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

HENRIQUE PLÁCIDO

Tackling the Drawbacks of a Lagrangian Relaxation Based Discrete Gate Sizing Algorithm

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Microeletronics

Advisor: Prof. Dr. Ricardo Augusto da Luz Reis

Porto Alegre
November 2018

CIP — CATALOGING-IN-PUBLICATION

Plácido, Henrique

Tackling the Drawbacks of a Lagrangian Relaxation Based Discrete Gate Sizing Algorithm / Henrique Plácido. – Porto Alegre: PGMICRO da UFRGS, 2018.

100 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2018. Advisor: Ricardo Augusto da Luz Reis.

1. Leakage Power Minimization. 2. Timing Constraints. 3. Lagrangian Relaxation. 4. Cell Selection. 5. Physical Design. 6. EDA. 7. Microelectronics. I. Reis, Ricardo Augusto da Luz. II. Título.

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

ACKNOWLEDGMENTS

To my family, for the love and support through all these years.

To my advisor, Ricardo Reis, for accepting me as his student.

To Guilherme Flach, for sharing with me his knowledge and for all the meetings we had during the development of this work.

ABSTRACT

The shrink of the devices sizes allows the number of transistors in the integrated circuits to grow, leading to an increase in the leakage power. The discrete gate sizing technique consists in assigning each gate of the circuit to a cell option among the implementation versions available in the cell library. It is a powerful method used in the design flow to carry out optimizations, e.g., timing violations fixing and power and/or area minimization. The Lagrangian relaxation based gate sizer proposed in [Flach et al. 2013] has the best leakage power results published so far for the 2012 ISPD Gate Sizing Contest benchmarks. However, its Lagrangian relaxation phase has some drawbacks. It requires many iterations to converge to a good solution in terms of leakage power. Also, during the initial iterations, the leakage power blows up, so a parcel of the iterations is used to reduce this peak of leakage power. Yet, the Lagrangian relaxation subproblem solver does not rely on any technique to perform cell option candidate filtering, so it can be very timing consuming. Therefore, in this work, the discrete gate sizing flow proposed in [Flach et al. 2013] is extended to tackle the drawbacks aforementioned. It is proposed some enhancements to the Lagrange multiplier update formula that enable the Lagrangian relaxation core to converge faster. It is also used a scaling factor to properly scale timing cost and leakage power when evaluating a cell candidate in the Lagrangian relaxation subproblem solver. So, the scaling factor, alongside the new Lagrange multipliers update method, controls the leakage power blow up during the initial Lagrangian relaxation iterations. Moreover, it is applied a cell option candidate filtering strategy to reduce the runtime of each Lagrangian relaxation iteration. Finally, the post-processing timing recovery and power recovery phases of the original work are improved to reduce the overall flow runtime. The new approach achieved leakage power results similar to the baseline work, taking $4.28\times$ fewer iterations and $9.11\times$ fewer cell option candidates evaluation, on average, in the Lagrangian relaxation phase. Also, the leakage power blow up during the initial iterations of the Lagrangian relaxation was reduced from $9.55\times$ the final value, on average, to $2.74\times$ the final value, on average. Finally, compared to [Sharma et al. 2017], which is the fastest gate sizing algorithm published so far, the new approach produced, without using the post-processing power recovery phase, similar leakage power results in general, performing slightly better for the largest benchmark.

Keywords: Leakage Power Minimization. Timing Constraints. Lagrangian Relaxation.

Cell Selection. Physical Design. EDA. Microelectronics.

Tratando as Desvantagens de um Algoritmo de Dimensionamento Discreto Baseado em Relaxação Lagrangiana

RESUMO

A redução das dimensões dos dispositivos permite que o número de transistores nos circuitos integrados aumente, levando ao aumento da potência estática do circuito. A técnica de dimensionamento discreto de portas lógicas consiste em atribuir a cada porta lógica do circuito uma célula dentre todas as opções de implementação disponíveis na biblioteca de células. É uma poderosa técnica empregada no fluxo de síntese de circuitos integrados para realizar otimizações, como, por exemplo, remoção de violações de *timing* e minimização de potência e/ou área do circuito. O algoritmo de dimensionamento discreto de portas lógicas baseado em relaxação Lagrangiana proposto em [Flach et al. 2013] apresenta os melhores resultados em termos de potência estática publicados até então para os *benchmarks* da competição de dimensionamento discreto de portas lógicas do ISPD que ocorreu em 2012 [Ozdal, Burns and Hu 2012]. Contudo, a fase de relaxação Lagrangiana desse algoritmo possui algumas desvantagens. São necessárias muitas iterações para o algoritmo convergir para uma boa solução em termos de potência estática. Também, durante as iterações iniciais, a potência estática aumenta consideravelmente, assim, uma parcela das iterações é utilizada para reduzir o pico de potência estática. Ainda, o resolvedor do subproblema Lagrangiano não utiliza nenhuma técnica de filtragem de células candidatas, então, o algoritmo pode ser muito lento. Então, nesse trabalho, o fluxo de dimensionamento discreto de portas lógicas proposto em [Flach et al. 2013] é estendido para tratar as desvantagens citadas. São propostas algumas melhorias para a fórmula de atualização dos multiplicadores de Lagrange que permitem a fase de relaxação Lagrangiana convergir mais rapidamente. Também é utilizado um fator de escala para balancear adequadamente o custo de *timing* e a potência estática quando uma célula candidata é avaliada pelo resolvedor do subproblema Lagrangiano. Assim, o fator de escala, juntamente com o novo método de atualização dos multiplicadores de Lagrange, controla a explosão de potência estática durante as iterações inicias da fase de relaxação Lagrangiana. Ainda, é utilizada uma estratégia de filtragem de células candidatas para reduzir o tempo de execução das iterações do algoritmo de relaxação Lagrangiana. Finalmente, as etapas de pós-processamento *timing recovery* e *power recovery* foram modificadas para reduzir o tempo de execução do fluxo. A nova abordagem atingiu resultados em termos

de potência estática similares ao algoritmo original, tendo 4,28 vezes menos iterações, em média, e 9,11 vezes menos testes de células candidatas, em média, na fase de relaxação Lagrangiana. Também, o grande aumento de potência estática durante as iterações iniciais da relaxação Lagrangiana foi reduzido de 9,55 vezes a potência final obtida, em média, para 2,74 vezes a potência final obtida, em média. Finalmente, comparado ao algoritmo de dimensionamento discreto de células proposto em [Sharma et al. 2017], que é o mais rápido publicado até então, a ferramenta desenvolvida nesse trabalho produziu, mesmo não utilizando a fase de pós processamento *power recovery*, resultados muito próximos em termos de potência estática, tendo resultados levemente melhores para o maior *benchmark*.

Palavras-chave: Minimização de potência Estática, Restrições Temporais, Relaxação Lagrangiana, Seleção de Células, Síntese Física, EDA, Microeletrônica.

LIST OF ABBREVIATIONS AND ACRONYMS

ALU	Arithmetic and Logic Unit
CAD	Computer Aided Design
CISC	Complex Instruction Set Machine
CPS	Critical Path Sizing
CTS	Clock Tree Synthesis
DRC	Design Rule Check
EDA	Electronic Design Automation
ERC	Electrical Rule Check
Fast-GTR	Fast Greedy Timing Recovery
GTR	Global Timing Recovery
GWTW	Go-With-The-Winners
HDL	Hardware Description Language
IC	Integrated Circuit
ISPD	International Symposium on Physical Design
KKT	Karush-Kuhn Tucker
LDP	Lagrangian Dual Problem
LM	Lagrange Multiplier
LM	Linear Programming
LR	Lagrangian Relaxation
LRS	Lagrangian Relaxation Subproblem
LVS	Layout Versus Schematic
MGS	Multi-Gate Sizing
Min-Clock LR	Minimum Clock Period Lagrangian Relaxation
NLDM	Non-Linear Delay Model

PP	Primal Problem
PR	Power Recovery
PR-LR	Power Reduction Lagrangian Relaxation
PRFT	Power Reduction With Feasible Timing
PVT	Process, Voltage and Temperature
RGS	Rapid Gate Sizing
RISC	Reduced Instruction Set Machine
RTL	Register Transfer Level
SA	Simulated Annealing
SPEF	Standard Parasitic Extraction Format
STA	Static Timing Analysis
TNS	Total Negative Slack
TR	Timing Recovery
TR-LR	Timing Recovery Lagrangian Relaxation
WNS	Worst Negative Slack

LIST OF FIGURES

CONTENTS

1 INTRODUCTION

In 1958, the first integrated circuit (IC) was designed at Texas Instruments by Jack Kilby. This technology allowed the integration of the components of an electronic circuit in the same substrate of semiconductor material [Rabaey, Chandrakasan and Nikolic 2003]. Initially, the integrated circuits were manually designed. However, with the increasing number of components, this method has become prohibitive. For instance, at the end of the 1980s, there were circuits with more than one million of transistors, such as the Intel 486 DX [Intel 2008]. Besides, time-to-market, which is the time from conception of a product to its sale, is a crucial factor, since the company must launch the product into the market in the smallest time to ensure its success. Therefore, computer aided design (CAD) tools, also known as electronic design automation (EDA) tools, were developed to automate the design of integrated circuits [Rabaey, Chandrakasan and Nikolic 2003].

The shrinking of planar MOSFETs during the last four decades enabled a continuous increase in the transistor density in the integrated circuits. However, the continuity of this tendency is very challenging for the reason that the power consumption increases as a consequence of the increasing number of transistors in ICs.

The power consumption of ICs falls into two main components: dynamic and static (leakage). The dynamic power is related to the swithing activity of the transistors, whereas the leakage power is the power consumed due to undesirable leakage currents. According to [Dadoria, Khare and Singh 2015], the leakage power corresponds to approximately 45% of the total power consumption in ICs designed using a 90nm technology.

Since many chips are embedded in mobile systems, e.g, smartphones, tablets, laptops, and so forth, the reduction of power consumption, specially the undesired leakage power, becomes an unavoidable concern because it impacts the battery's life. Moreover, it is also important to design systems with low power consumption due to energy costs.

If compared to the CMOS planar technology, the FinFET dramatically improved the leakage power in modern designs [Bhattacharya and Jha 2014], however it has not eliminated this power component. Therefore, optimization techniques addressing leakage power minimization play a major role in modern designs.

In standard cell based designs, the discrete gate sizing may be employed in the design flow to perform leakage power optimization. Due to the relevance of the discrete gate sizing, the International Symposium on Physical Design (ISPD) contests 2012 [Ozdal et al. 2012] and 2013 [Ozdal et al. 2013] addressed it. The objective in both contests was

to minimize the leakage power of the circuit while meeting timing constraints. Since then, several papers addressing this problem were published [Hu et al. 2012], [Li et al. 2012], [Livramento et al. 2013], [Flach et al. 2013], [Flach et al. 2013], [Sharma et al. 2015], [Yella and Sechen 2017], [Sharma et al. 2017].

State-of-the-art works addressing the discrete gate sizing problem rely on Lagrangian relaxation (LR) due to its effectiveness to produce good solutions in terms of leakage power. However, LR based sizers can take many LR iterations to produce a good solution in terms of leakage power and timing. For instance, although the discrete gate sizing flow proposed in [Flach et al. 2013] has the best results in terms of leakage power published so far for the ISPD 2012 Contest on Discrete Gate Sizing benchmarks [Ozdal et al. 2012], its LR core requires 100 iterations to converge to a good solution. [Sharma et al. 2017] addressed this issue by proposing a lagrange multiplier update strategy that accelerates the convergence of the algorithm. Besides the many iterations drawback, the evaluation of all cell options to compute the lowest LR local cost can be very slow, specially if a complex timing model is used. Furthermore, in the initial LR iterations of [Flach et al. 2013], the leakage power tends to grow significantly, e.g 15x the final value [Flach et al. 2013], so several iterations are required to reduce this power increase. Yet, its LRS solver does not rely on any strategy for cell option candidate filtering. [Reimann, Sze and Reis 2016] proposed a cell ranking method to filter cell candidates, while [Sharma et al. 2017] applied a simple filter only in their multi-gate-sizing step.

Considering that a design can have a large number of gates, a coarse-grained LR based sizer can be very timing consuming if the drawbacks aforementioned are not properly handled. Thus, the main objective of this work is to propose a set of strategies to tackle the drawbacks related to the LR core of the flow presented in [Flach et al. 2013].

1.1 Standard Cell Based Design Flow

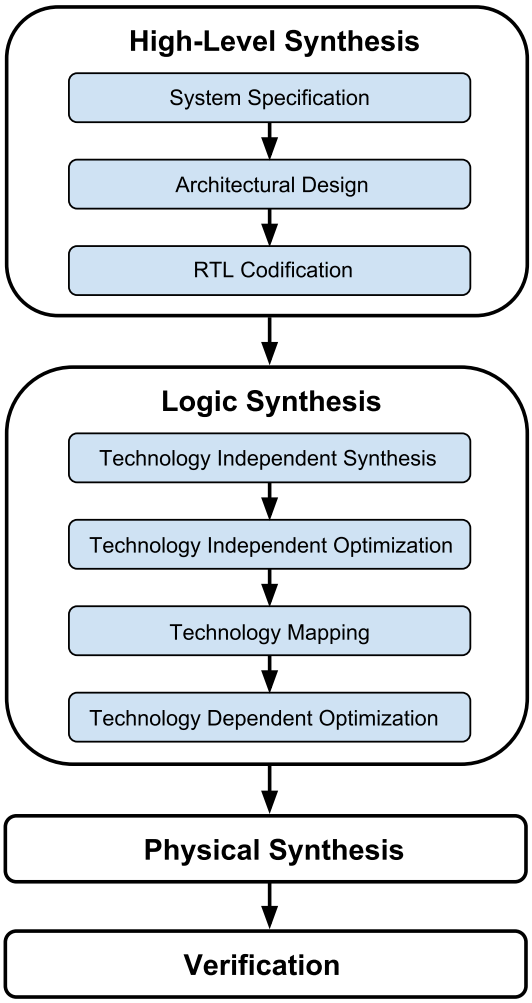
The design flow of an integrated circuit is a set of steps carried out to implement an integrated circuit from its specification. According to literature - [Sherwani 1999], [Kahng et al. 2011], [Gerez 1999], [Wang, Chang and Cheng 2009], [Lavagno, Scheffer and Martin 2006] -, a typical design flow is composed by high level synthesis, logic synthesis, physical synthesis and verification. The overall flow is depicted in Figure 1.1.

1.1.1 High Level Synthesis

The first step is the system specification, which is a high level representation of the system to be designed. In this step, the objectives and requeriments are specified, e.g. performance, functionalities and physical dimensions. Besides, it is also defined the fabrication technology.

After the system is specified, it is proposed a basic system architecture to satisfy the requeriments specified previously. For instance, if a processor is being designed, the engineers decide if it will be RISC (Reduced Instruction Set Computer) or CISC (Complex Instruction Set Computer), the number os ALUs (Arithmetic and Logic Unit) and floating point units, structure and number of pipelines, amount of cache memory, among others. The result of this step is a micro-architectural specification, which is a textual description of the decisions taken.

Figure 1.1: Standard Cell Based Design Flow.



Source: from author (2018)

In the last step of the high level synthesis, it is created a description of the circuit in the register transfer level (RTL). The RTL describes the system in terms of registers, flow of signals between them and the logic operations performed in these signals [Churiwala and Garg 2011]. To create the description, hardware description languages (HDLs), such as VHDL and verilog, are employed.

1.1.2 Logic Synthesis

In the logic synthesis step, the RTL description is translated into a gate level netlist mapped to the specified technology library. Initially, the logic synthesis is technology independent. That is, the initial gate-level netlist is composed of generic gates, such as OR, AND and NOT that are not related to any specific technology library [Wang, Chang and Cheng 2009]. Therefore, at this point, several technology-independent optimizations can be performed before mapping the netlist to the target library.

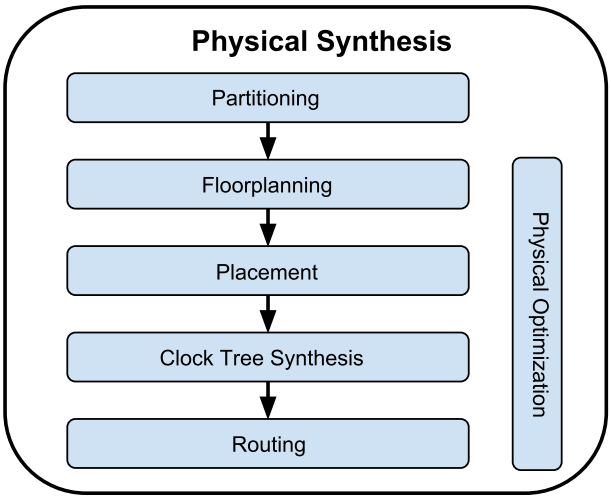
After the mapping was carried out, the resulting netlist can be optimized. As described in [Flach 2015], logic minimization, gate sizing, logic restructuring, retiming and buffering can be applied to optimize the mapped netlist.

1.1.3 Physical Synthesis

In the physical synthesis, it is generated the layout of the integrated circuit. Since this process is complex, the physical synthesis falls into several steps: partitioning, floorplanning, placement, clock tree synthesis (CTS) and routing [Kahng et al. 2011]. Figure 1.2 illustrates the physical synthesis flow.

An integrated circuit may be composed of millions of logic gates. Therefore, due to limitations of processing power and memory, it could not be possible to generate the layout considering the whole set of logic gates during the physical synthesis. Thus, in the partitioning step, the circuit is partitioned into several sub-circuits (blocks), so that the problem is divided into smaller and independent problems. An example of circuit partitioning is presented in Figure 1.3, which shows a circuit that was partitioned into three partitions. According to [Kahng et al. 2011], a common method employed to perform circuit partitioning is to create sub-circuits such that the number interconnections between these circuits is minimized. An example of algorithm that is based on this ap-

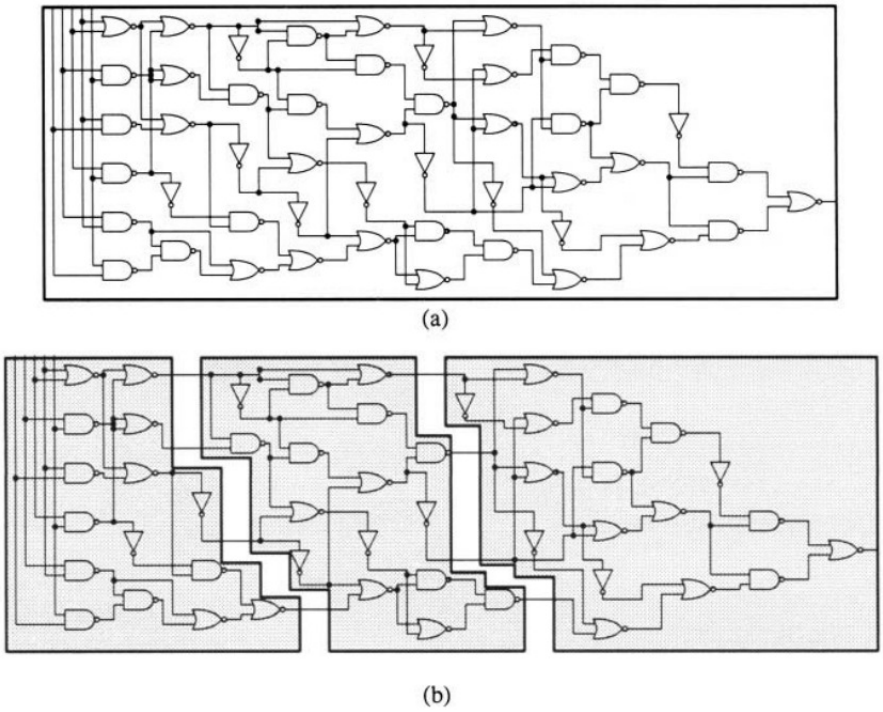
Figure 1.2: Physical Synthesis Flow.



Source: from author (2018)

proach is [Kernighan and Lin 1970], which partitions the netlist into two sub-circuits.

Figure 1.3: Partitioning example.



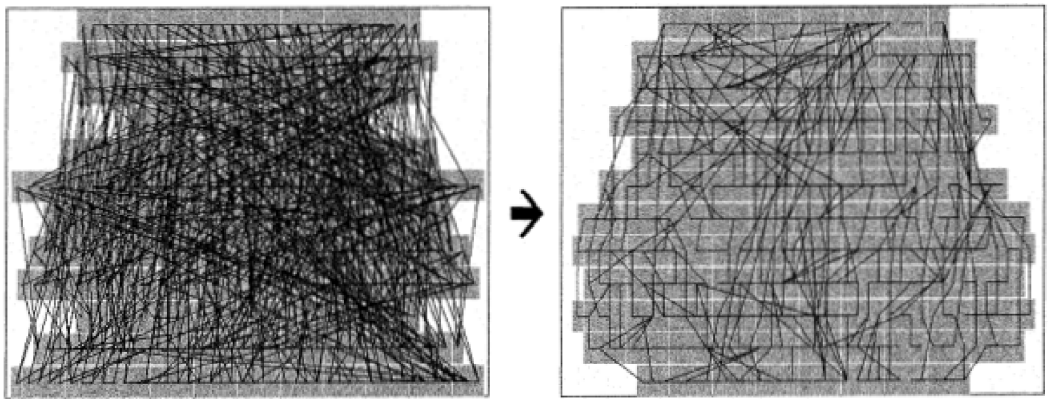
Source: [Sherwani 1999]

After the partitioning, the floorplanning step takes place. The floorplanning performs the estimation of the chip's dimensions, I/O pads and logic blocks placement and the power planning. The power planning distributes the vdd and gnd signals over the chip.

Afterwards performing the partitioning and the floorplanning, the placement is executed, which is responsible for determining the location of each logic cell. This is a crucial step, since a low quality placement can make the circuit unroutable or gener-

ate a solution that does not satisfy the performance requeriments. Figure 1.4 shows an example of a random placement and a solution obtained by performing the simulated annealing (SA). In the prior, it is possible to notice that there is a huge number of congested interconnections, while in the second the congestion was dramatically reduced.

Figure 1.4: Random placement versus placement obtained with the *simulated annealing* algorithm.



Source: adapted from [Hentschke, Johann and Reis 2005]

After the sequential elements, e.g. flip-flops, are placed, the next step is the clock tree synthesis, in which it is generated a clock tree of buffers. The clock tree distributes the clock signal to all synchronous elements, so that the timing requeriments are met.

Subsequently to the clock tree generation, the routing step determines the routes of the interconnections. In order to ensure the chip's manufacturability, the paths generated must satisfy the design rules. The main objective of this step is to route 100% of the wires, since, otherwise, the circuit will not work properly. The delay of the interconnections can degrade the performance of the design, therefore, another important objective is to minimize the total wirelength, although the placement plays the major role in this objective. The routing step usually falls into two substeps: global routing and detailed routing. The global routing performs a planning of the routing, that is, it specifies the regions of the chip where each wire will pass through. The detailed routing takes the planning carried out by the global routing and determines the exact location of each wire.

Once the layout is generated, a complete verification is performed to ensure that the solution is valid. Thus, during the physical verification, one of the processes that is executed is the design rule check (DRC). The DRC verifies if the layout respects the design rules of the manufacturing process technology, e.g. minimum distance between wires. Another process executed during the physical verification is the layout versus schematic (LVS), which generates a netlist from the layout and compares it with the original netlist

to verify if they are logically equivalent. A parasitic extraction process is also employed to obtain the electrical parameters of the layout, such as resistences, capacitances and indutances. Yet, the antenna rule checking process is performed to prevent antenna effects, which could cause damages to the transistor gates in the manufacturing process. Finally, the electrical rule check (ERC) is carried out to verify the power and ground connections and if the slew times, fanouts and capacitive loads properly meet the constraints [Kahng et al. 2011].

During the design flow, the discrete gate sizing is widely used to perform optimizations [Lee and Gupta]. Although this technique, which is the topic of this work, is not explicitly shown in the design flow discussed above, it can be used in different stages of the design flow, such as logic synthesis, technology mapping, critical path optimization and power/area optimization [Reimann, Sze and Reis 2016]. For instance, after the clock tree synthesis, it can be used to fix setup and hold violations using the clock information obtained from the CTS. After routing, it can be used once more to fix setup and hold violations [Lee and Gupta]. Yet, it can be used incrementaly in a pre-optimized design to perform power and area optimization [Reimann, Sze and Reis 2016].

It is important to highlight that, currently, the logic synthesis and physical synthesis steps are not completely separated as shown in the flow above. In the old days, the delay of interconnections were negligible if compared to the gates delay. However, today, due to the reduction of the transistor dimensions, load capacitances and channel resistences were dramatically reduced. Thus, the gate delay reduced as well. On the other hand, the shrinking of the interconnections increases their resistance due to the reduction of the transversal section. Besides, the capacitance of wires keeps increasing due to the reduction of the distance between these wires. Therefore, the delay of the interconnections has became a limiting factor in the circuit performance [Rossum 2009]. For this reason, the logic synthesis relies on physical information and the physical synthesis started to call optimization methods that were previously used only during the logic synthesis [Flach 2015]. One example of this interation between logic synthesis and physical synthesis is the placement algorithm proposed in [Stenz et al. 2000]. The algorithm takes a design whose cells are previously placed and determines the propagation delay of the wires. The delay information is used to perform transformations in the original netlist, so that the paths delay are reduced. After the netlist is changed, the cells are placed again, since during the netlist transformation process several cells can be added and removed.

