuri, P. Kalla, and R. Tessier. BDD-based logic synthesis for LUT-based FPGAs. *ACM Transactions on Design Automation of Electronic Systems*, 7(4):501–525, 2002. Available at `http://www.ecs.umass.edu/ece/tessier/rcg/bds-pga-2.0/`.

[115] M. M. Waldrop. More than Moore. *Nature*, 530(7589):144–148, 2016.

[116] L. Wang and A. Almaini. Multilevel logic simplification based on a containment recursive paradigm. *IEE Proceedings Computers and Digital Techniques*, 150(4):218–226, 2003.

[117] K. Wilcox, R. Cole, H. R. Fair III, K. Gillespie, A. Grenat, C. Henrion, R. Jotwani, S. Kosonocky, B. Munger, S. Naffziger, et al. Steamroller module and adaptive clocking system in 28 nm CMOS. *IEEE Journal of Solid-State Circuits*, 50(1):24–34, 2015.

[118] S. Wolf, D. Awschalom, R. Buhrman, J. Daughton, S. Von Molnar, M. Roukes, A. Y. Chtchelkanova, and D. Treger. Spintronics: a spin-based electronics vision for the future. *Science*, 294(5546):1488–1495, 2001.

[119] K. L. Wong, T. Rahal-Arabi, M. Ma, and G. Taylor. Enhancing microprocessor immunity to power supply noise with clock-data compensation. *IEEE Journal of Solid-State Circuits*, 41(4):749–758, 2006.

[120] Z. Zeng, X. Ye, Z. Feng, and P. Li. Tradeoff analysis and optimization of power delivery networks with on-chip voltage regulation. In *Proceedings of the Design Automation Conference*, pages 831–836. ACM, 2010.