1.2 Objectives of this work

The objectives throughout this work are:

- Understand the discrete gate sizing problem.
- Implement a discrete gate sizing flow on Rsyn [Flach et al. 2017].
- Propose improvements to tackle some drawbacks related to the LR core of the implemented algorithm.

1.3 Organization of this Work

This work is organized as follows: Chapter 2 covers the basic concepts related to static timing analysis and power consumption in CMOS technology; Chapter 3 discusses the gate sizing problem, presenting an overview of continuous gate sizing and covering in detail the discrete gate sizing; Chapter 4 presents a literature review of discrete gate sizing algorithms, covering the early work and state-of-the-art; Chapter 5 dives into the details of the baseline algorithm proposed in [Flach et al. 2013]; Chapter 6 presents the improvements proposed for the baseline gate sizing algorithm and the results obtained. Finally, in Chapter 7, it is presented the conclusions and future works.

2 CONCEPTS

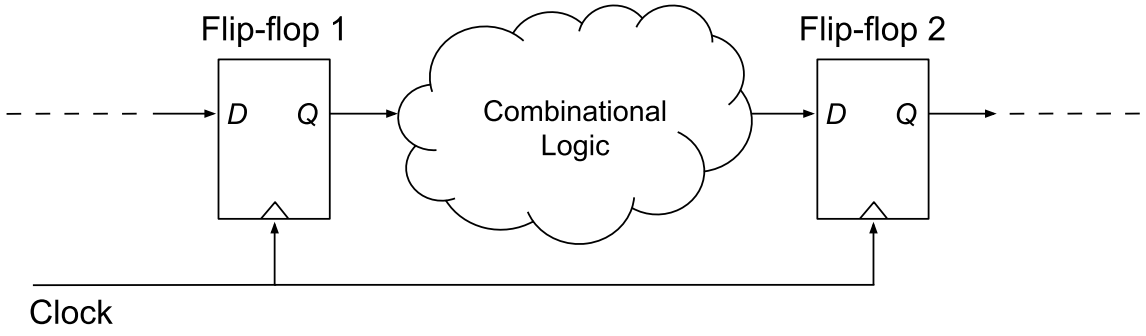
2.1 Static Timing Analysis

The Static Timing Analysis (STA) [Hitchcock, Smith and Cheng 1982, Güntzel 2000] is a technique used to evaluate the timing of a digital circuit. This method performs the timing analysis without relying on data inputs or simulations, which is in contrast with dynamic techniques, that apply simulation vectors to carry out the analysis [Lee and Gupta]. Considering a circuit with millions of gates, simulation techniques can be very slow [Bhasker and Chadha 2009], thus prohibitive, since the timing of the design must be verified several times during the design flow. On the other hand, the STA is a faster and simpler method, hence, it is able to cope with dense designs. The ideia of STA is to evaluate if the circuit will properly operate at the specified frequency by computing the best and worst case path delays [Sapatnekar 2004, Bhasker and Chadha 2009]. The next subsections dive into STA, presenting several concepts related to it.

2.1.1 Synchronous Digital Circuits

Basically, a synchronous digital integrated circuit is composed by logic gates and storage elements, i.e., flip-flops. Due to design requeriments, a circuit can contain one or more clock domains. A clock domain could be stated as a portion of the circuit in which all flip-flops are synchronized by the same clock signal. As depicted in Figure 2.1, a sub-circuit belonging to a same clock domain can be decomposed into a set of combinational blocks interconnected by flip-flops.

Figure 2.1: High level representation of a synchronous digital circuit.



Source: from author (2018)

Considering the illustration above, the signals must propagate through the combinational logic between two temporal barriers (flip-flops) within a time window. Thus, the worst path delay between each pair of flip-flops can not exceed:

$$t_{Ck \rightarrow Q,max} + t_{comb,max} \leq T - t_{setup} + \delta - 2t_{jitter} \tag{2.1}$$

where $t_{Ck \rightarrow Q,max}$ is the maximum propagation delay of the first flip-flop; $t_{comb,max}$ is the maximum propagation delay of the combinational block; setup time (t_{setup}) is the amount of time a signal must be stable at the input of the second flip-flip before the clock signal reaches this flip-flop; clock skew (δ) is the spatial variation in the arrival time of the clock signal at each flip-flop; $t_{jitter}$is the clock uncertainty window. If the condition above is not satisfied, the circuit has a setup or late violation.

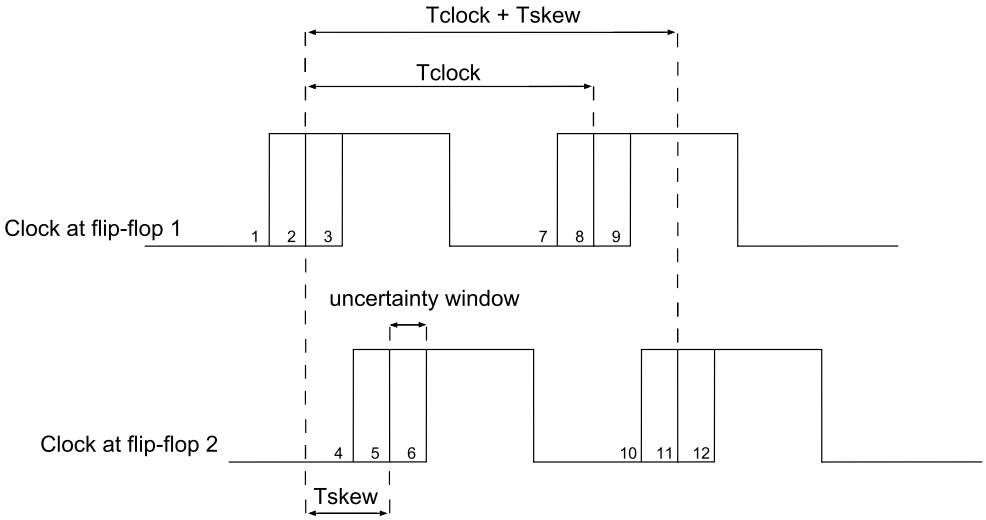
In order to avoid a hold or early violation, that is, a situation when signal at the input of a flip-flop is overwritten before it is stored in this flip-flop, the following condition must be met:

$$t_{Ck \rightarrow Q,min} + t_{comb,min} > \delta + t_{hold} + 2t_{jitter} \tag{2.2}$$

where $t_{Ck \rightarrow Q,min}$ is the minimum propagation delay of the first flip flop; $t_{comb,min}$ is the minimum propagation delay of the combinational block; hold time (t_{hold}) is the amount of time a signal must be stable at the input of the second flip-flip after the clock signal reaches this flip-flop. Figure 2.2 shows a temporal diagram from which the conditions established in Equations 2.1 and 2.2 were derived. It is possible to notice that the Equation 2.1 was obtained by analyzing the case in which the time window that the signals have to propagate through the circuit is minimal. It corresponds to the situation when the leading edge of the current clock happens late (edge 3) and the leading edge of the next clock happens early (edge 10). The Equation 2.2 is formulated by observing the case in which the leading edge of the current clock arrives early (edge 1) in the flip-flop 1 and arrives late (edge 6) in flip-flop 2. Hence, if the minimum path delay is small enough, a signal at the input of flip-flop 2 will be overwritten.

In order to verify the existence of a late violation, a STA tool computes the maximum path delay, whereas the existence of an early violation is verified by performing the computation of the minimum path delay. The computation of the minimum path delay is called early mode analysis while the computation of the maximum path delay is called late mode analysis. For each mode, different characterizations for cells are used. Each

Figure 2.2: Clock signal at two flip-flops considering skew and jitter.



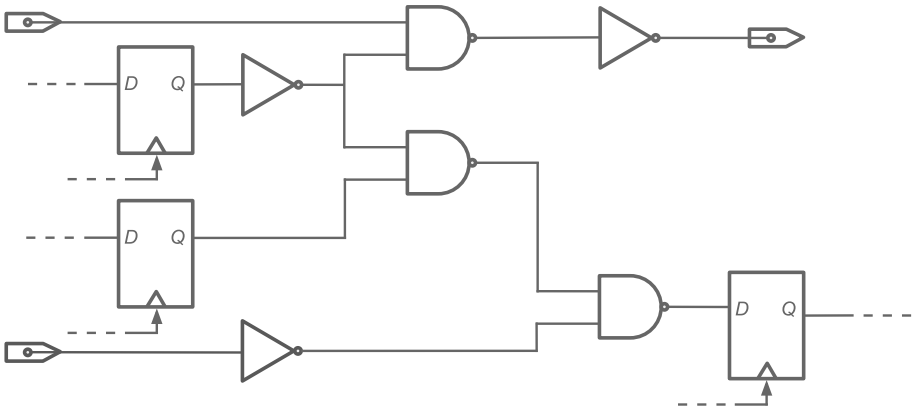
Source: adapted from [Rabaey, Chandrakasan and Nikolic 2003]

characterization is carried out at a specific process, voltage and temperature (PVT) corner. Hence, in the early mode, the cells work as fast as possible, while, for the late mode, the analysis is done such that the cells work as slow as possible. In the context of this work, only the late mode analysis is performed. Thus, from this point forward, the early mode is not addressed in the discussions.

2.1.2 Modeling of Synchronous Digital Circuits

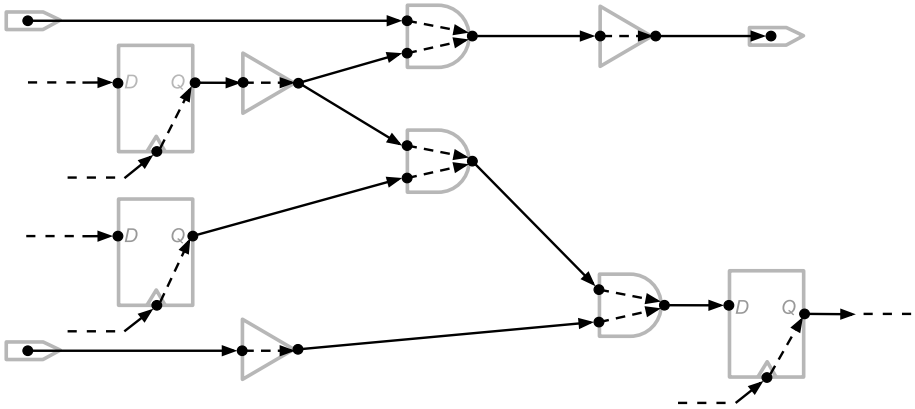
Figure 2.3 shows an example of a portion of a synchronous digital circuit containing combinational gates and flip-flops for storage. For the purpose of static timing analysis, a design is modeled using a directed timing graph in which the nodes represent the pins and the edges represent the timing arcs. A timing arc provides the timing information between two adjacent pins of the circuit [Lee and Gupta]. There are two types of timing arcs: cell timing arc, which connects two pins of a same cell, and net timing arc, which provides the connection between different elements, i.e, logic gates, flip-flops and i/o. Figure 2.4 shows the timing graph that was derived from the circuit depicted in Figure 2.3.

Figure 2.3: Example of a portion of a synchronous digital circuit.



Source: from author (2018)

Figure 2.4: Directed timing graph.

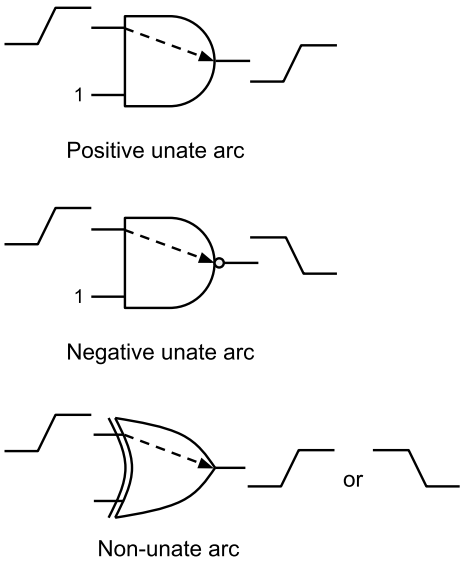


Source: from author (2018)

2.1.3 Timing Sense

Each timing arc has a timing sense, which indicates how the output changes w.r.t. the input signal transition. A timing arc is positive unate if a rise/fall transition at the input causes a rise/fall transition at the output. On the other hand, a timing arc is negative unate if a rise/fall transition at the input causes a fall/rise transition at the output. Finally, a timing arc is non-unate if the transition at the output do not depends only on the transition at the input, but also depends on the state of the other inputs. Figure 2.5 ilustrates these three cases.

Figure 2.5: Timing sense of arcs.

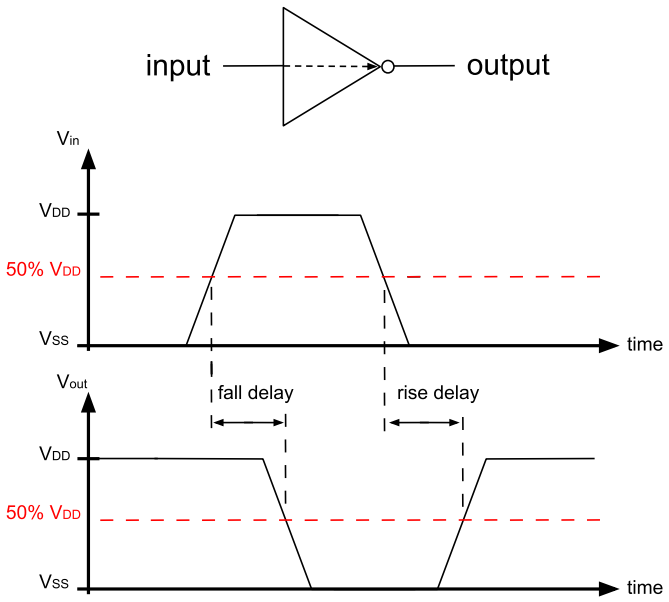


Source: from author (2018).

2.1.4 Propagation Delay of a Timing Arc

The propagation delay of a timing arc is defined as the amount of time it takes the output to reach 50% of Vdd after the input reached 50% of Vdd [Bhasker and Chadha 2009]. Figure 2.6 ilustrates the propagation delay of a negative unate timing arc.

Figure 2.6: Propagation delay of an inverter.

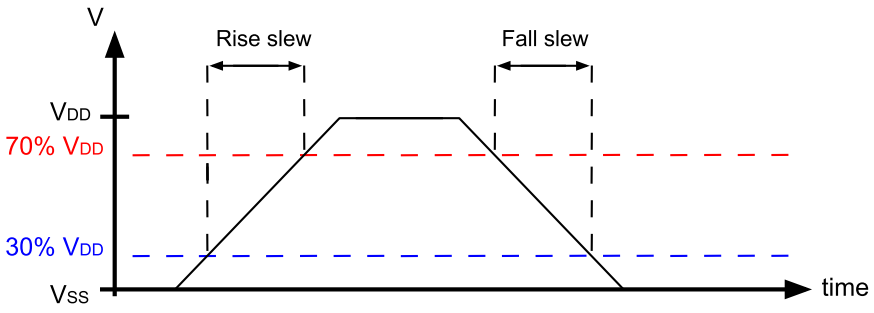


Source: from author (2018).

2.1.5 Transition Time (Slew)

The transition time is defined as the time it takes a signal to change its logic level. The rising slew is typically measured from the time the signal reaches 30% of Vdd to the time it reaches 70% of Vdd. In the same way, the falling slew is typically measured from the time the signal reaches 70% of Vdd to the time it reaches 30% of Vdd. Another typical threshold points are 10% to 90% and vice versa and 20% to 80% and vice versa. Figure 2.7 ilustrates the measure of a rise transition time using the 30% of Vdd and 70% of Vdd threshold points.

Figure 2.7: Rise and fall transition times.



Source: from author (2018).

2.1.6 Time Modeling

2.1.6.1 Cells

The cell libraries provide look-up table models that contain sampled values of propagation delay and output slew for each timing arc of a cell. The look-up table model is also referred as non-linear delay model (NLDM) [Bhasker and Chadha 2009]. Each table entry is indexed using the input slew and output load capacitance. If the input slew and/or output load do not match any of the table entries, the delay/output slew is obtained by linear interpolation.

- **cell_rise:** Specifies values of delay for a rise transition at the output.
- **cell_fall:** Specifies values of delay for a fall transition at the output.
- **rise_transition:** Specifies values of rise output slew.
- **fall_transition:** Specifies values of fall output slew.

It is presented below two look-up tables that were provided in the ISPD 2012 Contest on Discrete Gate Sizing [Ozdal et al. 2012]. The first one contains sampled values of delay for a rise transition at the output, whereas the second one contains sampled values of rise output slew.

Table 2.1: Look-up table which provides sampled values of delay for a rise transition.

Input slew (ps) \ Output load (ff)	**5.00**	**30.00**	**50.00**	**80.00**	**140.00**	**200.00**	**300.00**	**500.00**
0.00	11.72	18.22	22.67	27.61	34.82	40.44	48.21	61.24
1.00	16.93	23.43	28.60	34.95	44.15	51.18	60.65	75.83
2.00	22.13	28.63	33.83	41.24	52.23	60.57	71.67	89.00
4.00	32.55	39.05	44.25	52.05	66.05	76.76	90.85	112.34
8.00	53.38	59.88	65.08	72.88	88.48	103.09	122.40	151.42
16.00	95.05	101.55	106.75	114.55	130.15	145.75	171.59	213.31
32.00	178.38	184.88	190.08	197.88	213.48	229.08	255.08	307.08

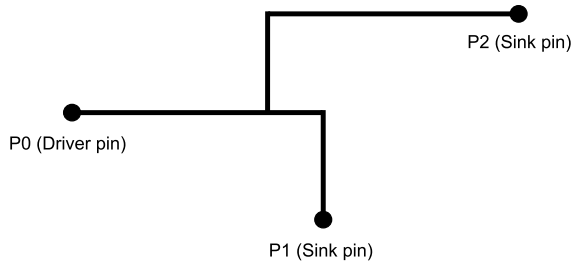
Table 2.2: Look-up table which provides sampled values of rise output slew.

Input slew (ps) \ Output load (ff)	**5.00**	**30.00**	**50.00**	**80.00**	**140.00**	**200.00**	**300.00**	**500.00**
0.00	12.50	13.72	16.38	20.10	25.17	28.65	32.95	39.83
1.00	18.75	19.20	21.15	25.28	32.15	37.06	42.98	51.31
2.00	25.00	25.09	26.41	29.91	37.97	44.09	51.66	61.86
4.00	37.50	37.50	37.82	40.10	47.66	55.77	66.14	80.57
8.00	62.50	62.50	62.50	62.90	67.51	74.76	88.78	110.23
16.00	112.50	112.50	112.50	112.50	113.06	116.67	126.92	154.48
32.00	212.50	212.50	212.50	212.50	212.50	212.50	215.15	232.09

2.1.6.2 Interconnections

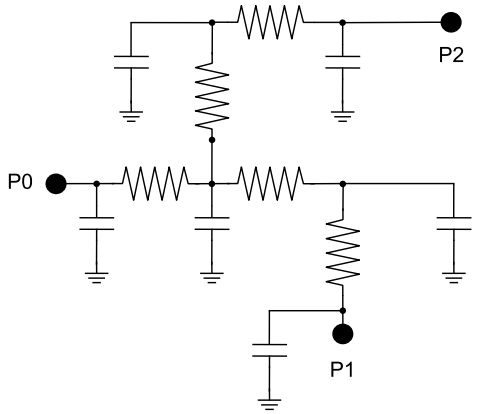
The delay of a wire is calculated based on its parasitic elements (resistences and capacitances). In this work, the interconnection parasitics are extracted from a standard parasitic extraction format (SPEF) file. In SPEF, three models of interconnections are supported: distributed net model, reduced net model and the lumped capacitance. In the distributed net model, each interconection segment has its own resistence and capacitance. In the reduced net model, the net is modeled using a resistence-capacitance-resistence structure connected to the driver pin and a resistence-capacitance structure connected to each sink pin. In the lumped capacitance model, the entire net is modeled just as a single capacitance which is summed up with the capacitances of the sink pins. Figure 2.8 shows an example of an interconnection, while Figures 2.9, 2.10 and 2.11 depict this interconnection net represented using the distributed net model, reduced net model and lumped capacitance model, respectively. In this work, only the lumped capacitance model is used, since the benchmarks of the ISPD 2012 Contest [Ozdal et al. 2012] used to perform the experiments adopt this model.

Figure 2.8: Example of an interconnection.



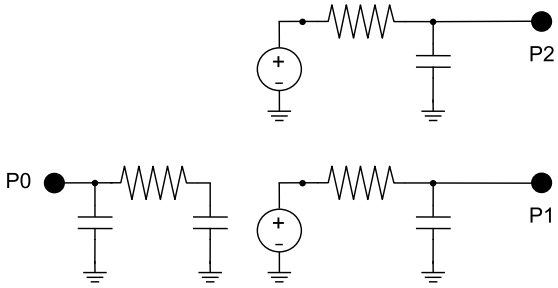
Source: adapted from [Bhasker and Chadha 2009]

Figure 2.9: Distributed net model.



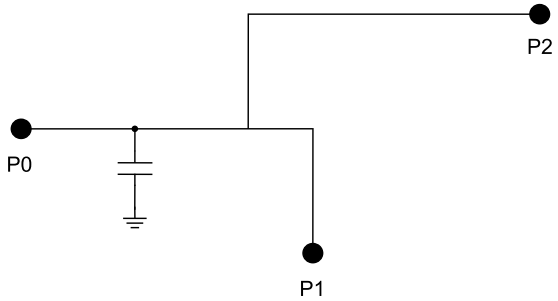
Source: adapted from [Bhasker and Chadha 2009]

Figure 2.10: Reduced net model.



Source: adapted from [Bhasker and Chadha 2009]

Figure 2.11: Lumped capacitance model.

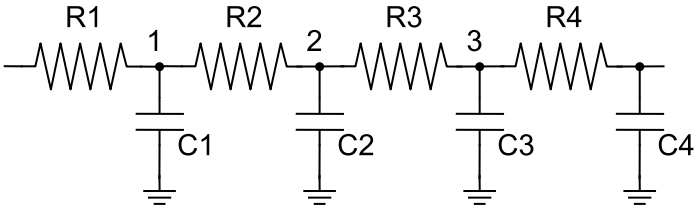


Source: adapted from [Bhasker and Chadha 2009]

For the lumped capacitance model, the arrival times at the driver pin of a net and at the sink pin of the same net are equal. Hence, it is not actually used to model the delay

of an interconnection. On the other hand, when the distributed net model is adopted, the delay of the interconnection is considered in the analysis. In order to compute the delay, the Elmore Delay [Elmore 1948] can be used.

Figure 2.12: Elmore example.



Source: adapted from [Bhasker and Chadha 2009]

For the circuit illustrated in Picture 2.12, the delay to the points 1, 2 and 3 are calculated as using Elmore delay as follows:

$$T_{d1} = R_1 \times (C_1 + C_2 + C_3 + C_4) \tag{2.3}$$

$$T_{d2} = R_1 \times (C_1 + C_2 + C_3 + C_4) + R_2 \times (C_2 + C_3 + C_4) \tag{2.4}$$

$$T_{d3} = R_1 \times (C_1 + C_2 + C_3 + C_4) + R_2 \times (C_2 + C_3 + C_4) + R_3 \times (C_3 + C_4) \tag{2.5}$$

The general form of the Elmore delay to the node i is given by Equation 2.6:

$$T_{di} = \sum_{k=1}^{N} (C_k \times R_{ik}) \tag{2.6}$$

where N is the number of nodes of the net and R_{ik} is the sum of all resistences shared among the paths from the source node to nodes k and i.

The slew at a node n of a net may be calculated as in [Flach et al. 2014]:

$$slew_n = \sqrt{slew_{parent}^2 + 1.93 \times (RC)^2} \tag{2.7}$$

where R is the resistence which connects n to its parent node and C is the downstream capacitance.

2.1.7 Timing Information

A STA tool can provide many information related to timing arcs and pins. It is presented below the typical information that is commonly used throughout this work.

- **Arrival time at pin** p (at_p)**:** It is the maximum time it takes a signal to arrive at the pin p.
- **Required time at pin** p (rt_p)**:** It is the maximum instant at which a signal should arrive at the pin p so that a timing violation does not occur.
- **Slack at pin** p $(slack_p)$**:** The slack is the difference between the required time and the arrival time at the pin p, i.e., $slack_p = rt_p - at_p$.
- **Slew at pin** p $(slew_p)$**:** Indicates the transition time at the pin p.
- **Delay of a timing arc** $(delay_{i \to j})$**:** Indicates the delay of a timing arc starting at node i and ending at node j.

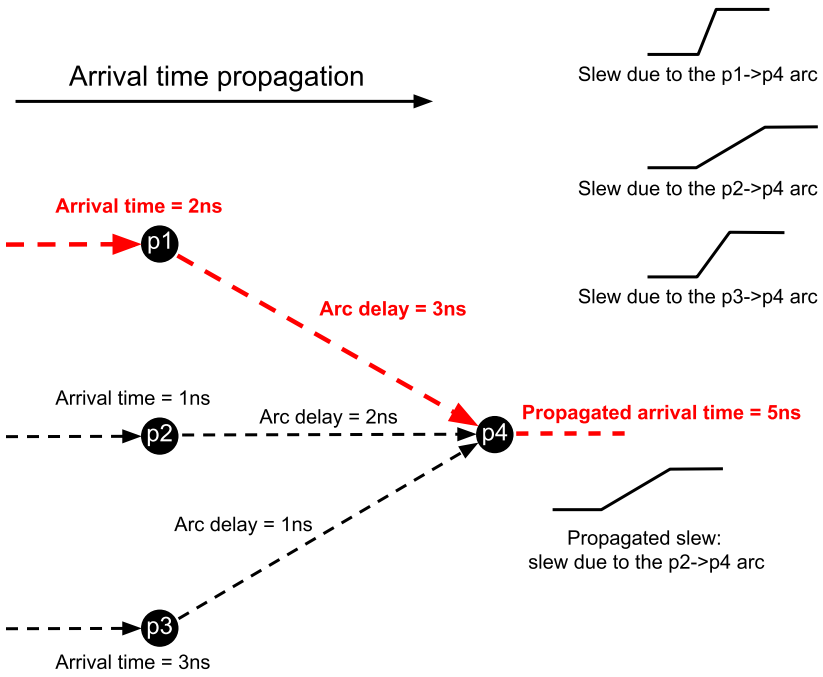
Considering the late mode analysis, a negative slack at the pin P indicates that a timing violation has occurred at this pin. On the other hand, a positive slack may be explored to perform optimizations.

2.1.8 Timing Propagation

This subsection discusses how the typical timing information presented in the former subsection is computed for all pins of the circuit. Since the early mode is out of the scope of this work, only the timing propagation in late mode is presented.

In order to compute the arrival time for each pin of the circuit, the STA tool traverses the timing graph in topological order. It starts from the primary inputs and, for each timing arc visited, the arrival time at its output is computed by summing the arrival time at its input with the timing arc propagation delay. If two or more timing arcs converge into a same output pin, e.g. a nand gate, only the worst arrival time and largest slew at the output are propagated towards the primary outputs. The example depicted in Figure 2.13 shows three timing arcs converging into a pin node. The arrival times at the p4 pin are 5ns, 3ns and 4ns due to the timing arcs p1->p4, p2->p4 and p3->p4, respectively. Hence, the largest arrival time, which is 5ns, is propagated forward. Also, the largest slew at the p4 pin is due to the p2->p4 timing arc, therefore it is the propagated transition time.

Figure 2.13: Arrival time and slew propagation.



Source: from author (2018).

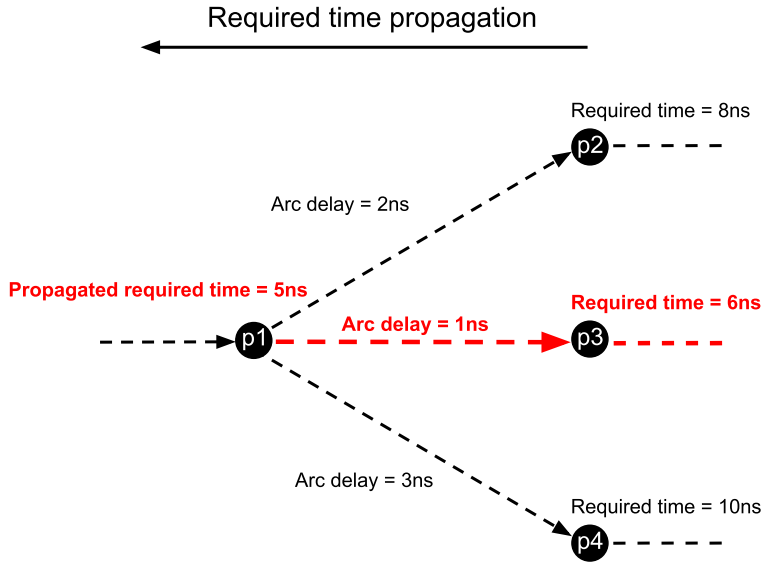
After the arrival times for all pins of the circuit are obtained, the timing graph is traversed in reverse topological order to compute the required time and slack for each node. The required time at each endpoint node is set as its target delay. The required time for a non-endpoint node v_i is computed as:

$$rt_{v_i} = min(rt_{v_j} - delay_{v_i \to v_j}), v_j \in fanout(v_i) \tag{2.8}$$

Figure 2.14 illustrates an example of required time propagation. The required times at the node p1 are 6ns, 5ns and 7ns due to the timing arcs p1->p4, p2->p4 and p3->p4, respectively. Therefore, the p1 node receives the minimum required time, which is 5ns.

Algorithm 2.1 summarizes the static timing analysis for the late mode. For the sake of simplicity, this algorithm presents the STA in a generic way, omitting the type of transition at each node, since it depends on the unateness of the timing arcs. The term $slew_{v_i \to v_j}$ refers to a transition at the node v_j due to the timing arc $v_i \to v_j$. The other terms are the presented in the former subsection.

Figure 2.14: Required time propagation.



Source: from author (2018).

Algorithm 2.1: Static Timing Analysis

set arrival time of startpoints
set required time of endpoints
for *each node* v_j *in topological order* **do**
    $at_{v_j} \leftarrow -\infty$
    $slew_{v_j} \leftarrow -\infty$
    for *each timing arc* $v_i \rightarrow v_j$ **do**
        $at_{v_j} \leftarrow max(at_{v_j}, at_{v_i} + delay_{v_i \rightarrow v_j})$
        $slew_{v_j} \leftarrow max(slew_{v_j}, slew_{v_i \rightarrow v_j})$
    end
end
for *each node* v_i *in reverse topological order* **do**
    $rt_{v_i} \leftarrow +\infty$
    for *each timing arc* $v_i \rightarrow v_j$ **do**
        $rt_{v_i} \leftarrow min(rt_{v_i}, rt_{v_i} - delay_{v_i \rightarrow v_j})$
    end
    $slack_{v_i} \leftarrow rt_{v_i} - at_{v_i}$
end

2.1.9 WNS and TNS

For the late mode, the worst negative slack (WNS) is the slack of the endpoint which has the largest timing violation. The total negative slack (TNS) is the summation of all negative slacks of the endpoints.

2.1.10 Criticality and Centrality

The criticality of a pin $\in [0, 1]$ is defined as the worst negative slack of this pin divided by the worst negative slack of the design. The centrality of a pin $\in [0, 1]$ is an estimation of how many and how critical are the endpoints affected by this pin [Flach et al. 2016].

2.2 Power Consumption in CMOS Technology

The power dissipation in CMOS circuits is divided into two componentes [Weste and Harris 2010]:

- Dynamic power
- Static power

The dynamic power is related to the charging and discharging of the load capacitance and the "short circuit" current that flows when both PMOS and NMOS networks are partially active. When the pullup stack of a gate is active, it charges the output load capacitance. Thus, a certain quantity of energy is drawn from the power supply. Part of this energy is stored in the output load capacitance, while the remaining part is dissipated in the PMOS transistores. When the output load is discharged, its energy is dissipated in the NMOS transistores. The total energy drawn from the power suply is given as follows:

$$E = \frac{C_L \times Vdd^2}{2} \tag{2.9}$$

where C_L is the output load capacitance and Vdd is the power supply voltage.

The power dissipated is computed based on the switching activity factor, which is a statistical measure of how many times the device switches per second. Therefore, the

power dissipated is calculated as:

$$P_{dyn} = C_L \times Vdd^2 \times f_{0\rightarrow1} \tag{2.10}$$

where $f_{0\rightarrow1}$ is the switching activity factor.

Since the transition time of the input signal is not zero, there is a short period of time when both NMOS and PMOS transistors are active simultaneously. Therefore, a direct current path between Vdd and GND exists during this period. The power consumption due the direct path current can be approximately calculated by assuming that the current spikes have triangle shapes and that the gate is symmetrical in its rising and falling output edges. Thus, the energy drawn from the power supply during the switching period is given by:

$$E = t_{sc} \times Vdd \times I_{peak} \tag{2.11}$$

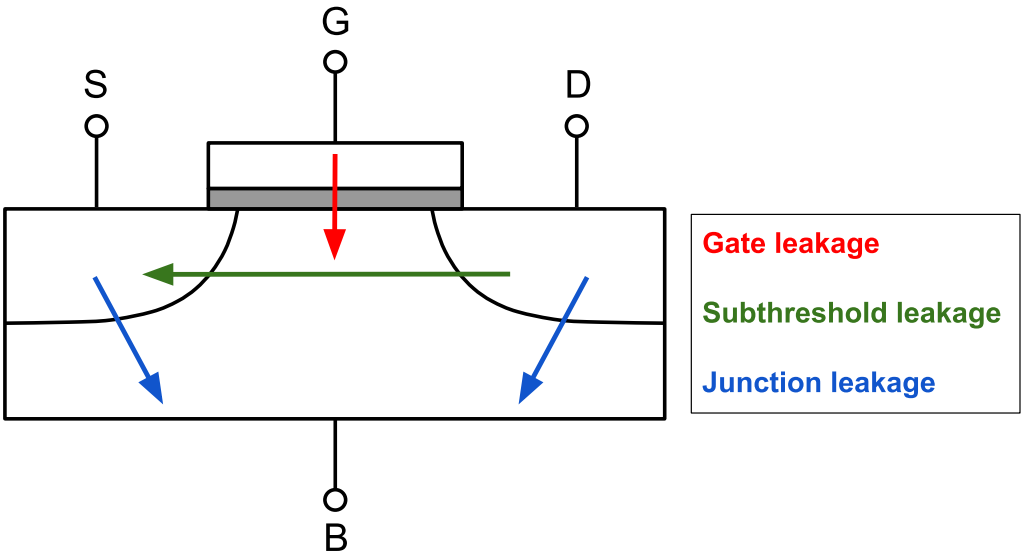
where t_{sc} is the period of time when both NMOS and PMOS are conduncting and I_{peak} is the maximum value of the direct path current during t_{sc}. Then, the average power consumption is

$$P = t_{sc} \times Vdd \times I_{peak} \times f \tag{2.12}$$

where f is the switching activity factor.

The static power is a consequence of the subthreshold, gate and reverse-biased diode juntions leakage currents [Weste and Harris 2010]. The subthreshold leakage current flows between drain-source when it is applied a gate-source voltage (V_{GS}) smaller than the threshold voltage (V_{th}). The smaller the V_{th}, the larger the leakage current when the transistor is "OFF", that is, when V_{GS} is 0V. Therefore, the smaller the V_{th}, the larger the static power consumption. The gate leakage current is caused by carriers that are tunneled through the gate when it is biased. This current depends on the thickness of the gate oxide and voltage across the gate [Weste and Harris 2010]. The last component of static power, junction leakage, is caused by junction current that flows when the drain or source diffusion is biased with a potential different from the substrate. Figure 2.15 illustrates the three leakage components.

Figure 2.15: Sources of leakage current.



Source: from author (2018)

In this work, only the static power is optimized. The leakage power information of each cell is provided by a liberty file. The liberty file is the industry standard format for representing timing and power information of the cells provided by a library [Prakash 2007]. For example, for the inverter cell in01s01 of the ISPD2012 benchmark suite [Ozdal et al. 2012], the static power is given by the following field:

cell_leakage_power : 1.00;

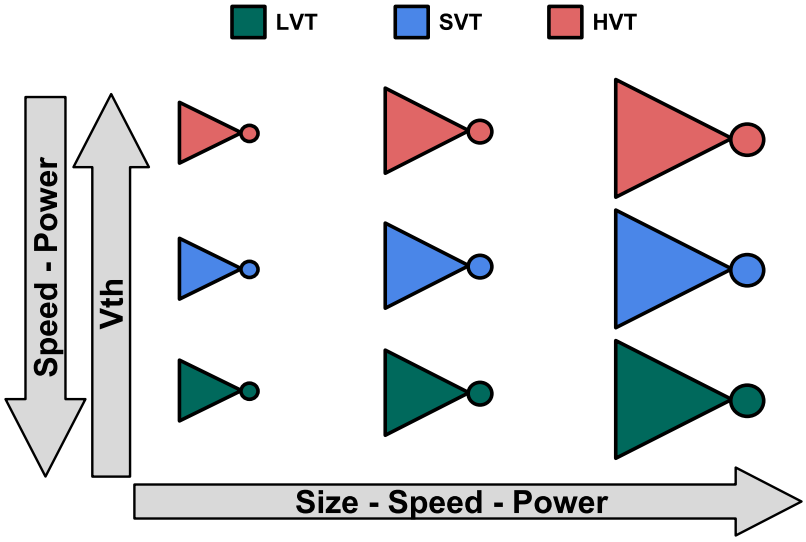
The leakage power unit is specified in the liberty file header as:

leakage_power_unit : 1uW;

3 GATE SIZING

In standard cell based designs, the gate-level netlist is mapped to a technology library, which contains different implementation options for each available logic function. Considering a particular logic function, the versions differ in the width of the gate's transistors, referred here as the size of the gate, and V_{th}. Hence, they have different timing and leakage power characteristics to tradeoff. As illustrated in Figure 3.1 for a not gate, the larger is the size, the faster it is, however, it has a greater power consumption. In the same way, the lower the V_{th}, the greater the speed and the power consumption. In this context, the discrete gate sizing consists in assigning each gate of the netlist to an implementation option available in the cell library. It is a powerful technique that can be used in many stages of the design flow to perform optimizations, such as timing violations fixing and power and/or area minimization.

Figure 3.1: Inverter gates with different driver strengths and threshold voltages.



Source: from author (2018)

3.1 Gate Sizing Classification

The gate sizing algorithms falls into two main approaches: continuous gate sizing and discrete gate sizing [Lee and Gupta]. The continuous gate sizing is not in the scope of this work, however, for the sake of contextualization, it will be presented next an overview about it.

3.1.1 Continuous Gate Sizing

In the continuous gate sizing approach, the gates can be implemented using transistors of any size and V_{th}. Due to the continuous nature of the transistor parameters, the gate sizing problem can be formulated using convex delay models. Therefore, it is possible to find a set of continuous sizes and V_{th} that optimally solves the optimization problem. Since the cells must be generated considering the solution found by the gate sizing algorithm, a full-custom methodology or automatic cell generation are required. In literature, several works addressing continuous sizing can be found, such as [Fishburn and Dunlop 1985], [Berkelaar and Jess 1990], [Chen, Chu and Wong 1999], [Tennakoon and Sechen 2002], [Posser et al. 2011] among others.

3.1.2 Discrete Gate Sizing

As mentioned earlier, the discrete gate sizing consists in assigning each gate of the netlist to an cell option available in the cell library. It is a combinatorial optimization problem and [Li 1993] proved to be NP-Hard, thus, efficient heuristic algorithms are fundamental to solve this problem within feasible runtimes. The first works to tackle the discrete gate sizing directly in the discrete domain were [Chan 1990] and [Coudert 1996]. Since then, several approaches have been proposed, such as linear programming [Chinnery and Keutzer 2005], target slew [Held 2009], dynamic programming [Liu and Hu 2010, Ozdal, Burns and Hu 2011], sensitivity guided metaheuristic [Hu et al. 2012], simulated annealing [Reimann et al. 2013] and Lagrangian relaxation [Liu and Hu 2010, Huang, Hu and Shi 2011, Li et al. 2012, Livramento et al. 2013, Flach et al. 2014, Sharma et al. 2015, Reimann, Sze and Reis 2016, Yella and Sechen 2017, Sharma et al. 2017]. All these algorithms are covered in the next chapter.

In this work, the discrete gate sizing is used to minimize the leakage power of a given circuit subject to timing constraints. Therefore, the discrete gate sizing for power optimization can be formally defined as:

$$
\begin{aligned}
\textbf{minimize} \quad & leakage \\
\textbf{subject to} \quad & a_{t_i} + delay_{i \to j} \leq a_{t_j}, \quad \forall i \to j \in \text{Arcs} \\
& a_{t_k} \leq T, \quad \forall k \in \text{Endpoints} \\
& x_i \in \text{Sizes}_i
\end{aligned}
\tag{3.1}
$$

where x_i is an implementation option of the gate i; Sizes_i is the set of implementation options of the gate i; T is the clock period.

Continuous gate sizing can be used to guide the discrete gate sizing. The problem is initially solved in the continuous domain, then, the solution is mapped to the discrete domain. The drawback of this approach is that the mapping process degrades the solution [Ozdal, Burns and Hu 2012]. Some of the works that rely on this strategy are [Chuang, Sapatnekar and Hajj 1993] and [Sirichotiyakul et al. 1999].

3.2 Lagrangian Relaxation based Discrete Gate Sizing

Lagrangian Relaxation is a mathematical technique used to handle optimization problems with hard constraints. In the LR approach, the original optimization problem, called Primal Problem (PP), is rewritten by removing the hard constraints and incorporating them into the objective function, making the problem easier to solve. The PP is described in a general way as follows:

$$
\begin{aligned}
\textbf{minimize} \quad & f(x) \\
\textbf{subject to} \quad & g_i(x) \leq 0 \qquad i = 1, 2, \ \ldots, \ n \\
& h_j(x) = 0 \qquad j = 1, 2, \ \ldots, \ m
\end{aligned}
\tag{3.2}
$$

where $f(x)$ is the function that will be minimized and the hard constraints are given by $g_i(x)$ and $h_j(x)$.

By incorporating the hard constraints into the objective function, it is obtained the relaxed problem, which is given as follows in Equation 3.3:

$$
\begin{aligned}
\textbf{minimize} \quad & \mathcal{L}(x, \lambda, \mu) \\
\textbf{subject to} \quad & \lambda_i \in \Re_+ \qquad i = 1, 2, \ \ldots, \ n \\
& \mu_j \in \Re \qquad j = 1, 2, \ \ldots, \ m
\end{aligned}
\tag{3.3}
$$

where $\mathcal{L}(x, \lambda, \mu)$ is given by

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum \lambda_i g_i(x) + \sum \mu_j h_j(x). \tag{3.4}$$

The weights λ_i and μ_j in Equations 3.3 and 3.4 are called Lagrange multipliers (LM), which indicate how much the constraint is being violated. The solution of the relaxed problem is a lower bound to the solution of the original problem, thus:

$$\mathcal{L}(\tilde{x}, \lambda, \mu) \leq f(\hat{x}) \tag{3.5}$$

where $\hat{x}$ is the optimum solution of the original problem and $\tilde{x}$ is the solution of the relaxed problem. Therefore, the goal of the LR is to find the solution of the relaxed problem closest to the solution of the original problem, in other words, the maximum lower bound. The maximization of the relaxed objective function is called the Lagrangian dual problem (LDP), Equation 3.6.

$$\begin{aligned} &\textbf{maximize} \quad \textbf{min} \quad f(x) + \sum \lambda_i g_i(x) + \sum \mu_j h_j(x) \\ &\textbf{subject to} \qquad\qquad \lambda_i \geq 0 \qquad i = 1, 2, \ldots, n \\ &\qquad\qquad\qquad\qquad\quad \mu_j \in \Re \qquad j = 1, 2, \ldots, m \end{aligned} \tag{3.6}$$

Considering the equation above, the minimization problem is called Lagrangian relaxation subproblem (LRS). Each LRS is defined by a set of Lagrange multipliers.

When applying the LR to solve the discrete gate sizing for leakage power minimization, the objective function defined in the Equation 3.1 can be rewritten by incorporating into it the relaxed constraints, as shown below in Equation 3.7.

$$\begin{aligned} \mathcal{L}_\lambda(x, a_t) \; = leakage + \sum_{\forall i \to j} \lambda_{i \to j} \left(a_{t_i} + delay_{i \to j} - a_{tj}\right) + \\ \sum_{\forall i \in \text{ endpoints}} \lambda_i \left(a_{t_i} - T\right) \end{aligned} \tag{3.7}$$

Therefore, the LRS for a set of multipliers λ is given as follows in 3.8:

$$\begin{aligned} &\mathcal{LRS}_\lambda : \\ &\textbf{minimize} \quad \mathcal{L}_\lambda(x, a_t) \\ &\textbf{subject to} \quad x_i \in \text{Sizes}_i \end{aligned} \tag{3.8}$$

As shown in [Chen, Chu and Wong 1999], by applying the Karush-Kuhn-Tucker

(KKT) conditions for optimality, the $\mathcal{LRS}_\lambda$ can be simplified to:

$$\begin{aligned} &\mathcal{LRS}_\lambda(\text{simplified}) : \\ &\textbf{minimize} \quad leakage + \sum_{\forall i \to j} \lambda_{i \to j} delay_{i \to j} \\ &\textbf{subject to} \quad x_i \in \text{Sizes}_i \end{aligned} \tag{3.9}$$

This simplification implies that the sum of the Lagrange multipliers of the incomming timing arcs of a given pin must be equal to the sum of the Lagrange multipliers of the outgoing arcs of this pin:

$$\sum_{\forall i \to j} \lambda_{i \to j} = \sum_{\forall j \to k} \lambda_{j \to k} \tag{3.10}$$

4 RELATED WORK

This chapter presents a review of the discrete sizing algorithms. First, it is introduced the early work, then, the state-of-the-art techniques are covered.

4.1 Early Work

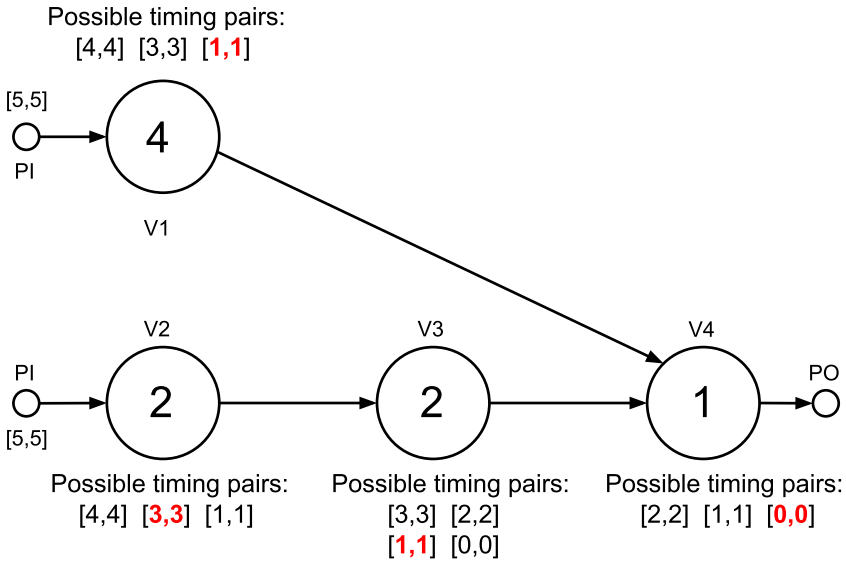
4.1.1 [Chan 1990]

The algorithm proposed in [Chan 1990] has two phases: in the first one, the circuit is traversed from the primary inputs to the primary outputs and, for each visited gate, it is calculated the lower bound and upper bound timing requirements. Considering a gate u, its timing values pair is calculated by subtracting every possible delay that u can assume from the timing requirements that were propagated from its predecessor gates. A gate that has two or more inputs receives pairs of timing values from two or more predecessor gates. Therefore, the timing requirements of this gate are calculated considering only the intersection of the requirements of its predecessors.

In the second phase, the circuit is traversed from outputs to inputs. Each gate is assigned to a cell option that satisfies the timing requirements of the gate and that minimizes the total cost e.g. power. It is presented below in Figure 4.1 an example that considers a simplified timing model in which a delay of a logic gate is inversely proportional to its size and does not depend on the output load. Although the timing model used is simplistic, any timing model can be used in the algorithm. For the example below, the clock period T is 5 and the size options for the gates V1, V2, V3 and V4 are $\{1, \frac{1}{2}, \frac{1}{4}\}$. Due to the delay model used, the delay of each gate is calculated as $delay = \frac{1}{size}$. Since all gates have the same size options, the possible delay all gates can assume are 1, 2 and 4. The algorithm starts by assigning to the primary inputs the pair of timing requirements [5,5], therefore, this pair is propagated to the nodes V1 and V2. For these nodes, the possible pairs of timing requirements are [4,4], [3,3] and [1,1], that are propagated forward. The node V3 received the pairs [4,4], [3,3] and [1,1] from V2, therefore, its possible pairs of timing requirements are [3,3], [2,2], [1,1] and [0,0], that are propagated to V4. The node V4 received from V1 the pairs [4,4], [3,3] and [1,1] and received from V3 [3,3], [2,2], [1,1]. Hence, only the pairs that intersect will be considered in V4, that are [3,3] and [1,1]. From these pairs, the timing requirements calculated for V4 are [2,2], [1,1] and

[0,0]. After all requirements were computed, it is performed the cell option assignment, such that the lower bound and upper bound requirements at each node are satisfied. For this example, the delay at the nodes V1, V2, V3 and V4 are 4, 2, 2 and 1, respectively, and they satisfy the timing requirements highlighted with the red color.

Figure 4.1: Example of timing requirements propagation.



Source: adapted from [Chan 1990]

The algorithm has a pseudo-polynomial runtime if the circuit has a tree structure. Circuits that do not have a tree structure are expanded into a cloned tree.

4.1.2 [Coudert 1996] and [Coudert 1997]

In this work, it is presented a general purpose gate sizing algorithm that can be used to optimize delay, power and area. Different from the previous works, it uses a look-up table delay model since, as the author states, linear delay models are quite inaccurate.

The delay optimization is based in a technique that alternates an optimization and a perturbation steps until there are not room for improvements. In the optimization step, the netlist is traversed and, for each visited gate, the cell options are evaluated based on the gradient of the local cost. For each cell option, the change in the local cost is computed only considering the gates that belong to the first two levels of fanin and fanout. The gate and its implementation option that causes the largest reduction in the local cost are inserted into a list of gates that have potential to be sized. After the whole netlist have been traversed, the algorithm resizes the gates that maximise the slack of the circuit. This optimization method drives the solution into a local minimum, therefore, the perturbation

step is executed to get the solution out of this point, so that a potentially better solution can be found.

The power or/and area optimization is performed on top of a delay optmized solution. It is executed the optimization method described earlir with a local cost function Relax, Equation 4.1. This function balances the gain in power taking into account the delay, so that it acts as a penalty/benefit function. In the Relax function, S_0 is the current slack; ΔS is the slack variation; ΔP is the variation in power; ϵ and α are precomputed values considering characteristics of the circuit.

$$\begin{aligned} Relax &= (\alpha \Delta P - \epsilon) \times \phi\left(\frac{\Delta S}{\epsilon + |S_0|}\right), \text{ where} \\ \phi(x) &= \begin{cases} 1 + x & \text{if} x \geq 0 \\ \frac{1}{1-x} & \text{otherwise} \end{cases} \end{aligned} \tag{4.1}$$

4.1.3 [Chinnery and Keutzer 2005]

[Chinnery and Keutzer 2005] proposed a discrete gate sizing formulation using linear programming (LP) for power reduction. The technique is divided into two main steps that alternate until the power improves less than a threshold value. In the first one, the algorithm starts on top of a design which was previosly optimized by a comercial tool and, for each gate, it is computed its smallest sensitivity value, which is given by:

$$sensitivity = \frac{\Delta P}{\Delta d} \tag{4.2}$$

where ΔP and Δd are the power and delay variations, respectively, w.r.t the current cell option.

The linear program finds for each gate i a real value γ_i in the range [0,1], which indicates if the gate must be swapped to the cell option with the smalles sensitivity. A gate i is swapped if its value is greater than 0.99. If timing constraints were violated, the second step is executed to fix the violations.

4.1.4 [Held 2009]

This work presents an algorithm based on target slew. The technique is divided into two main phases: fast global gate sizing and local search gate sizing.

In the first phase, the algorithm starts by assigning to the output pin of each gate a target slew, such that the input slew limit of the fanout is not violated. Then, iteratively, it is performed a cell assignment step and a target slew refinement until the stopping criterion is met. During the cell assignment step, the circuit is traversed in reverse topological order. Each visited gate is assigned to the smallest cell option, so that that the target slew at its output is met. Since the circuit is being traversed from outputs to the inputs, the fanout gates are already sized, therefore, the downstream capacitance is known. However, the choice of the cell option also depends on the input slew, which is unknown, since the fanin gates have not been sized yet. Hence, considering a predecessor pin $p^{'}$, the input slew is estimated using the followin formula:

$$\text{est\_slew}(p^{'}) = \theta \ \text{slewt}(p^{'}) - (1 - \theta) \ \text{slew}(p^{'}) \tag{4.3}$$

where θ is the weighting factor $\in$ [0,1], such that in the initial iteration θ is equal to 1 and is updated in the slew refinement; slewt($p^{'}$) is the target slew at $p^{'}$; slew($p^{'}$) is the slew at $p^{'}$.

During the slew refinement, the target slew at each gate output pin p is updated based on the global and local criticality of p. The global criticality of p, denoted as $slk^{+}(p)$, is the slack of this pin, whereas the local criticality indicates if the worst slack of p or any predecessor pin of p of cell c can be improved by increasing or decreasing the slewt(p). Hence, it is defined the predecessor criticality of cell c $slk^{-}(c)$ as:

$$slk^{-}(c) = \min\{slack(p^{'}) \mid p^{'} \text{direct predecessor of } c\} \tag{4.4}$$

Therefore, the local criticality of p, $lc(p)$, is calculated considering $slk^{+}(p)$ and $slk^{-}(c)$ as follows:

$$lc(p) = \max\{slk^{+}(p) - slk^{-}(c),\ 0\} \tag{4.5}$$

So, if p is global critical and $lc(p)$ is equal to 0, the target slew of p is decreased by subtrating from it a number that is proportional to $|slk^{+}|$. Otherwise, the target slew of p is increased.

In the local search gate sizing phase, the solution obtained in the fast global gate sizing is improved. In each iteration of this phase, it is selected a small set of gates connected to the most critical nets. Then, each gate of the set is sized to its local optimum considering a more accurate slack evaluation.

4.1.5 [Liu and Hu 2010]

[Liu and Hu 2010] proposed a Lagrangian relaxation based algorithm to address power optimization subject to timing constraints. In their formulation, they consider two components of the power consumption. The first is the dynamic power, given by $1/2\alpha V_{DD}^2 f_{clk} C$, where α is the swithing factor of a gate, f_{clk} is the clock frequency and C is the output capacitance of a gate due to wires and gates. The second is the leakage power, given by $V_{DD} I_{off}$, where the off current I_{off} is provided by the cell library. To solve the Lagrangian Subproblem, they relied on a dynamic programming-like technique, while the Lagrangian dual problem was addressed using the subgradient method [Bazaraa 2003].

The Lagrangian objective function, which was simplified as in [Chen, Chu and Wong 1999], is given by

$$\phi(w, u; \mu) = \sum_{v_i \in V} p(v_i) + \sum_{v_j, v_i \in E} \mu_{ji} D(v_j, v_i) \tag{4.6}$$

where w is the vector of gate sizes, u is the gate V_t levels, $D(v_j, v_i)$ is the delay of the gate v_i and μ_{ji} is the Lagrangian multiplier related to the delay of the gate v_i. The objective function $\phi(w, u; \mu)$ is evaluated through the dynamic programming using the weighted summation of power and delay in the fanout cone, which is recursively calculated traversing the circuit in reverse topological order. The minimum summation of power and delay in the fanout cone is given by

$$f(v_i^k) = \sum min_{h \in options(v_j)} (f(v_j^h) + \mu_{ij} D(v_i^k, v_j^h)) + p(v_i^k) \tag{4.7}$$

where v_i^k is an implementation option of the gate v_i, v_j^h is an implementation option of the fanout gate v_j and $D(v_i^k, v_j^h)$ is the delay of the gate v_i.

After $f(v_i^k)$ is calculated, the circuit is forward traversed in order to assign to each gate an implementation option. The algorithm chooses the option with the lowest cost,

defined by

$$solution(v_i) = min_{k \in options(v_i)}(f(v_i^k) + \sum_{v_j \in fanin(v_i)} (\mu_{ji} D(v_j, v_i^k) + p(v_j))). \quad (4.8)$$

4.1.6 [Huang, Hu and Shi 2011]

As in [Liu and Hu 2010], this work aims the optimization of power consumption (dynamic and static) subject to timing constraints using Lagrangian relaxation. However, the authors state that for discrete gate sizing, the Lagrangian dual problem is no longer convex as in continuous cases, so the subgradient method is inefficient. To handle this drawback, they proposed a projection-based descent method, which relies on the history of previous iterations to update the Lagrangian multipliers. In each iteration, the algorithm calculates the change in the Lagrangian multipliers at the primary outputs using the equation

$$\Delta\mu_i = \frac{q_i - a_i}{\eta^{'}(T_i, \mu_i)} \quad (4.9)$$

where q_i is the required arrival time, a_i is the arrival time and $\eta^{'}$ is function of the current value of the Lagrangian multiplier and the table T_i of past values of arrival times and Lagrangian multipliers. After the multiplier at the primary output is updated, $\Delta\mu$ is distributed to the other nodes of the circuit in reverse topological order. Although μ must be a non-negative value, $\Delta\mu$ can be negative if the constraint is met.

4.1.7 [Ozdal, Burns and Hu 2012]

In [Ozdal, Burns and Hu 2012] (extension published in journal of the paper [Ozdal, Burns and Hu 2011]), the authors addressed discrete gate sizing aiming leakage power minimization for high-performance industrial designs. They proposed a Lagrangian relaxation formulation that decouples timing analysis from optimization, so that the sizer can rely on slack values computed by a signoff timer. This allows the encapsulation of the complexity of the timing analysis into the signoff timing engine, enabling the optimization engine to rely on simpler timing models. In their formulation, the LRS formula

is given as follows:

$$\alpha \times leakage\ power + \sum_{i \to j} \mu_{i \to j} delay_{i \to j} + \sum_{po} \mu_{po}(-r_{t_{po}}) + \sum_{pi} \mu_{pi} a_{t_{pi}} \tag{4.10}$$

where α is a scaling factor to balance leakage power and timing; po and pi are primary outputs and primary inputs, respectively; $i \to j$ is a timing arc from the node i to the node j; $\mu_{i \to j}$, μ_{po} and μ_{pi} are the Lagrange multipliers; $r_{t_{po}}$ is the required time at primary outputs and $a_{t_{pi}}$ is the arrival time at primary inputs. The terms $r_{t_{po}}$ and $a_{t_{pi}}$ are fixed, so, the only variables in this formula are the cell leakage power and the timing arc delay of each cell. The optimization of this LRS formula can be performed relying on the data of the cell library. Thus, the signoff timer is called only once per LR iteration after all cell options were chosen in the LRS solver.

The Lagrange multipliers are updated using the subgradient-based algorithm proposed in [Chen, Chu and Wong 1999]. The updating process relies on two margin values provided by the signoff timer: m_v and $m_{u \to v}$. These margin values are defined as follows:

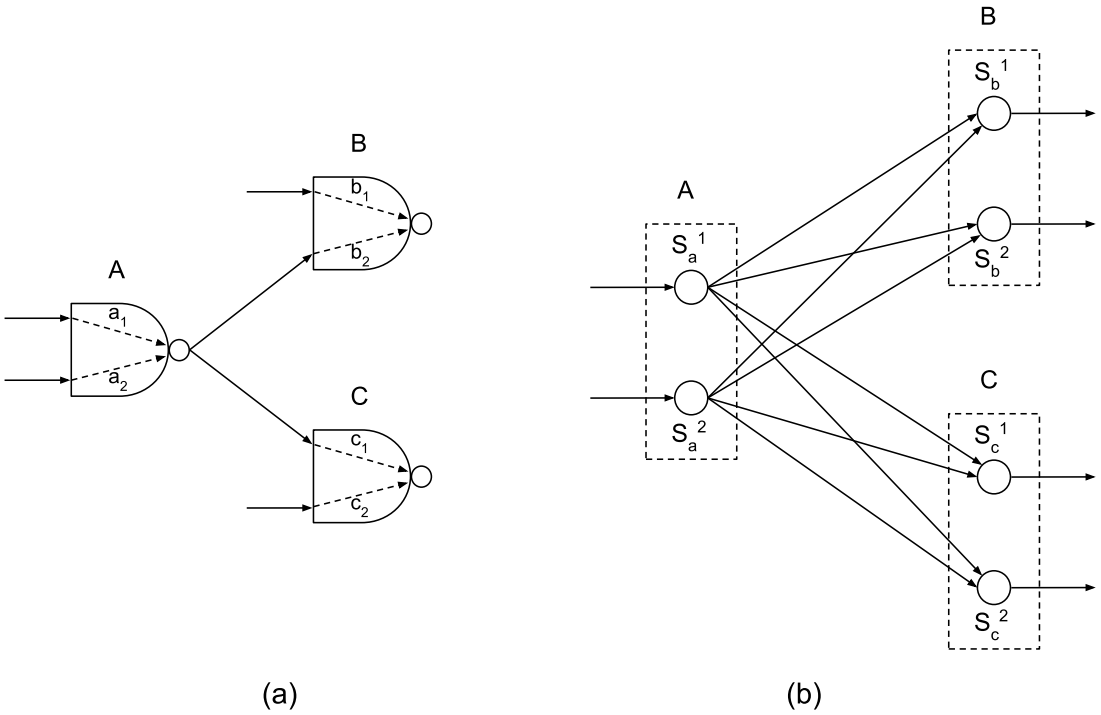
$$m_v = a_v - r_v \tag{4.11}$$

$$m_{u \to v} = a_u + d_{u \to v} - r_v \tag{4.12}$$

The LRS is modeled using a graph model which accurately captures the delay costs of discrete cells using the timing information provided by the lookup tables of the cell library. In their graph model, it is defined a node S_i^j for each cell option j of each combinational gate i of the circuit.

Figure 4.2 (a) shows an example of a subcircuit that will be optimized by the LRS solver. a_1 and a_2 denote the timing arcs of cell A. In the same way, b_1 and b_2 denote the timing arcs of cell B and c_1 and c_2 correspond to the timing arcs of cell C. In Figure 4.2 (b), the subnodes $\{S_a^1, S_a^2\}$, $\{S_b^1, S_b^2\}$ and $\{S_c^1, S_c^2\}$ correspond to the implementation options of the gates A, B and C, respectively.

Figure 4.2: (a) Example of subcircuit. (b) Corresponding graph model.



Source: adapted from [Ozdal, Burns and Hu 2012]

Considering an LR iteration, the weight of a subnode S_i^j of the graph is calculated as follows:

$$weight(S_i^j) = power(S_i^j) + \sum_{k \in arcs(S_i^j)} \mu_k delay\_ref_k(S_i^j) \tag{4.13}$$

where $power(S_i^j)$ is the leakage power of the implementation option S_i^j and μ_k and $delay\_ref_k(S_i^j)$ are the Lagrange multiplier and reference delay of the timing arc k of S_i^j. The reference delay $delay\_ref_k(S_i^j)$ is given as follows:

$$delay\_ref_k(S_i^j) = delay_k \left(\sum_{t \in fanout(i)} cap(S_t^{ref}) \right) \tag{4.14}$$

where S_t^{ref} is the implementation option defined in the previous iteration for the fanout gate t of S_i^j and $delay_k$ is the delay of the timing arc k of S_i^j considering the input load of each cell S_t^{ref}.

The weight of an edge from S_i^j to S_m^n is calculated as follows:

$$weight(S_i^j \rightarrow S_m^n) = \Delta cap(S_m^n) \sum_{k \in arcs(S_i^j)} \mu_k \frac{\partial delay_k}{\partial cap}\bigg|_{ref} + slew\_impact(S_i^j \rightarrow S_m^n) \tag{4.15}$$

where $\Delta cap(S_m^n)$ is the variation in the input capacitance of (S_m^n) w.r.t. the input capacitance of S_m^{ref}; $\frac{\partial delay_k}{\partial cap}$ is the delay sensitive of timing arc k of S_i^j w.r.t. its output load. The slew impact $slew\_impact(S_i^j \rightarrow S_m^n)$ is computed as follows:

$$\begin{aligned} &slew\_impact(S_i^j \rightarrow S_m^n) \; = \\ &\sum_{t \in fanout(i) \setminus m} \left(\mu_t \Delta cap(S_m^n) \times \max_{k \in arcs(S_i^j)} \left\{ \frac{\partial slew_k}{\partial cap} \right\} \times \frac{\partial delay_t}{\partial slew} \right) \end{aligned} \tag{4.16}$$

where $\frac{\partial slew_k}{\partial cap}$ is the output slew sensitivity of timing arc k of S_i^j w.r.t. the output capacitance and $\frac{\partial delay_t}{\partial slew}$ is the delay sensitivity of a timing arc of a fanout gate t w.r.t. the input slew.

It is used a dynamic programming-based algorithm to choose a node of the graph for each gate of the circuit. The graph can contain reconvergent paths. So, the authors proposed a heuristic approach that consists in first extracting the critical trees from the graph, then, each tree is optimized independently using dynamic programming.

4.1.8 [Rahman and Sechen 2012]

The authors proposed an algorithm for V_{th} selection aiming leakage power minimization, which is applied in post synthesis step. The technique starts with a solution without timing violations and seeks to swap each cell to a higher V_{th} version in order to reduce the leakage maintaining the delay goal. So, it is stated that the ideal option to be selected is the one which has the best tradeoff between leakage reduction and total available slack consuming, which allows more cells to have their V_{th} raised. Thus, it is calculated a cost, Equation 4.17, for each gate of the design, which is the ratio of the total available slack reduction and the total leakage reduction when a cell is swapped to the next higher V_{th} option.

$$cost = \frac{Total\_slack\_reduction\_from\_design}{Total\_leakage\_reduction} \tag{4.17}$$

After the cost 4.17 is obtained, the algorithm evaluates the cells in the order of least cost. The change in the V_{th} option is accepted only if the timing was not violated. However, the authors state that simply increasing the V_{th} of each cell does not yield the best result. Therefore, in order to improve the solution, the original target delay is iteratively incremented by Δ time units, so that the V_{th} of more cells can be raised even further. To restore the original target delay, the technique proposed in [Rahman, Tennakoon and Sechen 2011] is applied.

4.1.9 [Hu et al. 2012]

The algorithm proposed is divided into two stages: global timing recovery (GTR) and power reduction with feasible timing (PRFT). In both stages, greedy heuristics are executed in parallel and the best solutions, which are determined by the metaheuristic go-with-the-winners (GWTW) [Aldous and Vazirani 1994], are stored.

The GTR stage starts with a minimum leakage solution and seeks to generate a solution without timing violation by increasing cell sizes or lowering cell V_{th}. For each gate c_i of the circuit, it is calculated the sensitivity of each cell option, given by the equation below

$$sensitivity_{GTR} = \frac{\Delta TNS}{\Delta leakage\_power^{\alpha}} \tag{4.18}$$

where α was empirically determined. It is prohibitive to run the STA whenever the impact on TNS is calculated. Thus, ΔTNS is aproximated by

$$\Delta TNS(m_i^k) \approx \sum_{c_j \in N_i} -\Delta delay_j^k \times \sqrt{NPaths_j} \tag{4.19}$$

where m_i^k is the cell option of c_i, N_i is a set composed by c_i and the gates that have a driver in common with c_i, $\Delta delay_j^k$ is an estimate of the change on delay of a gate c_j due to m_i^k and $NPaths_j$ is the number of fanin and fanout gates of c_j affected by the change on delay of c_j. The sensitivities are sorted in non-increasing order and just the first $\gamma\%$ are commited, since the error after some changes may be considerable. γ was empirically determined and is in the range $0 < \gamma < 60\%$. At each iteration of the GTR, the capacitance violations are fixed.

The PRFT stage, which aims the leakage power reduction, starts with a solution

without timing, load capacitance and slew violations. In this stage, the cells are downsized using a heuristic based on sensitivity. Since five sensitivities functions are employed, Equations 4.20 to 4.24, the heuristic is executed in parallel and the best result, with the smallest leakage power, is considered.

$$SF1 = -\frac{\Delta leakage\_power}{\Delta delay} \tag{4.20}$$

$$SF2 = -\Delta leakage\_power \times slack \tag{4.21}$$

$$SF3 = -\frac{\Delta leakage\_power}{\Delta delay \times \#paths} \tag{4.22}$$

$$SF4 = -\frac{\Delta leakage\_power \times slack}{\#paths} \tag{4.23}$$

$$SF5 = -\frac{Deltaleakage\_power \times slack}{\Delta delay \times \#paths} \tag{4.24}$$

In the Equations above, $\Delta leakage\_power$ and $\Delta delay$ are the changes in leakage power and cell delay after the cell c_i is downsized. $\#paths$ is the number of paths passing through c_i.

4.1.10 [Li et al. 2012]

The authors state that the power optimization subject to timing using Lagrangian relaxation will often be inefficient. Thus, they proposed a hybrid technique that is divided into three stages: Minimum clock period Lagrangian relaxation (Min-Clock LR), network flow based cell optimization and power pruning. In the Min-Clock LR step, the capacitance and slew violations are removed, then the performance of the circuit is maximized through the Lagrangian relaxation method. Since the power consumption is not considered in this stage, the Lagrangian subproblem is given by

$$min\ L\mu = \sum_{uv} \mu^r_{uv} d^r_{uv} + \sum_{uv} \mu^f_{uv} d^f_{uv} \tag{4.25}$$

where d^r_{uv} is the rise delay of the timing arc from the input u to the output v of a cell, μ^r_{uv} is the Lagrangian multiplier associated with the rise delay, d^f_{uv} is the fall delay of the

timing arc from the input u to the output v of a cell and μ_{uv}^{f} is the Lagrangian multiplier associated with the fall delay.

The network flow based cell optimization step starts with a solution free of timing violations and seeks to optimize the power consumption subject to timing constraints. The sizing problem is modeled using a network flow, in which the implementation option of a gate with the largest sensitivity is choosen when a cell is swapped.

In the last step, the power pruning, a greedy heuristic is employed to further reduce the leakage, subject to timing, using the residual slacks obtained in the network flow step.

4.1.11 [Reimann et al. 2013]

In this work, the logical effort [Sutherland, Sproull and Harris 1999] and the fanout-of-4 rule [Rabaey 2002, Weste and Harris 2010] were used to provide a good initial solution. In order to optimize the leakage power of the circuit subject to timing constraints, the simulated annealing was employed. In each iteration, the SA randomly chooses a gate and randomly changes its implementation option. The SA cost function adopted in this work, given by the Equation 4.26, is such that allows some violations during the initial iterations when the temperature is high, and don't accept violations when the temperature is decreased.

$$\begin{aligned} cost = & \alpha \times (timing_{violation} + slew_{violation}) + \\ & \beta \times load_{violation} + leakage_{total} \end{aligned} \tag{4.26}$$

where the terms α and β are given by

$$\alpha = temperature^{-1} \tag{4.27}$$

$$\beta = temperature^{-2} \tag{4.28}$$

This work was submitted to the ISPD 2012 Contest on Discrete Gate Sizing [Ozdal et al. 2012] and achieved the second place at the primary ranking, in which the quality of the solution was evaluated. It also won the first place in the second ranking, in which the tradeoff between quality of the solution and runtime is evaluated.

4.1.12 [Livramento et al. 2013]

In this work, the Lagrangian relaxation is employed to minimize the leakage power of the circuit subject to timing constraints. Different from other works based on LR, it was adopted in this work a LR formulation that incorporates into the LRS the max capacitance and max slew constraints provided by the library. Thus, the Lagrangian function is given by

$$\begin{aligned} L_{\mu,\gamma,\beta}(w,u) = \sum_{v_i \in X} \alpha p_i(w_i,u_i) + \sum_{v_i \in (X \cup PI)} \mu_i D_{ji}(w_i,u_i) + \\ \sum_{v_i \in (X \cup PI)} \gamma_i(out\_slew_i - max\_slew) + \\ \sum_{v_i \in (X \cup PI)} \beta_i(out\_cap_i - max\_cap_i(w_i,u_i)) \end{aligned} \tag{4.29}$$

The authors claim that it is enough to select a cell based only on local information, since the delay of a gate depends only on its neighborhood. Therefore, the Lagrangian subproblem is solved by a greedy heuristic that selects a cell which minimizes locally the Lagrangian function.

It was adopted a modified subgradient method from [Tennakoon and Sechen 2002] to update the Lagrangian multipliers related to the delays. Differently from the traditional subgradient method, which uses a unique step size ρ_k, the modified subgradient method is sensitive to local delay information. To update the Lagrangian multipliers related to the output capacitances, was used a step size which is scaled by the maximum capacitance value of the current implementation of the gate. The authors observed that the slew constraints can be handled by only controlling the capacitance constraints. Thus, the Lagrangian multipliers related to the slew are not employed in the algorithm.

In order to fix remaining capacitance and slew violations, a final LR-based step is executed. This procedure traverses the circuit in reverse topological order and tries to raise the width of gates whose output capacitance is violated.

4.2 State-of-the-Art

4.2.1 [Flach et al. 2014]

This work is an extension of [Flach et al. 2013], that was built upon the ISPD 2012 Contest infrastructure [Ozdal et al. 2012], in which the wires are modeled as lumped capacitances. In this work, the previous tool was changed to support the new ISPD 2013 Contest benchmarks [Ozdal et al. 2013], in which the interconnections are modeled as RC trees.

The algorithm uses Lagrangian relaxation for leakage power minimization subject to timing constraints. It starts assigning each gate to the smallest leakage implementation option. Then, load and slew violations are removed using the algorithm proposed in [Li et al. 2012].

After load and slew violations were removed, the LR-based power optimization phase begins. The Lagrangian subproblem is solved by a greedy heuristic which relies on a Lagrangian cost funtion that captures the timing information of the immediate neighborhood and whole logical cone. In order to quickly estimate how a change in a cell affects the whole logic cone, a technique based on sensitivities is employed.

The Lagrangian multipliers related to each timing arc are updated by multipling their current value by a damping value. The damping value is calculated using the arrival time a and the required timing q at the output of the arc and the clock period T, Equation 4.30.

$$damping = \begin{cases} \left(1 + \frac{a-q}{T}\right)^{+1/k} & a \geq q \\ \left(1 + \frac{q-a}{T}\right)^{-k} & a < q \end{cases} \tag{4.30}$$

After the Lagrangian relaxation based optimization step, if there are some timing violations to be fixed, the timing recovery step is executed to fix them. Finally, to further improve the leakage power consumption of the circuit, a final step called power recovery is executed.

This work was used as the baseline discrete gate sizing algorithm in this work. Therefore, it is covered in detail in the next chapter.

4.2.2 [Sharma et al. 2015]

The algorithm proposed in this work relies on Lagrangian relaxation for leakage power minimization subject to timing constraints. The main contribution of this work is the parallelization of the Lagrangian subproblem solver. The authors observed that there are two key properties that must be satisfied to change two or more cells at the same time: gates being processed simultaneously should not have the same fanin and a gate that will be swapped should not have it's fanin undergoing resizing as well. Thus, to ensure these properties, it was proposed a technique that firstly computes the precendence counter of each gate, which indicates the number of fanins and pseudofanins of the gate. Then, gates whose precendence counter equals zero are queued into the ready queue and picked up by threads. Whenever a gate is changed, the precendence counter of its fanout and pseudofanout gates are decremented. If the precendence counter becomes zero, the fanout gate is queued into the ready queue and waits for a thread to process it.

To further improve the runtime of the algorithm, the STA and Lagrangian multipliers update were parallelized as well. To perform a parallel STA, the circuit is traversed in topological order and groups of ten gates at the same topological level are processed simultaneously. After all cells of the same level were updated, cells of the next level are visited. The Lagrangian multipliers update is performed in the same way, the only difference is that the circuit is traversed in the reverse topological order.

After the Lagrangian relaxation iterations, the fast greedy timing recovery (Fast-GTR) algorithm is executed to fix remaining timing violations without increasing too much leakage power consumption of the circuit. The Fast-GTR is based on the observation that visited gates that do not improve the timing probably won't optimize the timing in future iterations unless one of its side cells is upsized. Thus, such gates are skipped until a side cell is upsized.

4.2.3 [Reimann, Sze and Reis 2016]

In the work presented in [Reimann, Sze and Reis 2016] (extension published in the Integration journal of the paper [Reimann, Sze and Reis 2015]), it is inserted an LR-based discrete gate sizing algorithm after a timing optimization step of an industrial flow. The LR-based sizer is based on [Flach et al. 2014], which is extended to keep the timing quality obtained during the timing optimization step. In order to keep the timing quality,

the Lagrange multipliers update method is modified as follows to handle designs with negative slacks:

Algorithm 4.1: Lagrange Multipliers Update

1 **for** *each timing arc* $i \rightarrow j$ **do**

2 $$\lambda_{i \rightarrow j} \leftarrow \lambda_{i \rightarrow j} \times \begin{cases} \left(1 + \frac{a_{t_j} - r_{t_j} + slk_{init}}{T}\right)^{+1/k} & a_{t_j} \geq r_{t_j} - slk_{init} \\ \left(1 + \frac{r_{t_j} - a_{t_j} - slk_{init}}{T}\right)^{-k} & a_{t_j} < r_{t_j} - slk_{init} \end{cases}$$

3 **end**

4 KKT projection

In the above method, slk_{init} is the initial slack of the timing arc $i \rightarrow j$. This factor aims the preservation of the timing quality of the input solution.

Although the LR-based sizer was successfully integrated into the industrial flow, it presents some drawbacks. It is not able to perform an incremental optimization, since it is not used a set of initial Lagrange multipliers that reflect the timing quality of the input solution. As a consequence, the TNS is degraded in the initial LR iterations. Hence, some iterations are required to mitigate the timing degradation. Another issue is the lack of the capability of handling a multi-objective optimization, such that area and power can be both optimized. Thus, in [Reimann, Sze and Reis 2016], published in ISPD, these drawbacks are addressed. Rather than focusing only in leakage power, it was inserted into the objective function the area, so that the area of the circuit is not dramatically affected in the optimization process. In their new formulation, the LRS is given as follows:

$$\begin{aligned} &\mathcal{LRS}: \\ &\textbf{minimize } \beta \times power + \theta \times area + \alpha \times \sum \lambda_{i \rightarrow j} delay_{i \rightarrow j} \end{aligned} \tag{4.31}$$

In the formulation above, β, θ and α are library dependent scaling factors used to balance power, area and timing, respectively, since they are from different natures. These factors are computed as shown in Equations 4.32, 4.33 and 4.34, where c_n is the cell option with the highest leakage power whereas c_0 is the cell option with the lowest leakage power; $N_{C_{REF}}$ is the number of cell options available in the cell library for a reference cell C_{REF}; $P_l(c)$ and $A(c)$ are the leakage power and area, respectively, of a given cell c; $D(c)$ is the delay of a cell c and is computed based on the cell library with

the same reference output load.

$$\beta = \frac{N_{C_{REF}}}{P_l(c_n) - P_l(c_0)} \tag{4.32}$$

$$\theta = \frac{N_{C_{REF}}}{A(c_n) - A(c_0)} \tag{4.33}$$

$$\alpha = \frac{N_{C_{REF}}}{D(c_n) - D(c_0)} \tag{4.34}$$

In order to make the algorithm capable of incremental optimization, it is proposed a simple method to provide a set of good initial lagrange multipliers values that reflect the timing quality of the input solution. The initial Lagrange multipliers are obtained by running a few LR iterations. At the end of each iteration, the initial solution of the design is restored.

As in the previous work, the Lagrange multiplier update method enables the algorithm to handle designs with negative slack, since the designs may have timing violations that are not expected to be solved during the sizing. However, the new work relies on a more aggressive update method to achieve fast convergence.

In the new flow, it is incorporated a sign-off timer to ensure the accurary of the timing of the solution. However, evaluating all cells options during the LRS solver using a sign-off timer is very timing consuming. Thus, the authors proposed a cell filtering technique, which is based on cell options ranking. Each cell option is ranked using a less accurate timing model. After that, only the top t ranked cells are evaluated using the sign-off timer. In their experiments, t was set to 2.

It was also incorporated into the flow modified versions of the timing recovery and power reduction methods from [Flach et al. 2014]. The timing recovery was adapted to take into account placement legalization. A cell option being evaluated could not fit in the current position due to increase in its size, so the algorithm searches a new location for the cell. Thus, the timer considers the change in the interconnection during the timing update. The power reduction was enhanced to prioritize the choice of a smaller size cell for gates in the fanout of critical paths.

4.2.4 [Yella and Sechen 2017]

Different from previous works based on Lagrangian relaxation, this algorithm considers in the LRS cost function the delay of wires, which are computed using SPEF delay calculation [Wang and Sechen 2014]. Yet, the number of lambda delay arcs is reduced to half, since, for each timing arc, just the worst case between rise and fall delays are considered.

In the LRS cost minimization step, the implementation option that locally minimizes the lambda delay and leakage product is initially choosen and the next two best options are stored as well to be explored in the future by the proposed *try_new_cell_option* algorithm for timing recovery. After each LR iteration, the algorithm tries to down size and raise the V_{th} of non-critical cells in order to reduce the leakage power.

The authors state that if a design has a large number of logic levels, the LRS cost function of gates in topological lower levels is dominated by the lambda-delay product instead of the leakage. Thus, when the Lagrange multipliers are propagated from primary outputs towards the inputs, a constant term called *level_factor* is introduced to make the λs near primary outputs more emphasized than the λs near primary inputs.

4.2.5 [Sharma et al. 2017]

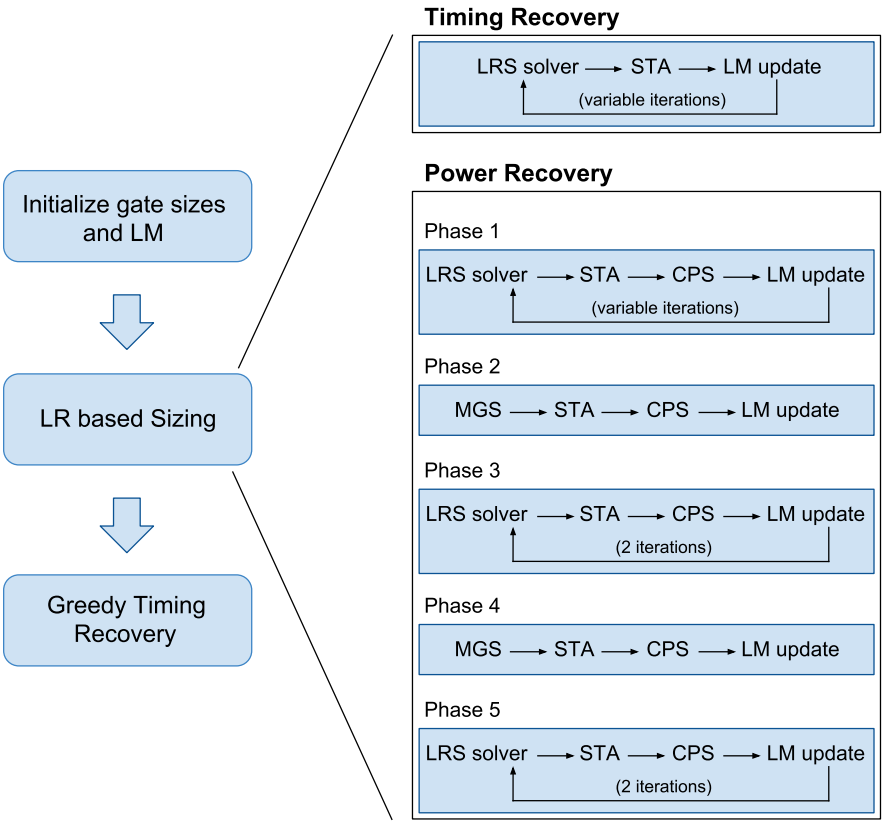
It was proposed in [Sharma et al. 2017] the rapid gate sizing (RGS), an LR based gate sizing algorithm for leakage power minimization. As illustrated in Figure 4.3, the algorithm is mainly divided into 3 phases: initialization, LR optimization and a greedy post-processing for timing violations removal.

In the initialization phase, all cell are swapped to the smallest leakage implementation option and the circuit is traversed in reverse topological order to remove load and slew violations, as in [Li et al. 2012] and [Flach et al. 2013]. Also, the Lagrange multipliers are set to 1.

After the initializations were performed, the LR based optimization phase takes place. The LR is divided into two main stages: timing recovery and power recovery. In the timing recovery, most of the timing violations are fixed. It is basically composed of LRS solving and Lagrange multipliers update, as in the traditional LR based gate sizing algorithms. This stage runs until the TNS is not below a specified threshold value.

The power recovery is divided into 5 steps. In the first step, it is performed a

Figure 4.3: RGS flow.



Source: adapted from [Sharma et al. 2017]

coarse-grained leakage power optimization. It iteratively alternates between LRS solving, STA update, critical path sizing (CPS) and Lagrange multipliers update until the improvement in power is not less than a specified threshold value. The CPS is employed to reduce the timing violations of the critical paths. During the iterations, the CPS is invoked only when the timing violations exceed a specified threshold value, so that the overall timing violations of the cricuit are kept under control.

In phases two through five, it is performed a finer-grained power optimization. Phase two starts by executing the multi-gate sizing (MGS), followed by a STA update, CPS and Lagrange multiplier update. In the MGS, the circuit is traversed in topological order and the algorithm tries to downsize each gate g visited. If the change does not generate load and slew violations, the fanout of g is sized as well. In order to avoid large perturbations during the MGS and also to favor the runtime, only three implementation options are evaluated for each fanout of g: the current cell option, the cell option with the next smaller size and the option with the next bigger size. In phase three, it is executed two iterations composed of LRS solving, STA update CPS and Lagrange multipliers update. Phases four and five are the same of phases two and three, respectively.

The Lagrange multipliers are updated using the Equation 4.35:

$$\lambda_{i \to j} = \lambda_{i \to j} \times \left(\frac{delay_{i \to j}}{T} \right)^{k} \tag{4.35}$$

where $\lambda_{i \to j}$ is the Lagrange multiplier associated with the timing arc $i \to j$ and $delay_{i \to j}$ is the worst path delay through the timing arc $i \to j$. The exponent k assumes different values in the timing recovery and power recovery stages. When the timign recovery is executed, k is set to 4 for critical arcs and 1 for non-critical arcs, so that the delay of timing arcs with timing violations are emphasized, enabling a fast timing recovery. During the power recovery, k is set to 1 for critical arcs and 6 for non-critical arcs. Thus, the leakage power is emphasized, allowing the power recoveryto quickly reduce the power.

After the LR based optimization phase, there could be remaining timing violations. In such cases, the greedy timing recovery is executed. The timing recovery used in this work is similar to the one used in [Flach et al. 2013].

4.3 Summary

Table 4.1 summarizes the discrete gate sizing algorithms covered in this chapter, presenting the optimization techniques employed in each one.

Table 4.1: Summary of the discrete gate sizing algorithms presented in this chapter.

Work	**Optimization techniques**
[Chan 1990]	Greedy heuristic
[Coudert 1996]	Greedy heuristic
[Chinnery and Keutzer 2005]	Linear programming
[Held 2009]	Target slew
[Liu and Hu 2010]	Lagrangian relaxation
[Huang, Hu and Shi 2011]	Lagrangian relaxation
[Ozdal, Burns and Hu 2012]	Lagrangian relaxation and dynamic programming
[Rahman and Sechen 2012]	Lagrangian relaxation
[Hu et al. 2012]	Sensitivity
[Li et al. 2012]	Lagrangian relaxation and network flow
[Reimann et al. 2013]	Simulated annealing
[Livramento et al. 2013]	Lagrangian relaxation
[Flach et al. 2014]	Lagrangian relaxation
[Sharma et al. 2015]	Lagrangian relaxation
[Reimann, Sze and Reis 2016]	Lagrangian relaxation
[Yella and Sechen 2017]	Lagrangian relaxation
[Sharma et al. 2017]	Lagrangian relaxation

5 BASELINE DISCRETE GATE SIZING FLOW

This chapter covers in detail the discrete gate sizing algorithm proposed in [Flach et al. 2013], which was used as the baseline flow in this work. This algorithm has the best leakage power results published so far for the ISPD 2012 Discrete Gate Sizing Contest benchmarks [Ozdal et al. 2012].

5.1 Formulation

As described earlier in subsection 2.1.2, the digital circuit is modeled as a timing graph, in which the nodes represent pins and the edges are the timing arcs. All the information which is commonly used throughout this chapter is presented in Table 5.1.

Table 5.1: Definitions.

T	clock period
TNS	total negative slack
STA	static timing analysis
$i \rightarrow j$	timing arc from node i to node j
$delay_{i \rightarrow j}$	delay of timing arc $i \rightarrow j$
a_{t_i}	arrival time at node i
r_{t_i}	require time at node i
$slew_i$	slew at node i
$islew_{i \rightarrow j}$	input slew of timing arc $i \rightarrow j$
$oslew_{i \rightarrow j}$	output slew of timing arc $i \rightarrow j$
$\lambda_{i \rightarrow j}$	Lagrange multiplier of timing arc $i \rightarrow j$

The discrete gate sizing for leakage power minimization is formulated as follows:

$$
\begin{aligned}
&\text{Primal Problem (PP):} \\
&\textbf{minimize} \quad leakage \\
&\textbf{subject to} \quad a_{t_i} + delay_{i \rightarrow j} \leq a_{t_j}, \forall i \rightarrow j \in \text{Arcs} \\
&\qquad\qquad\quad\; a_{t_k} \leq T, \forall k \in \text{Endpoints} \\
&\qquad\qquad\quad\; x_i \in \text{Sizes}_i
\end{aligned}
\tag{5.1}
$$

The Lagrangian relaxation subproblem is obtained by incorporating the constraints

into the objective function:

$$
\begin{aligned}
&LRS\text{:}\\
&\textbf{minimize} \quad leakage + \\
&\qquad \sum \lambda_{i\to j}(a_{t_i} + delay_{i\to j} - a_{t_j}) + \\
&\qquad \sum \lambda_k(a_{t_k} - T)
\end{aligned}
\tag{5.2}
$$

By applying the Karush-Kuhn-Tucker conditions, the problem presented above is simplified to:

$$
\begin{aligned}
&LRS \text{ (simplified):}\\
&\textbf{minimize} \quad leakage \; + \; \sum \lambda_{i\to j} delay_{i\to j}
\end{aligned}
\tag{5.3}
$$

where the term $\sum \lambda_{i\to j} delay_{i\to j}$ is referred as lambda-delay.

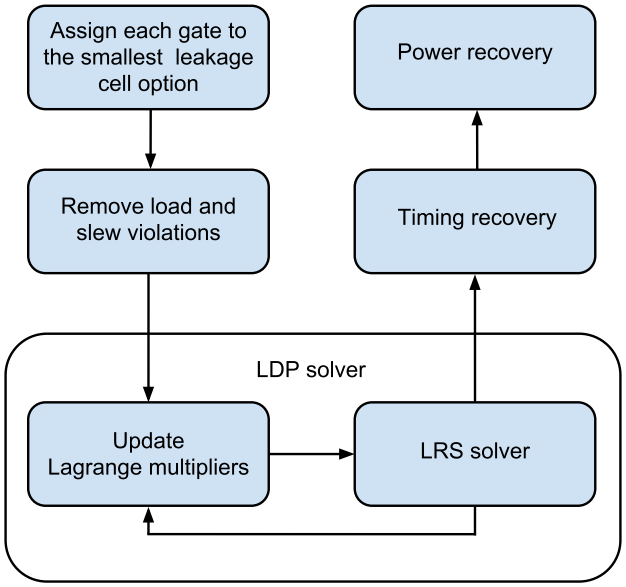
At last, by maximizing the LRS, it is obtained the Lagrangian dual problem:

$$
\begin{aligned}
&LDP\text{:}\\
&\textbf{maximise} \quad \Big(\textbf{minimize}\; leakage \; + \; \sum \lambda_{i\to j} delay_{i\to j}\Big)
\end{aligned}
\tag{5.4}
$$

5.2 Baseline Flow Overview

The overall flow of the base algorithm [Flach et al. 2013] is presented in Figure 5.1. The first step is the assignment of each gate to the smallest leakage cell option. Next, load and slew violations are removed. After performing the preconditioning of the solution, the LR-based power optimization comes into play. If any timing violation is left after the LR terminates, a post-processing timing fixing algorithm is executed. Finally, in order to reduce even further the leakage power, a power recovery algorithm is executed.

Figure 5.1: Overall flow of the baseline sizer.



Source: adapted from [Flach et al. 2014]

5.2.1 Load and Slew Violations Removal

The removal of load and slew violations are performed using a technique based on the method presented in [Li et al. 2012]. The circuit is traversed in reverse topological order. For each gate visited, it is selected the smallest leakage cell option such that the gate is able to drive a load 30% greater than the actual load being driven. Since each interconnection is modeled using a lumped capacitance, the slew is not degraded. Hence, when it is chosen a cell that satisfies the load condition aforementioned, it is assumed that the slew is fixed as well.

This step produces a solution without load and slew violations. So, during the optimization process, each cell option is validated so that load and slew violations are not created. The Algorithm 5.1 summarizes the load and slew violations process.

Algorithm 5.1: Load and Slew Violations Removal

```
1 for each gate g in reverse topological order do
2     for each cell option c from lowest leakage to largest leakage do
3         if 0.7 × max_load(c) > actual_load(g) then
4             stop for loop
5         end
6     end
7 end
```

5.2.2 Lagrangian Dual Problem Solver

The main phase of the flow is based on Lagrangian relaxation. In Algorithm 5.2, it is presented the Lagrangian dual problem solver. At each iteration of the LDP, the Lagrange multipliers (λ) are updated, so that they indicate how much a constraint, that is incorporated into the objective function, is being violated. The resulting set of λs define a specific Lagrangian relaxation subproblem, which is greedly solved using the method shown in Algorithm 5.5. The initial value of λs was empirically determined and is equal to 12. During each iteration, the solution found is accepted only if its cost is smaller than the cost of the previous solution and the worst negative slack is less than 10% of the clock period.

Algorithm 5.2: LDP Solver

```
1  set initial value of λs to 12
2  update λs //Alg. 5.3
3  update Lambda-Delay Sensitivities
4  bestCost ← +∞
5  γ ← (−(min(0, worstSlack))/T + 1)
6  for iter ← 1; iter ≤ 100; iter++ do
7      solve LRS //Alg. 5.5
8      update timing (STA)
9      γ ← (−(min(0, worstSlack))/T + 1)
10     cost ← γ × total_leakage
11     if cost < bestCost and γ < 1.1 then
12         store solution
13         bestCost ← cost
14     end
15     update λs //Alg. 5.3
16     update Lambda-Delay Sensitivities
17 end
18 restore best solution found
```

5.2.2.1 Lagrange Multipliers Update Method

For each transition edge of each timing arc, the λ is updated by multiplying its current value by a $damping$ value, as shown in Algorithm 5.3. After the Lagrange multipliers were updated, it is performed the KKT projection, Algorithm 5.4, to ensure that the λs satisfy the KKT conditions of optimality.

Algorithm 5.3: Lagrange Multipliers Update Method

for *each timing arc* $i \rightarrow j$ **do**

$$damping = \begin{cases} \left(1 + \frac{a_{t_j} - r_{t_j}}{T}\right)^{+1} & a_{t_j} \geq r_{t_j} \\ \left(1 + \frac{r_{t_j} - a_{t_j}}{T}\right)^{-1} & a_{t_j} < r_{t_j} \end{cases}$$

$$\lambda_{i \rightarrow j} = \lambda_{i \rightarrow j} \times damping$$

end

KKT projection //Alg. 5.4

Algorithm 5.4: KKT projection

```
for each pin p in reverse topological order of the circuit do
    if number of outgoing arcs of p == 0 then
        continue
    end

    Compute sum of incoming timing arc lambdas of pin p
    Compute sum of outgoing timing arc lambdas of pin p

    for each edge (rise and fall) do
        for each incoming timing arc do
            λ_arc = sumOutgoingLambdas * (λ_arc / sumIncomingLambdas)
        end
    end
end
```

5.2.3 Lagrangian Relaxation Subproblem Solver

Each Lagrangian relaxation subproblem instance is minimized through the method presented in Algorithm 5.5. It traverses the circuit in topological order and, for each gate, it selects the cell option that locally minimizes the sum of leakage power and lambda-delay cost.

Whenever a cell is swapped, running a full STA update or an incremental STA update would make the LRS solver very timing consuming. Hence, only the timing information related to the cell being changed and its neighboring cells is updated.

Since the LR method starts upon a solution without load and slew violations, the LRS solver filters cell options that create these violations. Thus, it is not necessary to handle load and slew violations ahead.

The term $localNegativeSlack$ is referred as the sum of the negative slacks at

the output pins of the current gate and its neighboring gates (fanin, fanout and the gates being driven by the fanin of the current gate). The parameter γ is used to simulate a "hill climbing" of stochastic methods. Hence, during the initial iterations, when the timing violations are high, the local negative slack is allowed to increase a little, so that larger changes are allowed. As the solution converges to a low timing violation solution, larger changes are avoided.

Algorithm 5.5: LRS Solver

```
1  for each gate g in topological order do
2      originalNegativeSlack ← computeLocalNegativeSlack(g)
3      bestCandidate ← version(g)
4      bestCost ← lambdaDelayCost(g) + leakage(g)
5
6      for each gate version t ∈ versions(g) do
7          version(g) ← t
8
9          if load violation has increased then
10             go to te next candidate
11         end
12
13         update timing locally
14         localNegativeSlack ← computeLocalNegativeSlack(t)
15
16         if localNegativeSlack < γ * originalNegativeSlack then
17             go to te next candidate
18         end
19
20         cost ← lambdaDelayCost(t) + leakage(t)
21         if cost < bestCost then
22             bestCandidate ← t
23             bestCost ← cost
24         end
25     end
26     version(g) ← bestCandidate
27     update timing locally
28 end
```

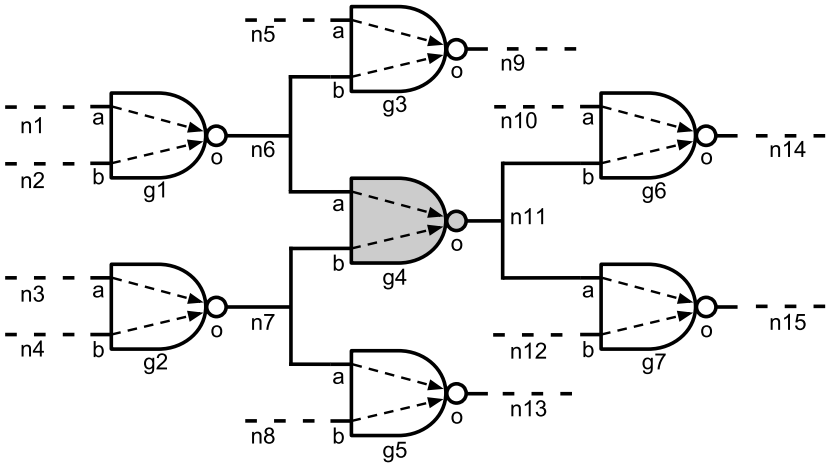
5.2.3.1 Lambda-Delay Cost of a Cell Option

As shown in Equation 5.3, the objective function of the LRS problem consists in minimizing the sum of the leakage power and lambda-delay. Whenever a gate is swapped, the lambda-delay of the whole logic cone starting from the fanin of the swapped gate

may be affected. Thus, the lamba-delay cost of a cell option indicates how much the cell option affects the lambda-delay of the logic cone. Keeping the timing information of the logic cone updated when a cell is changed would require an incremental STA. However, performing an incremental STA everytime a cell option is evaluated would lead to infeasible runtimes. Hence, as mentioned earlier, only the timing information of the neighboring cells of the cell being changed is updated. For the remaining cells of the logic cone, the lambda-delay cost computation relies on an estimation of how the timing of these cells is affected.

Considering a gate g, e.g., the g4 gate highlighted in Figure 5.2, the lambda-delay cost of each one of its cell options is given in Equation 5.5.

Figure 5.2: Example of Lambda-Delay Cost Computation



Source: from author (2018)

$$lambdaDelayCost(g) = \sum_{i \to j \in driverArcs(g) \cup gateArcs(g) \cup sinkArcs(g)} \lambda_{i \to j} delay_{i \to j} + \sum_{i \to j \in sideArcs(g)} \Delta islew_{i \to j} \phi_{i \to j} + \sum_{n \in drainNets(g)} \Delta slew_n \Phi_n \quad (5.5)$$

where

- $driverArcs(g)$ is the set of arcs driving the driver nets of g (e.g g1_a→g1_o, g1_b→g1_o, g2_a→g2_o and g2_b→g2_o in Figure 5.2);
- $sinkArcs(g)$ is the set of arcs driven by the output net of g (e.g g6_a→g6_o, g6_b→g6_o, g7_a→g7_o and g7_b→g7_o in Figure 5.2);
- $sideArcs(g)$ is the set of arcs driven by the driver nets of g, but which do not belong to g (e.g. g3_b→g3_o and g5_a→g5_o in Figure 5.2);

- $gateArcs(g)$ is the set of arcs of g (e.g. g4_a→g4_o and g4_b→g4_o in Figure 5.2);
- $drainNets(g)$ is the set of nets driven by sink gates of g (e.g. n14 and n15 in Figure 5.2);
- $\Delta islew_{i \to j}$ is the input slew change at the input of timing arc $i \to j$;
- $\Delta slew_n$ is the slew change at net n;
- $\phi_{i \to j}$ is the cumulative back-propagated lambda-delay sensitivity of arc $i \to j$ (explained later);
- Φ_n is the cumulative back propagated lambda-delay sensitivity at net n (explained later).

5.2.3.2 Back-Propagated Cumulative Lambda-Delay Sensitivity

As stated earlier, when a cell option is evaluated, it is performed only a local timing update, since an incremental STA would be very timing consuming. Thus, in order to quickly evaluate the timing change not only in the vicinity, but also in the whole logical cone, the back-propagated cumulative lambda-delay sensitivity is used.

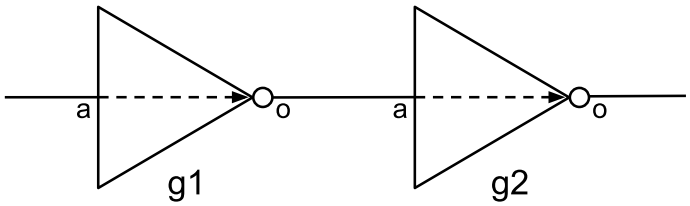
The lambda-delay sensitivity of an timing arc $i \to j$ is defined as:

$$lambda\text{-}delay\ sensitivity = \lambda_{i \to j} \frac{\delta delay_{i \to j}}{\delta islew_{i \to j}} \tag{5.6}$$

The sensitivities can be back-propagated from outputs to inputs. Thus, the resulting cumulative sensitivity of a timing arc provides in a single operation an estimation of how much the timing of the logical cone starting at such arc is affected due to a change in its input slew. During the back-progation process, it is assumed that each interconnection is modeled using a lumped capacitance, thus, the output slew of a gate is not degraded.

It is presented below in Figure 5.3 an example which shows how it is computed the back-propagated cumulative lambda-delay sensitivities of the timing arc g1_a→g1_o.

Figure 5.3: Example of Lambda-Delay Sensitivity Computation



Source: from author (2018)

The lambda-delay change in the timing arc g2_a→g2_o due to a change in its input slew is computed by multiplying the timing arc lambda-delay sensitivity by the input slew change:

$$\Delta delay_{\text{g2\_a}\rightarrow\text{g2\_o}} = \Delta islew_{\text{g2\_a}\rightarrow\text{g2\_o}} * \lambda_{\text{g2\_a}\rightarrow\text{g2\_o}} * \frac{\delta delay_{\text{g2\_a}\rightarrow\text{g2\_o}}}{\delta islew_{\text{g2\_a}\rightarrow\text{g2\_o}}} \tag{5.7}$$

In the same way, the lambda-delay change for the timing arc g1_a→g1_o is given the lambda-delay sensitivity times the change in its input slew plus the delay change in timing arc g2_a→g2_o, which was shown in Equation 5.7.

$$\begin{aligned}\Delta delay_{\text{g1\_a}\rightarrow\text{g1\_o}} = \Delta islew_{\text{g1\_a}\rightarrow\text{g1\_o}} * \lambda_{\text{g1\_a}\rightarrow\text{g1\_o}} * \frac{\delta delay_{\text{g1\_a}\rightarrow\text{g1\_o}}}{\delta islew_{\text{g1\_a}\rightarrow\text{g1\_o}}} + \\ \Delta delay_{\text{g2\_a}\rightarrow\text{g2\_o}}\end{aligned} \tag{5.8}$$

Assuming that

$$\Delta islew_{\text{g2\_a}} \approx \Delta islew_{\text{g1\_a}\rightarrow\text{g1\_o}} * \frac{\delta oslew_{\text{g1\_a}\rightarrow\text{g1\_o}}}{\delta islew_{\text{g1\_a}\rightarrow\text{g1\_o}}} \tag{5.9}$$

and by combining 5.7 and 5.7, the delay change for timing arc g1_a → g1_o is obtained, as shown in Equation 5.10. Thus, the delay change depends only on the back-propagated cumulative lambda-delay sensitivity (the terms summed inside the parentheses) and the unknown $\Delta islew_{\text{g1\_a}}$.

$$\begin{aligned}\Delta delay_{\text{g1\_a}\rightarrow\text{g1\_o}} = \Delta islew_{\text{g1\_a}\rightarrow\text{g1\_o}} * \Bigg(\lambda_{\text{g1\_a}\rightarrow\text{g1\_o}} * \frac{\delta delay_{\text{g1\_a}\rightarrow\text{g1\_o}}}{\delta islew_{\text{g1\_a}\rightarrow\text{g1\_o}}} + \\ \lambda_{\text{g2\_a}\rightarrow\text{g2\_o}} * \frac{\delta oslew_{\text{g1\_a}\rightarrow\text{g1\_o}}}{\delta islew_{\text{g1\_a}\rightarrow\text{g1\_o}}} * \frac{\delta delay_{\text{g2\_a}\rightarrow\text{g2\_o}}}{\delta islew_{\text{g2\_a}\rightarrow\text{g2\_o}}} \Bigg)\end{aligned} \tag{5.10}$$

Generalizing, the delay change of a timing arc $i \rightarrow j$ is defined as follows:

$$\Delta delay_{i\rightarrow j} = \Delta islew_{i\rightarrow j} \phi_{i\rightarrow j} \tag{5.11}$$

where $\phi_{i\rightarrow j}$ is the back-propagated cumulative lambda-delay sensitivity of the timing arc

$i \rightarrow j$. $\phi_{i \rightarrow j}$ is computed using the following recurrence relation:

$$\phi_{i \rightarrow j} = \frac{\delta delay_{i \rightarrow j}}{\delta islew_{i \rightarrow j}} + \frac{\delta oslew_{i \rightarrow j}}{\delta islew_{i \rightarrow j}} \Phi_n \tag{5.12}$$

where n is the net driven by the timing arc $i \rightarrow j$ and Φ_n is $\sum \phi_{i \rightarrow j'}$ for each timing arc $i \rightarrow j^{'}$ driven by n.

5.2.4 Timing Recovery

The timing recovery (TR) step is executed after the LR optimization step if there are remaining timing violations to be fixed, i.e., if the TNS is higher than ϵ (ϵ = 1e-6). The TR method is presented in Algorithm 5.6. The algorithm starts by sorting the nets in decreasing number of critical paths passing through them. For each net n of the list of sorted nets, the algorithm tries to swapp the driver of n to the next larger cell option. The new cell option is accepted if it does not generate slew and load violations and if it does not worsen the TNS. Whenever a cell is changed, the list of nets are sorted again and the algorithm continues by the first net sorted. This process stops when there are not timing violations left.

5.2.5 Power Recovery

In order to further reduce the leakage power of the solution, the power recovery (PR) algorithm is executed. The PR method is presented in Algorithm 5.7. It is divided into two main steps: in the first one, the circuit is traversed in topological order and, for each gate g visited, the algorithm tries to swapp g to the next higher V_{th} cell option. In the second step, the circuit is traversed in topolical order again, but the algorithm tries to change each gate g to the first smaller size option. These two steps alternates iteratively until no changes are made.

Algorithm 5.6: Timing Recovery

```
1  Sort nets in decreasing number of critical paths passing through them
2  for each net n sorted by critical path counter do
3      t is the actual cell type of the gate g driver of n
4      t + 1 is the next larger cell option with the same V_th
5
6      Try to upsize the gate driver on n changing t by t + 1
7      previousTNS ← TNS
8
9      if g is upsizable then
10         update TNS
11
12         if TNS < previousTNS then
13             update cell type to t + 1
14
15             if TNS < ε then
16                 break
17             end
18
19             Update the number of critical paths through each net
20
21             Process continues by the first net sorted by critical
22             path counter
23
24         end
25     end
26 end
```

Algorithm 5.7: Power Recovery

```
repeat
    changedCellsCounter ← 0
    for each gate g of the circuit in topological order do
        if V_th of g is increasable then
            increase Vth of g
            update timing (STA)
            if TNS ≥ 0 and no load/slew violations generated then
                changedCellsCounter++
            else
                undo
            end
        end
    end
    for each gate g of the circuit in topological order do
        if g is downsizable then
            downsize g
            update timing (STA)
            if TNS ≥ 0 and no load/slew violations generated then
                changedCellsCounter++
            else
                undo
            end
        end
    end
until changedCellsCounter = 0;
```

6 PROPOSED IMPROVEMENTS

In this chapter, it is presented several extensions proposed to enhance the base algorithm [Flach et al. 2013]. First, it is discussed some drawbacks related to the LR phase. Then, it is presented a set of improvements proposed to handle these drawbacks. Yet, it is proposed some modifications to the post-LR timing recovery and power phases that were done in order to improve the runtime. Finally, the experimental results obtained are discussed.

6.1 Points for Improvement in the LR Core of the Baseline Work

Although the baseline work [Flach et al. 2013] has the best leakage power results published so far for the set of benchmarks of the ISPD 2012 Contest on Discrete Gate Sizing [Ozdal et al. 2012], it presents some drawbacks in its LR core. The first point is that the algorithm requires an expressive number of iterations (100) to converge to a good solution. A portion of the iterations is used to reduce the huge increase in leakage power (e.g. $15\times$ the final value) that occurs in the first iterations of the LR. Also, the algorithm does not rely on any cell option candidate filtering strategy, thus, computing the lowest LR local cost can be a slow process. Therefore, in the subsequent sections, it is presented a set of improvements to tackle these drawbacks. The objective of the proposed strategies is to reduce the number of iterations required by the LR core and dramatically reduce the number of cell options evaluated during the LR phase.

6.2 New Lagrangian Relaxation Formulation

For each gate of the circuit, the Lagrangian relaxation subproblem solver of the baseline work greedly selects the cell option that locally minimizes the sum of timing cost and leakage power cost. However, these two quantities are from different natures, therefore, they must be scaled. A consequence of summing up these two quantities without proper scaling them is the huge increase in the leakage power during the initial LR iterations. This impacts in the runtime, since part of the iterations will be used to reduce the leakage peak. Also, the LR slowly adjusts the Lagrange multipliers such that a scaling factor would not be necessary. However, it is a waste of runtime.

Initially, the solution has excessive timing violations, therefore, the LR local cost is highly dominated by the timing cost. By scaling timing and power costs, the importance of the leakage power is increased, therefore, the LR core tends do be less aggressive in the initial iterations, reducing the excessive increase in the leakage power. In this work, the scaling is performed by multiplying the leakage power cost by a library-dependent factor α called average delay per leakage, whose computation is detailed later in this chapter. Thus, by defining α as delay/leakage, only quantities with the same unit are summed up in the cost function, as shown below:

$$\begin{aligned} localCost &= timingCost + \alpha \times leakage \\ localCost &= ps + ps/mW \times mW \\ localCost &= ps + ps \end{aligned} \tag{6.1}$$

Therefore, the new LR formulation used in this work is defined as:

$$\begin{aligned} &LRS: \\ &\textbf{minimize} \quad \alpha \times leakage \; + \; \sum \lambda_{i \to j} delay_{i \to j} \end{aligned} \tag{6.2}$$

6.3 Updated Flow

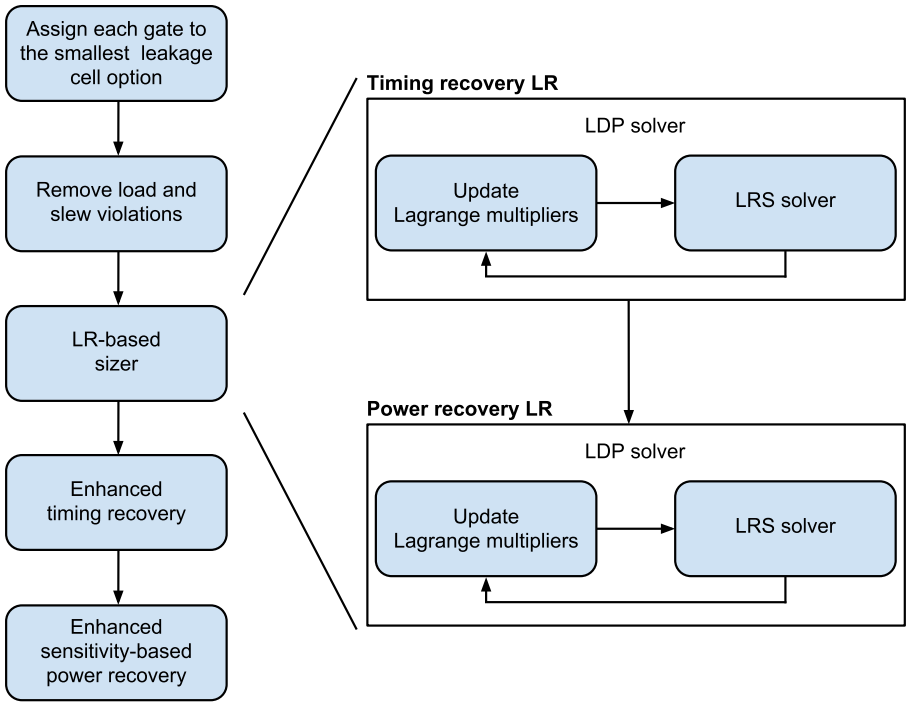
Figure 6.1 depicts the overall flow of the sizer with the modifications proposed in this work. The first two steps of the flow are the same of the first two steps of the baseline work: first, each gate is assigned to the smallest cell option and, next, load and slew violations are removed using the algorithm presented in the last chapter.

After the solution is preconditioned, the Lagrangian relaxation phase comes into play. As in [Sharma et al. 2017], the LR phase is divided into two main stages: timing recovery Lagrangian relaxation (TR-LR) and power reduction Lagrangian relaxation (PR-LR). During the TR-LR, the optimization is focused on fixing most of the timing violations. In order to quickly reduce the number of endpoints with negative slack, the Lagrange multiplier formula is adjusted to emphasize delay over leakage power. The PR-LR starts as soon as the TR-LR terminates. This stage is focused on power optimization. Thus, the Lagrange multiplier formula is adjusted to enable a quickly power reduction without harming too much the timing of the circuit.

After the LR phase, the remaining timing violations are fixed in the enhanced

timing recovery phase. Finally, the leakage power of the solution is further reduced in the enhanced sensitivity-based power recovery.

Figure 6.1: Sizer flow with the modifications proposed in this work.



Source: from author (2018)

6.3.1 Timing Recovery Lagrangian Relaxation

The LDP solver for the TR-LR is presented in Algorithm 6.2. It starts by setting each Lagrange multiplier to 1. As stated before, the TR-LR is focused on quickly removing most of the timing violations. Hence, at each iteration, the Lagrange multipliers are updated such that a rapid convergence is obtained and the leakage power does not dramatically blow up. The Lagrange multipliers update method is modified as follows:

Algorithm 6.1: Lagrange Multipliers Update Method

1 **for** *each timing arc* $i \rightarrow j$ **do**

2 $$\lambda_{i \rightarrow j} = \lambda_{i \rightarrow j} \times = \begin{cases} \left(1 + \frac{a_{t_j} - r_{t_j}}{T}\right)^{+k} & a_{t_j} \geq r_{t_j} \\ \left(1 + \frac{r_{t_j} - a_{t_j}}{T}\right)^{-k} & a_{t_j} < r_{t_j} \end{cases}$$

3 **end**

4 KKT projection

The values for the exponent k were empirically obtained in several experiments performed using the benchmarks of the ISPD Contest 2012 on Discrete Gate Sizing [Ozdal et al. 2012]. The exponent k was determined to be 3 for critical timing arcs in order to emphasize the delay over the leakage power. For critical cells that have most of the critical paths passing through them, referred here as bottleneck cells, k was determined to be 10. As stated in Chapter 2, the centrality of a pin, which varies from 0 to 1, is an estimation of how many and how critical are the endpoints affected by this pin. So, the centrality was used to give an estimation if a cell is bottleneck. Therefore, it was empirically established that a cell is bottleneck if its centrality is equal or greater than 0.9. By handling bottleneck cells in a special manner, it is expected that several critical paths are quickly fixed. This exponent distribution, alongside the α scaling factor discussed earlier, mitigates the leakage power blow up during the TR-LR stage, since the effort of the LR will be more aggressive on the critical cells, specially on the bottleneck ones. Table 6.1 summarizes the exponent values used during the TR-LR stage.

Table 6.1: Exponent values distribution used to adjust the Lagrange multipliers during the TR-LR stage.

Arcs	**Exponent (k) value**
Non-Critical Arcs	1
Non-Bottleneck Critical Arcs	3
Bottleneck Critical Arcs	10

Algorithm 6.2: LDP Solver used in TR-LR

```
1  set initial value of λs to 1
2  update λs
3  update Lambda-Delay Sensitivities
4  γ ← (−(min(0, worstSlack))/T + 1)
5  iter ← 0
6  while TNS > 0.1 × T and iter < 50 do
7      solveLRS(γ)
8      update timing (incremental STA)
9      γ ← (−(min(0, worstSlack))/T + 1)
10     update λs
11     update Lambda-Delay Sensitivities
12     iter ← iter + 1
13 end
14 store solution
15 update timing (incremental STA)
```

The Lagrangian relaxation subproblem is solved using the Algorithm 5.5 presented in the previous chapter. This method was adapted to consider the new LRS for-

mulation presented in Equation 6.2. Therefore, for each gate, the algorithm evaluates all its cell options and selects the version which locally minimizes the sum of timing cost and leakage cost multiplied by the scaling factor. The TR-LR execution finishes when the TNS is less than 10% of the clock period, such as in [Sharma et al. 2017], or when the number of iterations exceeds 50.

6.3.2 Power Reduction Lagrangian Relaxation

In [Reimann, Sze and Reis 2016], the proposed LR-based sizer for power and area optimization was inserted after a timing optimization step in an industrial flow. In a similar way, the TR-LR could be treated in this work as the timing optimization step, while the PR-LR performs the power optimization. As observed in [Reimann, Sze and Reis 2016], when an LR-based sizer is used in a pre-optimized design, it is necessary to provide a set of initial Lagrange multipliers that resemble the timing quality of the input solution, so that an incremental optimization is enabled. Otherwise, the TNS degrades during the initial iterations and several iterations are needed to fix it instead of focusing on power optimization. It was observed in this work that setting all Lagrange multipliers to the value 1 results in a similar phenomenon. As shown in subsection 4.2.3, this problem is solved in [Reimann, Sze and Reis 2016] by running several LR iterations in order to estimate the initial values of the Lagrange multipliers. In this work, however, it is not necessary, since the TR-LR already provides a set of Lagrange multipliers that reflect the timing quality of the timing optimized solution.

The LDP solver used in the PR-LR is presented in Algorithm 6.3. As stated in the last paragraph, the initial values of the Lagrange multipliers are the values computed in the last iteration of the TR-LR. Thus, in each iteration of the LDP solver, the Lagrange multipliers are updated using the method 6.1 presented earlier. However, it is used a different exponent distribution, so that the leakage power is quickly reduced. In order to achieve a rapid convergence without harming too much the timing of the design, it is used the values presented in Table 6.2. Like the exponent distribution used in TR-LR, the exponent values used during PR-LR were found empirically.

As in TR-LR, the Lagrangian relaxation subproblem is solved using the Algorithm 5.5 adapted to consider the new formulation presented in Equation 6.2. However, it is applied a cell option candidate filtering strategy to reduce the set of cells evaluated for each gate. Experiments performed in this work showed that it is sufficient to evaluate only the

Table 6.2: Exponent values distribution used to adjust the Lagrange multipliers during the PR-LR stage.

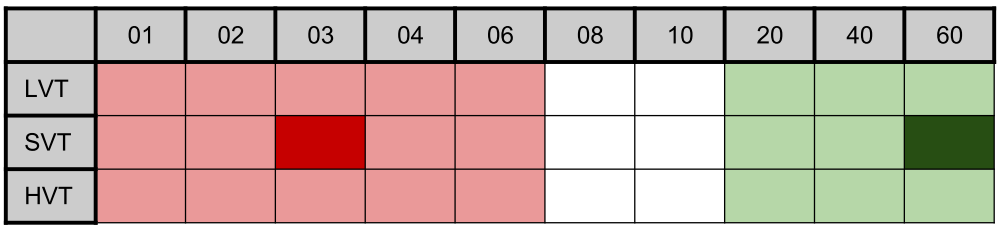
Arcs	**Exponent (k) value**
Non-Critical Arcs	5
Non-Bottleneck Critical Arcs	1
Bottleneck Critical Arcs	3

Algorithm 6.3: LDP Solver used in PR-LR

```
1  bestCost ← +∞
2  γ ← (−(min(0, worstSlack))/T + 1)
3  repeat
4      solveLRS(gamma)
5      update timing (incremental STA)
6      γ ← (−(min(0, worstSlack))/T + 1)
7      cost ← γ × total_leakage
8      if cost < bestCost and γ < 1.1 then
9          store solution
10         bestCost ← cost
11     end
12     update λs
13     update Lambda-Delay Sensitivities
14 until power improvement < 0.1%;
15 restore best solution found
16 update timing (incremental STA)
```

neighboring options of the current cell option, that is, the next two bigger sizes, the next smaller sizes, varying the V_{th} between LVT, SVT and HVT for each size and, also, options with the same size, but different V_{th}. This implies that the number of candidates evaluated depends on the current implementation option. For instance, consider the Figure 6.2. If the current cell option of a given gate is the darker red cell, the cell option candidates that will be evaluated are its neighboring cells, that is, the lighter red ones. So, in this case, 14 candidates would be evaluated. But, if the current cell option is the darker green cell, 8 candidates, that correspond to the lighter green cells, would be evaluated.

Figure 6.2: Cell option candidate filtering.



Source: from author (2018)

In each iteration of the LDP solver, the global solution found is considered only if

it has smaller leakage and its TNS is less than 10% of the clock period. The execution of the PR-LR finishes when the improvement in the leakage power is less than 0.1%.

6.3.3 Average Delay per Leakage Scaling Factor Calculation

For each logic function available in the cell library, its average delay per leakage factor is calculated using the method presented in the Algorithm 6.4. For each cell option of a given logic funtion f, the delay of its timing arcs is computed assuming an output load four times greater than its input load. Considering a timing transition of a timing arc, the delay is obtained by varying the input slew until it is approximately equal to the output slew. When this condition is met, the algorithm accumulates the ratio delay/leakage. After all cell options were evaluated, the average delay per leakage of f is computed by dividing the accumulated ratio/delay by the number of delay evaluations performed.

Algorithm 6.4: Average Delay per Leakage Computation

```
1  referenceFanout ← 4
2  acc ← 0
3  counter ← 0
4  maxIterations ← 10
5  for each cell option c ∈ versions(f) do
6      leakage ← getLeakage(c)
7      for each arc l ∈ c do
8          cin ← getInputCapacitance(c)
9          load ← cin × referenceFanout
10         for each timing transition t do
11             iSlew ← 0
12             for iter ← 0; iter < 10; iter++ do
13                 delay ← arcDelay(l, iSlew, load)
14                 oSlew ← arcOSlew(l, iSlew, load)
15                 if iSlew ≈ oSlew then
16                     stop for loop
17                 end
18                 iSlew ← oSlew
19             end
20             acc += delay/leakage
21             counter++
22         end
23     end
24 end
25 return acc/counter
```

6.3.4 Post-LR Optimization Phases Enhancement

Although the objective was to tackle the drawbacks related to the LR core of the baseline work, the post-LR phases timing recovery and power recovery are also modified in order to improve the runtime. In this section, it is described the modifications proposed in this work.

6.3.4.1 Enhanced Timing Recovery

The timing recovery method proposed in the baseline work was modified such that it is divided into 5 steps. The first one is focused on processing the bottleneck cells. It is performed by sorting all cells in decreasing order of centrality. This way, the cells that have most of the critical paths passing through them are processed first. Whenever 4% of the total numbers of cells are visited, they are sorted again. Hence, it is avoided the sorting of cells at any time a cell change improves the TNS, which harms the runtime. The algorithm tries to swap each cell to the option with the next bigger size, keeping the same V_{th}, so that the leakage does not increase too much. Also, in order to correctly evaluate the change in timing, an incremental timing update is performed. The cell option is accepted only if the TNS is not degraded and slew and load violations are not generated. Otherwise, the change is undone. When a few endpoints with timing violations are left, it was observed that sorting the cells in decreasing order of criticality is more effective in terms of convergence. Thus, the second step starts when at most 10 endpoints must be fixed to meet timing. In this step, the cells are sorted only once in decreasing order of criticality. Each visited cell is handled in the same way as before, that is, the algorithm tries to upsize the cells. The third step comes into play when there is only one endpoint left with timing violation. For this endpoint, only the cells in its critical path are processed. If the timing is not met, the algorithm of the second step is called once more. Finally, if this flow fails, the timing recovery from [Flach et al. 2013] is executed to fix the remaining timing violations, although in the experiments carried out it never occured. The enhanced timing recovery is presented in the Algorithm 6.5.

Algorithm 6.5: Enhanced Timing Recovery

```
bottleneckCellSizer() // Alg. 6.6
if getNumCriticalEndpoints() ≤ 10 then
    criticalCellSizer() // Alg. 6.7
end
if getNumCriticalEndpoints() == 1 then
    criticalPathSizer() // Alg. 6.8
end
if getNumCriticalEndpoints() >= 1 then
    criticalCellSizer() // Alg. 6.7
end
if getNumCriticalEndpoints() >= 1 then
    originalTimingRecovery() // [Flach et al. 2013]
end
```

Algorithm 6.6: Bottleneck Cell Sizer

```
sortedGates ← sortGatesByCentrality()
numGatesToProcess ← numGates × 0.04
counter ← 0
index ← 0
while index < numCells do
    if numCriticalEndpoints ≤ 10 then
        stop while loop
    end
    counter + +
    if counter == numGatesToProcess then
        index ← 0
        sortedGates ← sortGatesByCentrality()
        counter ← 0
    end
    g ← sortedGates[index]
    if g is not upsizable then
        continue
    end
    tryNextBiggerSizeOption(g)
    index + +
end
```

Algorithm 6.7: Critical Cell Sizer

```
1 sortedGates ← sortGatesByCriticality()
2 for index ← 0; index < numCells; index++ do
3 |   g = sortedGates[index]
4 |   tryNextBiggerSizeOption(g)
5 |   if TNS == 0 then
6 |   |   stop for loop
7 |   end
8 end
```

Algorithm 6.8: Critical Path Sizer

```
1 criticalPaths ← getCriticalPaths()
2 for each criticalPath in criticalPaths do
3 |   for each gate in criticalPath do
4 |   |   tryNextBiggerSizeOption(g)
5 |   |   if TNS == 0 then
6 |   |   |   stop for loop
7 |   |   end
8 |   end
9 end
```

6.3.4.2 Enhanced Sensitivity-Based Power Recovery

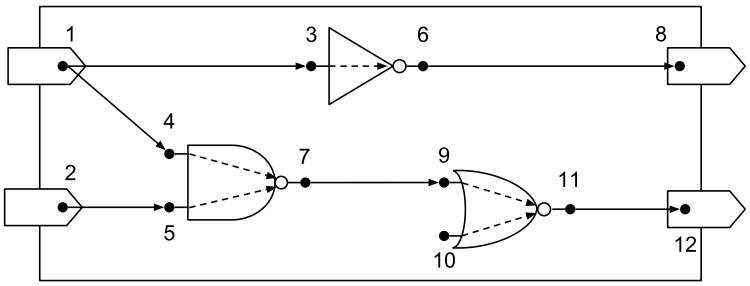
The post-LR power recovery method presented in [Flach et al. 2013] was also extended. In the new approach, the cells are sorted in increasing order of arrival time sensitivity, a metric that reflects how much a cell is likely to affect the timing given a change in the arrival time at its output pins. The arrival time sensitivity is a feature provided by the Rsyn framework [Flach et al. 2017] used in this work.

For each visited cell, the algorithm tries to increase the V_{th} as much as possible and, after that, decreases its size in the same way. Every change is accepted only if slew, load and timing violations are not generated. Algorithm 6.9 presents the new power recovery method.

The timing cost is defined as the sum of the squares of arrival times at the endpoints: $Timing = \frac{1}{2}\sum_{p \in endpoints} a_p^2$. The sensitivities are computed back-propagating the partial sensitivities from the endpoints to the start points.

For instance, the timing sensitivity at pin 1 in Figure 6.3 can be expressed using

Figure 6.3: Timing sensitivity computation.



Source: from author (2018)

the chain rule as in Equation 6.3.

$$\frac{\partial Timing}{\partial a_1} = \frac{\partial a_3}{\partial a_1}\frac{\partial a_6}{\partial a_3}\frac{\partial a_8^2}{\partial a_6} + \frac{\partial a_4}{\partial a_1}\frac{\partial a_7}{\partial a_4}\frac{\partial a_9}{\partial a_7}\frac{\partial a_{11}}{\partial a_9}\frac{\partial a_{12}^2}{\partial a_{11}} \tag{6.3}$$

Algorithm 6.9: Enhanced Sensitivity-Based Power Recovery

```
1  sortedGates ← sortGatesByArrivalTimeSensitivity()
2  for each gate g in sortedGates do
3      while V_th of g is increasable do
4          increase V_th of g
5          if TNS < 0 or load/slew violations are generated then
6              undo change
7              stop while loop
8          end
9      end
10     while g is downsizable do
11         downsize g
12         if TNS < 0 or load/slew violations are generated then
13             undo change
14             stop while loop
15         end
16     end
17 end
```

6.4 Experimental Results

The discrete gate sizing flow was implemented in C++ 11 on the Rsyn infrastructure [Flach et al. 2017] and was evaluated using the ISPD 2012 Discrete Gate Sizing Contest benchmark suite [Ozdal et al. 2012]. As shown in Table 6.3, the number of combinational gates in these benchamarks ranges from 23K to 861K. The experiments were carried out in a machine with four Intel(R) Core(TM) i7-6700 @ 3.4GHz CPUs and

32GB memory.

As mentioned in Chapter 2, the wire model used in the ISPD 2012 benchmarks is the lumped capacitance. This implies that the arrival times and the slew at the driver pin of the net is the same on the sink pins. According to [Flach et al. 2014], this simplified model is reasonable in early stages of the design flow. Also, the sizer developed in this work can be easily extended to handle more complex wire models.

In all tables presented in this chapter, the gate sizer implemented in this work is referred as new sizer, whereas the baseline work [Flach et al. 2013] and the sizer proposed in [Sharma et al. 2017] are referred as Flach and Sharma, respectively.

The leakage power results obtained are shown in Table 6.3. The results are compared with the the baseline work [Flach et al. 2013] and they are also compared with the RGS [Sharma et al. 2017], since it is the fastest gate sizing tool that can be found in literature. The results show that the new gate sizer performs only 1.1% worse in leakage power, on average, than the baseline sizer, while it performs 0.4% better, on average, than [Sharma et al. 2017]. Also, by analyzing the leakage power obtained for the netcard_fast circuit, which is the largest circuit of the benchmark suite, the sizer developed in this work marginally outperforms the results obtained in [Flach et al. 2013] and [Sharma et al. 2017] with a leakage power savings of 0.27% and 0.54%, respectively.

Table 6.3: Leakage power (W) results on ISPD 2012 benchmarks suite.

Benchmark	**# of Comb. Gates**	**Leakage Power** (W)				
		Flach	Sharma	New	New/Flach	New/Sharma
DMA_slow	23K	0.132	0.135	0.132	1.000	0.977
DMA_fast	23K	0.238	0.245	0.243	1.021	0.991
pci_bridge32_slow	30K	0.096	0.098	0.098	1.020	1.000
pci_bridge32_fast	30K	0.136	0.141	0.143	1.051	1.014
des_perf_slow	102K	0.570	0.583	0.582	1.021	0.998
des_perf_fast	102K	1.395	1.436	1.433	1.027	0.997
vga_lcd_slow	148K	0.328	0.329	0.331	1.009	1.006
vga_lcd_fast	148K	0.413	0.417	0.420	1.016	1.007
b19_slow	213K	0.564	0.569	0.565	1.001	0.992
b19_fast	213K	0.717	0.729	0.716	0.998	0.982
leon3mp_slow	540K	1.334	1.335	1.334	1.000	0.999
leon3mp_fast	540K	1.443	1.449	1.438	0.996	0.992
netcard_slow	861K	1.763	1.763	1.754	0.994	0.994
netcard_fast	861K	1.841	1.846	1.836	0.997	0.994
Avg.	-	-	-	-	**1.011**	**0.996**

As presented in Table 6.4, the new gate sizing tool requires $4.28\times$ fewer LR iterations, on average, than the baseline work. Also, during the LR phase, it evaluates $9.11\times$ fewer cell option candidates, on average, than the baseline sizer. It is also achieved a significant reduction in leakage power blow up during the LR phase. Compared to the final leakage power values presented in Table 6.3, the baseline gate sizing tool increases the

leakage power $9.65\times$, on average, while the new improved tool only increases the leakage power $2.74\times$, on average.

Table 6.4: # LR iterations, # evaluated cell candidates in LR and leakage power increase during LR for ISPD 2012 benchmarks suite.

Benchmark	**# LR Iterations**					**# Evaluated Cells in LR** ($\times 10^6$)			**Max Leakage/ Final Leakage**	
	Flach	New TR-LR	New PR-LR	New Total	Flach/ New	Flach	New	Flach/ New	Flach	New
DMA_slow	100	3	25	28	3.57	67.01	8.34	8.03	9.21	3.35
DMA_fast	100	5	28	33	3.03	67.01	10.79	6.21	6.66	3.4
pci_bridge32_slow	100	5	19	24	4.16	86.54	9.57	9.04	10.04	2.61
pci_bridge32_fast	100	13	35	48	2.08	86.54	20.92	4.13	7.72	2.29
des_perf_slow	100	5	17	22	4.54	297.03	36.78	8.07	4.58	2.0
des_perf_fast	100	7	25	32	3.12	297.03	55.46	5.35	3.17	2.0
vga_lcd_slow	100	7	13	20	5	428.65	47.34	9.05	15.35	2.66
vga_lcd_fast	100	9	14	23	4.34	428.65	58.28	7.35	13.23	3.22
b19_slow	100	5	23	28	3.57	616.75	73.38	8.40	5.20	1.53
b19_fast	100	7	30	37	2.70	616.75	100.08	6.16	5.19	1.98
leon3mp_slow	100	4	15	19	5.26	1567.02	132.08	11.86	15.27	3.74
leon3mp_fast	100	4	26	30	3.33	1567.02	183.22	8.55	15.83	4.46
netcard_slow	100	2	8	10	10	2496.75	110.16	22.66	10.46	1.76
netcard_fast	100	3	16	19	5.26	2496.75	195.72	12.75	13.26	3.46
Avg.	-	-	-	-	**4.28**	-	-	**9.11**	**9.65**	**2.74**

It was obtained an expressive reduction in the number of LR iterations for the netcard_slow benchmark. While the baseline work requires 100 iterations during the LR, this work required only 10 iterations. Thus, in terms of iterations, the speedup obtained was $10\times$. As a consequence, the new sizer evaluated $22.66\times$ less cell option candidates than the baseline sizer, which is a significantly reduction.

Considering the vga_lcd_slow benchmark, the leakage power peak during the LR phase of the baseline sizer reaches $15.35\times$ the final leakage power value, while the leakage power peak of the new sizer is $2.66\times$ the final value. This result shows the effectiveness of the strategies adopted to reduce the leakage power blow up during the initial LR iterations.

By comparing the leakage power results from Table 6.3 and the improvements presented in Table 6.4, it is possible to notice that the sizer tool developed in this work was able to tackle the drawbacks presented in the beginning of this chapter without harming too much the leakage power, since it is only 1.1% worse on average than the results obtained by [Flach et al. 2013].

The runtime and runtime breakdown are shown in Table 6.5. For some fast benchmarks, the new gate sizer spends most of the runtime in the timing recovery step. It is expected, since the slow benchmarks do not require a hard effort to remove the timing violations.

Table 6.5: Runtime breakdown (%).

Benchmark	Runtime (min)	Runtime breakdown (%)				
		Initialization	TR-LR	PR-LR	TR	PR
DMA_slow	2.30	0.50	13.45	50.79	9.00	26.26
DMA_fast	2.72	0.43	18.10	45.72	8.22	27.53
pci_bridge32_slow	1.90	0.60	31.65	48.37	2.45	16.93
pci_bridge32_fast	3.90	0.30	39.86	45.82	3.91	10.11
des_perf_slow	20.58	0.27	7.59	13.39	22.43	56.32
des_perf_fast	58.18	0.11	3.93	8.01	51.88	36.07
vga_lcd_slow	19.42	0.42	25.63	20.13	6.15	47.67
vga_lcd_fast	163.25	0.07	4.07	2.70	84.15	9.01
b19_slow	26.91	0.56	21.50	44.73	5.07	28.14
b19_fast	46.05	0.33	18.81	34.26	20.21	26.39
leon3mp_slow	123.98	0.30	14.92	21.72	3.56	59.50
leon3mp_fast	249.10	0.17	7.51	18.81	10.71	62.80
netcard_slow	87.30	0.68	12.83	21.57	0.42	64.50
netcard_fast	208.32	0.26	8.60	18.45	7.37	65.32
Avg.	-	**0.35**	**16.31**	**23.68**	**16.82**	**50.78**

By comparing the Table 6.5 and the Table 6.6 presented below, it is possible to notice that, in general, the power recovery step does not improve the leakage power significantly, although it requires an expressive runtime. This behavior is due to the slow incremental STA performed each time a cell option is evaluated. Yet, since the number of LR iterations were significantly reduced and since the number of cell option candidates evaluated were also considerably reduced, the enhanced power recovery phase is the most timing consuming of the flow, on average.

Table 6.6: Leakage power after each step.

Benchmark	Leakage Power (W) - After		
	LR	TR	PR
DMA_slow	0.133	+0.23%	-0.57%
DMA_fast	0.245	+0.30%	-1.24%
pci_bridge32_slow	0.099	+0.08%	-1.31%
pci_bridge32_fast	0.143	+2.66%	-2.29%
des_perf_slow	0.592	+0.22%	-1.90%
des_perf_fast	1.476	+0.37%	-3.28%
vga_lcd_slow	0.335	+0.10%	-1.38%
vga_lcd_fast	0.427	+1.24%	-3.01%
b19_slow	0.569	+0.02%	-0.71%
b19_fast	0.737	+0.15%	-3.00%
leon3mp_slow	1.342	+0.00%	-0.58%
leon3mp_fast	1.643	+0.05%	-12.54%
netcard_slow	1.758	+0.00%	-0.18%
netcard_fast	1.845	+0.03%	-0.52%
Avg.	-	**+0.38%**	**-2.32%**

Table 6.7 presents the runtime comparison between the new sizer and the sizers proposed in [Flach et al. 2013] and [Sharma et al. 2017] just to show how much the used infrastructure can impact the runtime. The results show that the new sizer is $4.41\times$ slower than [Flach et al. 2013], on average, and $66.33\times$ slower than [Sharma et al. 2017], on

average. It is important to highlight that the baseline sizer was implemented over an infrastructure designed specifically to address the discrete gate sizing problem, whereas this work was implemented using Rsyn, which is a generic framework designed for physical synthesis research and development, and so is not tuned for sizing algorithms. Therefore, it is expected a speedup if this work was implemented using the same infrastructure used in the baseline work. [Sharma et al. 2017] did not give details about the infrastructure used, but the sizing algorithm relies on parallelism. Besides the parallelism, the flow does not contain a post-LR power recovery like algorithm, which is, as previously demonstrated, very timing consuming. Instead, the leakage power is further reduced during the LR phase by a refinement step. Also, during the LR phase, the TNS of the circuit is kept under control, so that it does not exceed 10% of the clock period. It means that the greedy timing recovery phase, that can be very timing consuming, has to fix only some small remaining timing violations. As reported in their work, the greedy timing recovery corresponds to only 3%, on average, of the total runtime.

Table 6.7: Runtime comparison.

Benchmark	**Runtime (min)**				
	Flach	**Sharma**	**New**	**New/ Flach**	**New/ Sharma**
DMA_slow	0.79	0.07	2.30	2.91	32.85
DMA_fast	0.92	0.08	2.72	2.95	34.00
pci_bridge32_slow	0.87	0.09	1.90	2.18	21.11
pci_bridge32_fast	0.92	0.10	3.90	4.23	39.00
des_perf_slow	25.31	0.32	20.58	0.81	64.31
des_perf_fast	16.37	0.40	58.18	3.55	145.45
vga_lcd_slow	5.67	0.44	19.42	3.42	44.13
vga_lcd_fast	8.37	0.56	163.25	19.50	291.51
b19_slow	9.15	0.83	26.91	2.94	32.42
b19_fast	11.75	1.13	46.05	3.91	40.75
leon3mp_slow	38.98	2.52	123.98	3.18	49.19
leon3mp_fast	46.62	3.13	249.10	5.34	79.58
netcard_slow	34.39	2.35	87.30	2.53	37.14
netcard_fast	47.41	3.33	208.32	4.39	62.55
Avg.	-	-	-	**4.41**	**66.33**

It can be noticed that for the vga_lcd_fast benchmark, the new sizer is way slower than the other works, being $19.50\times$ slower than [Flach et al. 2013] and $291.51\times$ slower than [Sharma et al. 2017]. This could be explained by analyzing the runtime breakdown, Table 6.5, which shows that, for this benchmark, the timing recovery phase corresponds to 84.15% of the total runtime. Thus, the benefit of reducing the number of LR iterations and cell option candidates evaluated during the LR was lost due to the excessive timing recovery runtime, which was not able to quickly fix the timing violations.

Since the RGS does not rely on a post-LR power recovery phase, it is compared in

Table 6.8 the new gate sizer without the enhanced power recovey step and the RGS. The results show that, in general, the new sizer produces very close leakage power results, being only 2% worse, on average. However, for the largest benchmarks, the new sizer produces pratically the same result, since the power saved is only 0.1%. The RGS performs 13.4% better for the leon3mp_fast benchmark, since the post-LR power recovery phase of the new gate sizer improves the leakage power significantly for this benchmark. Also, without the power recovery phase, the new sizer is still slower than the RGS, but, as shown in the table below, it is now $34.94\times$ slower, on average, while with the power recovery phase it is $66.33\times$ slower, on average. So, a speedup of 47.32% is obtained by removing the power recovery.

Table 6.8: Comparison between the new gate sizer without PR and [Sharma et al. 2017].

Benchmark	**Leakage power (W)**			**Runtime (min)**		
	New sizer without PR	**Sharma**	**New/ Sharma**	**New sizer without PR**	**Sharma**	**New/ Sharma**
DMA_slow	0.133	0.135	0.985	1.69	0.07	24.14
DMA_fast	0.246	0.245	1.004	1.97	0.08	24.62
pci_bridge32_slow	0.099	0.098	1.010	1.57	0.09	17.44
pci_bridge32_fast	0.146	0.141	1.035	3.50	0.10	35
des_perf_slow	0.593	0.583	1.017	8.98	0.32	28.06
des_perf_fast	1.482	1.436	1.032	37.19	0.40	92.97
vga_lcd_slow	0.335	0.329	1.018	10.16	0.44	23.09
vga_lcd_fast	0.433	0.417	1.038	148.54	0.56	265.25
b19_slow	0.569	0.569	1.000	19.33	0.83	23.28
b19_fast	0.738	0.729	1.012	33.89	1.13	29.99
leon3mp_slow	1.342	1.335	1.005	50.21	2.52	19.92
leon3mp_fast	1.644	1.449	1.134	92.66	3.13	29.60
netcard_slow	1.758	1.763	0.997	30.99	2.35	13.18
netcard_fast	1.845	1.846	0.999	72.24	3.33	21.69
Avg.	-	-	**1.020**	-	-	**34.94**

Table 6.9 presents the true speedup obtained in this work. In this table, the new sizer is compared with an implementation of the baseline sizer implemented on Rsyn, which is refered in the table as Baseline Rsyn. In order to make a fair comparison, it has been chosen some benchmarks such that the leakage power results obtained by Baseline Rsyn match or are very close to the results reported in [Flach et al. 2013]. The results show that the new sizer is $9.77\times$ faster, on average, than the baseline sizer implemented on Rsyn. For the benchmark leon3mp_slow, the new sizer is $11.48\times$ faster. Therefore, as aforementioned, it is expected that if the new sizer was implemented on the same infrastructure of [Flach et al. 2013], its runtime results would be better than the runtimes reported by the authors of the baseline work.

Table 6.9: Comparison between the new gate sizer and an implementation of [Flach et al. 2013] on Rsyn.

Benchmark	Runtime (min)		
	New Sizer	Baseline Rsyn	Baseline Rsyn/ New Sizer
DMA_slow	24.06	2.30	10.46
pci_bridge32_slow	18.12	1.90	9.53
pci_bridge32_fast	18.89	3.90	4.84
vga_lcd_slow	213.11	19.42	10.97
b19_slow	296.71	26.91	11.02
b19_fast	435.35	46.05	9.45
leon3mp_slow	1424.25	123.98	11.48
netcard_slow	912.74	87.30	10.45
Avg.	-	-	**9.77**×

6.4.1 Runtime Reduction in LR Phase Using Cell Option Candidates Filter

In order to demonstrate the benefits of the cell option candidate filtering strategy, it is presented below the tables 6.10 and 6.11 obtained by running the sizer flow for the netcard_slow benchamrk. The first one shows the runtime of each LR iteration of the TR-LR, which does not rely on cell option candidate filtering, while the second table shows the runtime of each iteration of the PR-LR, which uses the cell option candidate filtering strategy explained in this chapter. By comparing both tables, it can be noticed that the runtime of an iteration of PR-LR is about 58% lower, on average, than an iteration of the TR-LR.

Table 6.10: Runtime (s) of TR-LR iterations for the netcard_slow benchmark.

Timing Recovery Lagrangian Relaxation	
Iteration	**Runtime (s)**
1	348.28
2	318.35
Avg.	**333.31**

Table 6.11: Runtime (s) of PR-LR iterations for the netcard_slow benchmark.

Power Reduction Lagrangian Relaxation	
Iteration	**Runtime (s)**
1	157.35
2	154.74
3	143.70
4	130.67
5	131.17
6	130.38
7	133.05
8	130.22
Avg.	**138.91**

In Table 6.12, it is provided the leakage power results obtained in the LR phase without the use of the cell option candidates filter and with the use of the filter. Compared

to the LR phase without filter, the LR phase with filter produced, in general, very similar leakage power results, being, on average, 66.7% faster.

Table 6.12: Comparison between the LR phase without cell option candidate filter and the LR phase with cell option candidate filter.

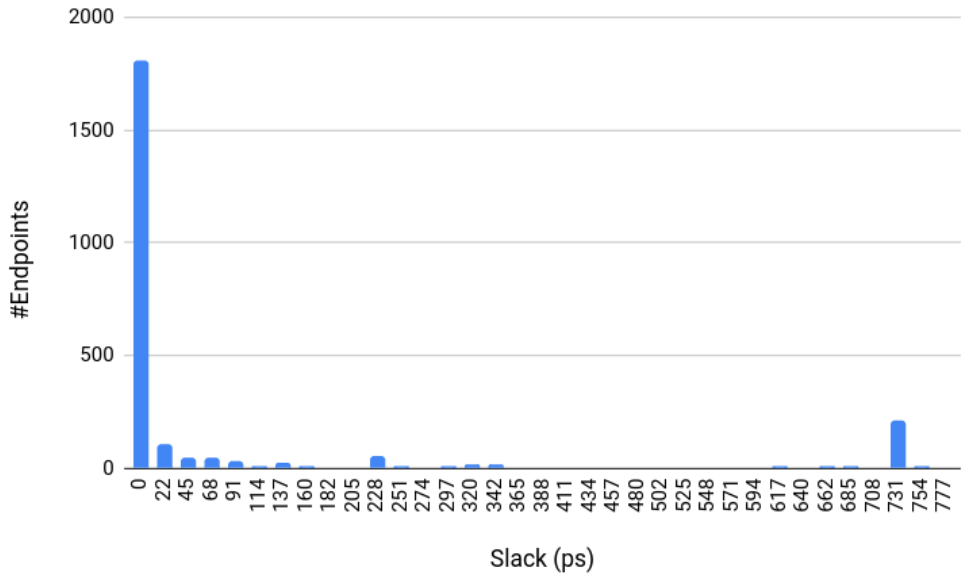
Benchmark	Leakage power (W) after LR			Runtime (min) of LR phase		
	Without filter	With filter	With filter/ Without filter	Without filter	With filter	Without filter/ With filter
DMA_slow	0.133	0.133	1.000	2.31	1.47	1.571
DMA_fast	0.246	0.245	0.995	2.83	1.73	1.635
pci_bridge32_slow	0.099	0.099	1.000	2.43	1.52	1.598
pci_bridge32_fast	0.142	0.143	1.007	5.02	3.34	1.502
des_perf_slow	0.591	0.592	1.001	7.08	4.31	1.642
des_perf_fast	1.484	1.476	0.994	10.17	6.94	1.465
vga_lcd_slow	0.332	0.335	1.009	17.35	8.88	1.953
vga_lcd_fast	0.428	0.427	0.997	16.69	11.05	1.510
b19_slow	0.569	0.569	1.000	33.79	17.82	1.896
b19_fast	0.743	0.737	0.991	44.94	24.43	1.839
leon3mp_slow	1.338	1.342	1.002	77.18	45.42	1.699
leon3mp_fast	1.452	1.643	1.131	99.23	65.56	1.513
netcard_slow	1.754	1.758	1.002	54.89	30.03	1.827
netcard_fast	1.837	1.845	1.004	95.43	56.35	1.693
Avg.	-	-	**1.009**	-	-	**1.667**

The biggest discrepancy occured for the leon3mp_fast benchmark. So, for this benchmark, the cell option filtering strategy adopted was not as effective as it was for the other benchmarks. Probably, it should consider more cell option candidates. The consequence of saving less power for this benchmark is that the power recovery will have to reduce more leakage power, which is a drawback, since, as previously demonstrated, the power recovery phase can be very timing consuming.

6.4.2 Slack Histogram Compression

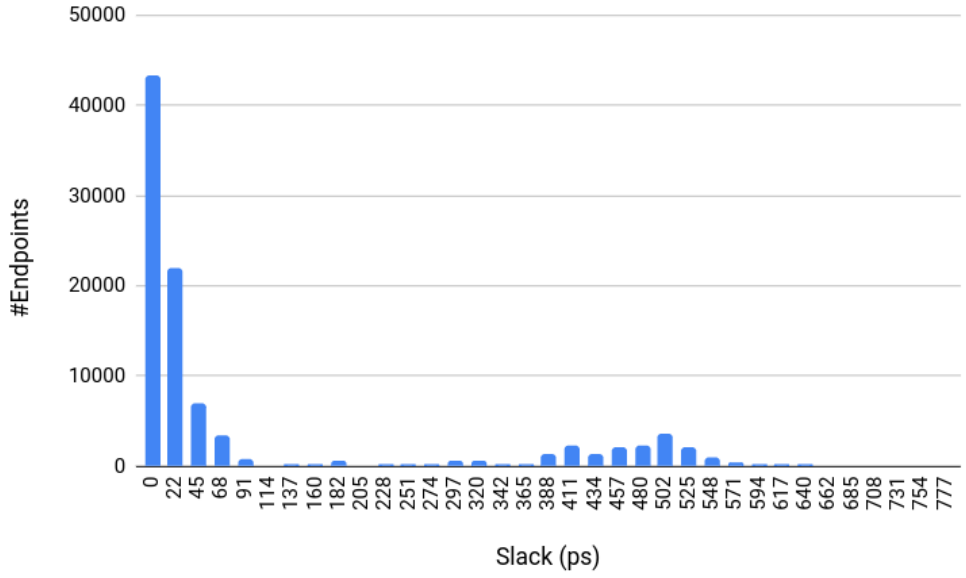
As demonstrated in [Flach 2015], the baseline work compress the slacks around the zero slack. This is desirable, since a slack distribution around the zero slack indicates that the sizer tool uses the positive slacks on non-critical paths to save power. In order to demonstrate that the sizer developed in this work keeps this property, it is presented below the slack histogram for the benchmarks DMA_fast and netcard_fast. The fast bechnmarks have been chosen since they have a tighter clock period, hence they are more difficult to optimize. These histograms show that most of the endpoints are in the zero slack bucket. Therefore, the sizer tool developed in this work is keeping the property of consuming the slacks on non-critical paths.

Figure 6.4: Slack histogram for the DMA_fast benchamrk.



Source: from author (2018)

Figure 6.5: Slack histogram for the netcard_fast benchamrk.



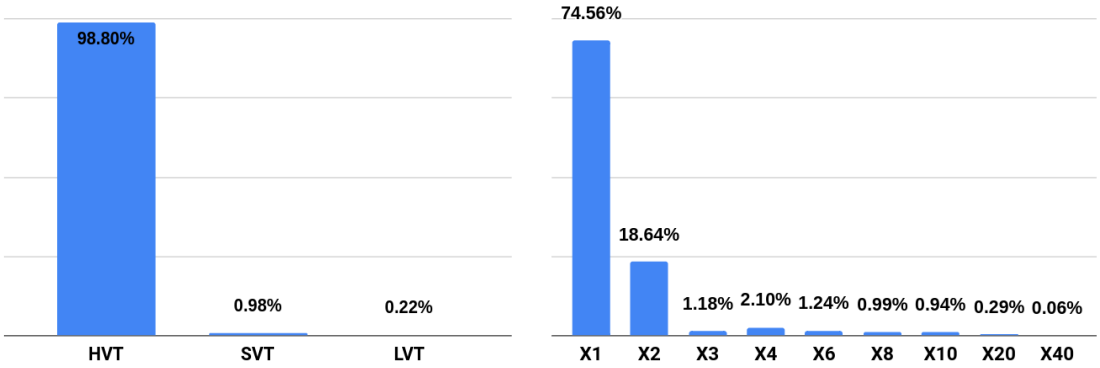
Source: from author (2018)

6.4.3 Cell Options Usage

To conclude this chapter, Figure 6.6 and Figure 6.7 show the cell option usage for the pci_bridge32_slow and pci_bridge32_fast benchmarks, respectively. It can be observed that for the pci_bridge32_fast benchmark, the sizing flow selected more cell options with greater drive strength and lower V_{th}, since the clock period is tighter than the

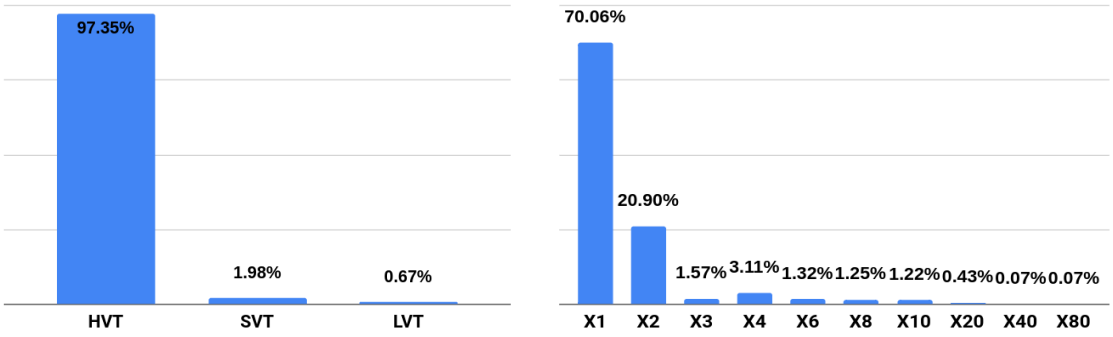
slower version of the same benchmark. Therefore, by comparing this figure with Table 6.3, it can be noticed how much the used faster cell options impacted the leakage power consumption for the fast benchmark. Regarding the extra effort needed to make the fast circuit meet timing, it can also be noticed that X80 cells, that are the largest available in the library used, were used in the fast benchmark, while for the slow benchmark the largest size used was X40.

Figure 6.6: Cell options usage by V_{th} and sizes for the pci_bridge32_slow benchmark.



Source: from author (2018)

Figure 6.7: Cell options usage by V_{th} and sizes for the pci_bridge32_fast benchmark.



Source: from author (2018)

7 CONCLUSION

Discrete gate sizing is an effective technique that is employed in the design flow of integrated circuits to peform optimizations, e.g. area and/or leakage power minimization. Due to its relevance, the ISPD 2012 and ISPD 2013 contests addressed this topic, aiming the minimization of leakage power.

State-of-the-art discrete gate sizing algorithms rely on Lagrangian relaxation due to its effectiveness to produce good quality solutions. The discrete gate sizing tool proposed in [Flach et al. 2013] has the best results in terms of leakage power, however it suffers from some drawbacks such as leakage power blow up during the LR optimization step, many LR iterations and lack of cell option candidate filtering.

Therefore, it was presented in this work a set of strategies to cope with some drawbacks of the LR based algorithm proposed in [Flach et al. 2013], which was used as the baseline work. Compared to [Flach et al. 2013], the new approach produced similar leakage power results, performing $4.28\times$ fewer LR iterations, on average, and $9.11\times$ fewer cell option candidate evaluations during LR, on average. Also, it was possible to reduce the leakage power blow up in the LR phase from $9.65\times$ the final value, on average, to $2.74\times$ the final value, on average. Furthermore, the new gate sizer without the post-LR power recovery phase still produces similar leakage power results if compared to [Sharma et al. 2017], performing slightly better for some benchmarks. In addition, it was modified the timing recovery method to improve the runtime. Finally, the post-LR power recovery phase was modified such that it uses the arrival time sensitivity metric to identify gates that have potential to be processed first so that the leakage power is reduced while meeting timing.

The drawback of the sizer developed in this work is the enhanced power recovery phase, which is, in general, very timing consuming and does not improve too much the leakage power. Other weakness that can be pointed is that a good amount of the total runtime is spent, in general, in the enhanced timing recovery phase, which fixes the remaining timing violations left by the LR phase. Therefore, although the LR phase is properly optimizing the leakage power of the circuits provided, it could perform a better control of the timing violations, such that the enhanced timing recovery can quickly makes the circuit meet timing.

7.1 Future Works

In this work, it was proposed a strategy to filter cell option candidates. Although this strategy enabled this work to achive good results, it is still simplistic. Thus, a next step is to investigate a better strategy such that less candidates are evaluated, keeping the overall quality.

It was observed that the post-LR enhanced timing recovery can be very timing consuming. Hence, in a future work, a better strategy could be developed, such that the benefit brought by reducing the number of iterations and cell options validated during the LR-core is not wasted. Or, during the LR phase, the timing violations could be kept under control, as in [Sharma et al. 2017]. Thus, the post-LR timing recovery could quickly fix small remaining timing violations. Other strategy that could be investigated is to modify the Lagrange multipliers update method, as in [Reimann, Sze and Reis 2016], such that the PR-LR stage keeps the timing quality obtained during the TR-LR.

Other point that could be explored in a futere work is a better usage of the arrival time sensitivity metric. In this work, it was only used in the post-LR power recovery to sort the cells, such that cells that do not harm too much the timing are processed first. Therefore, it could be investigated a post-LR power recovey algorithm that does not attempt to size all cells of the design, while saving a good amount of power. As demonstrated in the results section, the post-LR power recovery is, in general, very timing consuming. So, another alternative would be to remove this phase and add inside the LR phase a leakage power refinement step, like in the multi-gate sizing proposed in [Sharma et al. 2017].

Finally, this present work does not use multithreaded algorithms. In a future work, the Lagrangian subproblem solver could be parallelized, like in [Sharma et al. 2017]. Thus, by using a parallelized LR-core together with the future improvements proposed above, the sizer implemented in this work could be more competitive in terms of runtime.

REFERENCES

ALDOUS, D.; VAZIRANI, U. ldquo;go with the winners rdquo; algorithms. In: **Proceedings 35th Annual Symposium on Foundations of Computer Science**. [S.l.: s.n.], 1994. p. 492–501.

BAZARAA, M. S. **Nonlinear Programming: Theory and Algorithms**. 2nd. ed. [S.l.]: Wiley Publishing, 2003.

BERKELAAR, M. R. C. M.; JESS, J. A. G. Gate sizing in mos digital circuits with linear programming. In: **Proceedings of the European Design Automation Conference, 1990., EDAC.** [S.l.: s.n.], 1990. p. 217–221.

BHASKER, J.; CHADHA, R. **Static Timing Analysis for Nanometer Designs: A Practical Approach**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009.

BHATTACHARYA, D.; JHA, N. K. Finfets: From devices to architectures. **Advances in Electronics**, Hindawi, v. 2014, 2014.

CHAN, P. K. Algorithms for library-specific sizing of combinational logic. In: **27th ACM/IEEE Design Automation Conference**. [S.l.: s.n.], 1990. p. 353–356.

CHEN, C.-P.; CHU, C. C. N.; WONG, D. F. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 18, n. 7, p. 1014–1025, Jul 1999.

CHINNERY, D. G.; KEUTZER, K. Linear programming for sizing, vth and vdd assignment. In: **ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.** [S.l.: s.n.], 2005. p. 149–154.

CHUANG, W.; SAPATNEKAR, S. S.; HAJJ, I. N. A unified algorithm for gate sizing and clock skew optimization to minimize sequential circuit area. In: **Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)**. [S.l.: s.n.], 1993. p. 220–223.

CHURIWALA, S.; GARG, S. **Principles of VLSI RTL Design: A Practical Guide**. [S.l.]: Springer New York, 2011. (SpringerLink : Bücher).

COUDERT, O. Gate sizing: a general purpose optimization approach. In: **Proceedings ED TC European Design and Test Conference**. [S.l.: s.n.], 1996. p. 214–218. ISSN 1066-1409.

COUDERT, O. Gate sizing for constrained delay/power/area optimization. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 5, n. 4, p. 465–472, Dec 1997. ISSN 1063-8210.

DADORIA, A. K.; KHARE, K.; SINGH, R. P. A novel approach for leakage power reduction in deep submicron technologies in cmos vlsi circuits. In: **2015 International Conference on Computer, Communication and Control (IC4)**. [S.l.: s.n.], 2015. p. 1–6.

ELMORE, W. C. The transient response of damped linear networks with particular regard to wideband amplifiers. **Journal of Applied Physics**, p. 55–63, 1948.

FISHBURN, J. P.; DUNLOP, A. E. Tilos: A posynomial programming approach to transistor sizing. In: **Int. Conference on Computer Aided Design**. Las Vegas, Nevada - USA: [s.n.], 1985. p. 326–328.

FLACH, G. et al. Drive strength aware cell movement techniques for timing driven placement. In: **ISPD**. New York, NY, USA: ACM, 2016. (ISPD '16), p. 73–80. ISBN 978-1-4503-4039-7.

FLACH, G. et al. Rsyn: An extensible physical synthesis framework. In: **ISPD**. New York, NY, USA: ACM, 2017. (ISPD '17), p. 33–40. ISBN 978-1-4503-4696-2.

FLACH, G. et al. Simultaneous gate sizing and vth assignment using lagrangian relaxation and delay sensitivities. In: **2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2013. p. 84–89.

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

FLACH, G. A. **Discrete Gate Sizing and Timing-Driven Detailed Placement for the Design of Digital Circuits**. Thesis (PhD) — UFRGS, Porto Alegre, Brasil, 2015.

GEREZ, S. H. **Algorithms for VLSI Design Automation**. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 1999.

GüNTZEL, J. L. A. **Functional Timing Analysis of VLSI Circuits Containing Complex Gates**. Thesis (PhD) — UFRGS, Porto Alegre, Brasil, 2000.

HELD, S. Gate sizing for large cell-based designs. In: **2009 Design, Automation Test in Europe Conference Exhibition**. [S.l.: s.n.], 2009. p. 827–832. ISSN 1530-1591.

HENTSCHKE, R.; JOHANN, M.; REIS, R. New trends and technologies in computer-aided learning for computer-aided design: Ifip tc10 working conference: Edutech 2005, october 20–21, perth, australia. In: ____. Boston, MA: Springer US, 2005. chp. Blue Macaw: A Didactic Placement Tool Using Simulated Annealing, p. 37–47.

HITCHCOCK, R. B.; SMITH, G. L.; CHENG, D. D. Timing analysis of computer hardware. **IBM Journal of Research and Development**, v. 26, n. 1, p. 100–105, Jan 1982. ISSN 0018-8646.

HU, J. et al. Sensitivity-guided metaheuristics for accurate discrete gate sizing. In: **2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2012. p. 233–239.

HUANG, Y.-L.; HU, J.; SHI, W. Lagrangian relaxation for gate implementation selection. In: **Proceedings of the 2011 International Symposium on Physical Design**. New York, NY, USA: ACM, 2011. (ISPD '11), p. 167–174.

INTEL. **Intel Microprocessor Quick Reference Guide**. 2008. Available from Internet: <http://www.intel.com/pressroom/kits/quickreffam.htm#i486>. Accessed in: 15th July 2017.

KAHNG, A. et al. **VLSI Physical Design: From Graph Partitioning to Timing Closure**. 1. ed. [S.l.]: Springer Netherlands, 2011.

KERNIGHAN, B.; LIN, S. An Efficient Heuristic Procedure for Partitioning Graphs. **The Bell Systems Technical Journal**, v. 49, n. 2, 1970. Disponível em: https://www.cs.utexas.edu/ pingali/CS395T/2009fa/papers/kl.pdf. Acessado em: 4 de abril de 2016.

LAVAGNO, L.; SCHEFFER, L.; MARTIN, G. **EDA for IC Implementation, Circuit Design, and Process Technology**. [S.l.]: CRC Press, 2006. (Electronic Design Automation for Integrated Circuits Hdbk).

LEE, J.; GUPTA, P. Discrete gate sizing. <https://vlsicad.ucsd.edu/SIZING/ref/from_JohnLee/doc.pdf>, last accessed on 2017-07-17.

LI, L. et al. An efficient algorithm for library-based cell-type selection in high-performance low-power designs. In: **2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2012. p. 226–232.

LI, W. N. Strongly np-hard discrete gate sizing problems. In: **Computer Design: VLSI in Computers and Processors, 1993. ICCD '93. Proceedings., 1993 IEEE International Conference on**. [S.l.: s.n.], 1993. p. 468–471.

LIU, Y.; HU, J. A new algorithm for simultaneous gate sizing and threshold voltage assignment. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 29, n. 2, p. 223–234, Feb 2010.

LIVRAMENTO, V. S. et al. Fast and efficient lagrangian relaxation-based discrete gate sizing. In: **2013 Design, Automation Test in Europe Conference Exhibition (DATE)**. [S.l.: s.n.], 2013. p. 1855–1860.

OZDAL, M. M. et al. The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite. In: **ISPD 2012**. Napa, CA, EUA: [s.n.], 2012. p. 161–164.

OZDAL, M. M. et al. An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest. In: **ISPD 2013**. The Ridge Tahoe, Stateline, NV, EUA: [s.n.], 2013. p. 168–170.

OZDAL, M. M.; BURNS, S.; HU, J. Gate sizing and device technology selection algorithms for high-performance industrial designs. In: **2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2011. p. 724–731.

OZDAL, M. M.; BURNS, S.; HU, J. Algorithms for gate sizing and device parameter selection for high-performance designs. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 31, n. 10, p. 1558–1571, Oct 2012.

POSSER, G. et al. Gate sizing using geometric programming. In: **2011 IEEE Second Latin American Symposium on Circuits and Systems (LASCAS)**. [S.l.: s.n.], 2011. p. 1–4.

PRAKASH, M. **Library Characterization and Static Timing Analysis**. Dissertation (Master) — University of Southern California, Los Angeles, USA, 2007.

RABAEY, J.; CHANDRAKASAN, A.; NIKOLIC, B. **Digital integrated circuits: a design perspective**. 2. ed. Upper Saddle River, New Jersey: Pearson Education, 2003. (Prentice Hall electronics and VLSI series).

RABAEY, J. M. **Digital Integrated Circuits: A Design Perspective**. [S.l.]: Prentice-Hall, Inc., 2002.

RAHMAN, M.; SECHEN, C. Post-synthesis leakage power minimization. In: **2012 Design, Automation Test in Europe Conference Exhibition (DATE)**. [S.l.: s.n.], 2012. p. 99–104.

RAHMAN, M.; TENNAKOON, H.; SECHEN, C. Power reduction via near-optimal library-based cell-size selection. In: **2011 Design, Automation Test in Europe**. [S.l.: s.n.], 2011. p. 1–4.

REIMANN, T. et al. Simultaneous gate sizing and vt assignment using fanin/fanout ratio and simulated annealing. In: **2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)**. [S.l.: s.n.], 2013. p. 2549–2552.

REIMANN, T.; SZE, C. C.; REIS, R. Challenges of cell selection algorithms in industrial high performance microprocessor designs. **Integration, the VLSI Journal**, v. 52, p. 347 – 354, 2016. ISSN 0167-9260. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0167926015001169>.

REIMANN, T.; SZE, C. C. N.; REIS, R. Gate sizing and threshold voltage assignment for high performance microprocessor designs. In: **The 20th Asia and South Pacific Design Automation Conference**. [S.l.: s.n.], 2015. p. 214–219. ISSN 2153-6961.

REIMANN, T. J.; SZE, C. C.; REIS, R. Cell selection for high-performance designs in an industrial design flow. In: **ISPD 2016**. New York, NY, USA: ACM, 2016. (ISPD '16), p. 65–72. ISBN 978-1-4503-4039-7.

ROSSUM, M. V. Advanced nanoscale ulsi interconnects: Fundamentals and applications. In: ____. New York, NY: Springer New York, 2009. chp. MOS Device and Interconnects Scaling Physics, p. 15–38.

SAPATNEKAR, S. **Timing**. [S.l.]: Springer US, 2004. (Information Technology: Transmission, Processing & Storage).

SHARMA, A. et al. Fast lagrangian relaxation based gate sizing using multi-threading. In: **2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2015. p. 426–433.

SHARMA, A. et al. Rapid gate sizing with fewer iterations of lagrangian relaxation. In: **ICCAD 2017**. [S.l.: s.n.], 2017. p. 337–343.

SHERWANI, N. **Algorithms for VLSI Physical Design Automation**. 3. ed. Norwell, Massachusetts, USA: Kluwer Academic Publishers, 1999.

SIRICHOTIYAKUL, S. et al. Stand-by power minimization through simultaneous threshold voltage selection and circuit sizing. In: **Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)**. [S.l.: s.n.], 1999. p. 436–441.

STENZ, G. et al. Performance optimization by interacting netlist transformations and placement. **IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS**, v. 19, n. 3, p. 350–358, 7 2000.

SUTHERLAND, I.; SPROULL, B.; HARRIS, D. **Logical Effort: Designing Fast CMOS Circuits**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

TENNAKOON, H.; SECHEN, C. Gate sizing using lagrangian relaxation combined with a fast gradient-based pre-processing step. In: **IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.** [S.l.: s.n.], 2002. p. 395–402. ISSN 1092-3152.

TENNAKOON, H.; SECHEN, C. Gate sizing using lagrangian relaxation combined with a fast gradient-based pre-processing step. In: **IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.** [S.l.: s.n.], 2002. p. 395–402.

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

WANG, Z.; SECHEN, C. A new algorithm for accurate wire endpoint delay estimation. **Journal of Algorithms and Optimization**, 2014.

WESTE, N.; HARRIS, D. **CMOS VLSI Design: A Circuits and Systems Perspective**. 4th. ed. USA: Addison-Wesley Publishing Company, 2010.

YELLA, A. K.; SECHEN, C. Improved lagrangian relaxation-based gate size and vt assignment for very large circuits. In: **2017 1st Conference on PhD Research in Microelectronics and Electronics Latin America (PRIME-LA)**. [S.l.: s.n.], 2017. p. 1–4.