Power dissipation and timing in CMOS circuits

Jacobs, E.T.A.F.

Published: 01/01/2001

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 13. Apr. 2018
Power Dissipation and Timing in CMOS Circuits
Power Dissipation and Timing in CMOS Circuits

PROEFSCHRIFT

der verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof. dr. M. Rem, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 9 april 2001 om 16.00 uur

door

Etienne Theodorus Arnoldus Franciscus Jacobs

geboren te Tilburg
Dit proefschrift is goedgekeurd door de promotoren:

prof.Dr.-Ing. J.A.G. Jess
en
prof.dr.ir. R.J.H.M. Otten

© Copyright 2001 E.T.A.F. Jacobs
All rights reserved. No part of this publication may be reproduced, stored in
a retrieval system, or transmitted, in any form or by any means, electronic,
mechanical, photocopying, recording, or otherwise, without the prior written
permission from the copyright owner.

Druk: Universiteitsdrukkerij Technische Universiteit Eindhoven

CIP–DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Jacobs, Etienne T.A.F.

Power dissipation and timing in CMOS circuits / by Etienne T.A.F. Jacobs. –
ISBN 90–386–1820–4
NUGI 832
Trefw.: logische schakelingen ; CAD / stochastische analyse / grote
geïntegreerde schakelingen ; CAD / digitale systemen ; CAD / CMOS–schakelingen.
Subject headings: circuit optimisation / circuit analysis computing / delay
estimation / statistical analysis / logic CAD / CMOS digital integrated
circuits.
Summary

Since the introduction of the first digital integrated circuits, we have seen a shift in focus from first minimizing the area of the digital integrated circuit, to optimizing or delivering a certain timing. Of lately, minimizing the power dissipation of digital circuits has also become an important objective, next to delivering performance. Low power dissipation is important in portable battery operated devices, to lengthen the time the application will run on a single battery charge or to limit the weight of the battery pack. Also in other consumer electronics applications and microprocessors, power dissipation is an issue since increased power dissipation leads to more expensive power supplies or costly cooling measures.

In this thesis we discuss several aspects pertaining to power dissipation and performance of digital integrated circuits. More specifically, we limit ourselves to standard CMOS logic. Though both power dissipation and performance are to a large extent determined earlier in the design process, we focus on power and timing problems in logic synthesis. This choice is given by the fact that during logic synthesis several problems exist regarding predictability of both power dissipation and circuit speed. These problems during logic synthesis in turn make predicting power dissipation and circuit speed at higher abstraction levels difficult.

We will discuss the propagation of glitches and the modeling of dynamic and short–circuit power dissipation. This work is at the basis of the remainder of this thesis. We also briefly discuss several mathematical programming methods. These include: linear programming, integer linear programming, geometric programming and general nonlinear programming. These mathematical programs are the tools by which we solve the synthesis problems addressed in this thesis.

The first synthesis task we consider is gate sizing. Gate sizing is the problem of optimizing for an objective and thereby assigning a sizing factor (drive strength) to each gate in the circuit to realize that objective. We use gate sizing both as a means of maximizing circuit speed (performance) as well as of minimizing power subject to constraints on the circuit speed. We formulate all elements of the gate sizing problem: gate propagation delay as a function of the sizing factor, timing relations on the paths of the circuit and the contribution of dynamic and short–circuit power dissipation to the objective function. We also show that in some cases the in principal nonlinear relations can be approximated by a set of piece–wise–linear constraints. We investigate solving the gate sizing formulations through several mathematical solving methods: linear programming, general nonlinear programming and geometric programming. We also discuss a method to limit the switching activity in a circuit through removing glitches based on a combination of buffer insertion and gate sizing.
We investigate the problem of determining an upper bound on the maximum power dissipation of CMOS circuits. This problem is of direct relevance to the design of power and ground lines. We formulate an Integer Linear Program (ILP) to determine the maximum power dissipation of a circuit. As such an ILP is difficult to solve we relax this formulation to a linear program. We can solve the resulting linear program. This results in an overestimation of the maximum power dissipation. So the result is an upper bound maximum power estimate. We demonstrate our method on several benchmark circuits.

We also investigate the problem of determining a lower bound on the maximum power dissipation to complement the upper bound estimation approach. A lower bound estimate is important for two reasons: firstly, to determine the range in which the actual maximum power estimate exists, and secondly to determine the input vector pair that realizes the lower bound maximum power estimate. We formulate the problem of determining a lower bound maximum power estimate as a linear objective function subject to linear and nonlinear constraints. This gives a general nonlinear programming problem. We solve this general nonlinear program using a standard solver package LANCELOT. We also describe how to cast the problem in such a way that LANCELOT is able to solve the problem efficiently. As LANCELOT may terminate in a local optimum instead of the global optimum, which is a problem inherent to nonlinear programming, this leads to an underestimation. Hence this is a lower bound approach. We describe how to reconstruct the input vector pair from the solution of the nonlinear program, which consists of reals.

The final problem we address is static timing analysis. Usually static timing analysis of a circuit is performed based on a best–case, typical, and worst–case scenario. While the best and typical cases are usually too optimistic about circuit performance, the worst–case scenario is usually overly pessimistic. We introduce a method of statistical static timing analysis where we assume normal distributions for the gate delays. We have chosen this statistical method, because many physical sources of delay variation are inherently statistical in nature. Our method is on the one hand more accurate than the traditional timing model, while on the other hand it can be evaluated fast enough for repeated use during logic synthesis. We show the applicability of our statistical static timing analysis approach for use during logic synthesis by incorporating this statistical timing model in a gate sizing application.
Samenvatting

Sinds de introductie van de eerste digitale geïntegreerde schakelingen, hebben we een verschuiving gezien van eerst het minimalizeren van het oppervlak van de digitale schakeling naar het optimalizeren of garanderen van een bepaalde snelheid van de schakeling. Recentelijk is ook het minimalizeren van de vermogensdissipatie van digitale schakelingen een belangrijke doelstelling geworden, naast de snelheid van de schakeling. Een lage vermogensdissipatie is belangrijk voor draagbare apparaten en apparaten die op batterijen werken teneinde de tijd die een apparaat op een enkele batterijlading werkt te vergroten of om het gewicht van de batterijen te beperken. Ook in andere applicaties op het terrein van de consumentenelektronica en microprocessors is vermogensdissipatie belangrijk, omdat meer vermogensdissipatie leidt tot duurdere voedingen of dure koelmaatregelen.

In dit proefschrift bespreken we verschillende aspecten die te maken hebben met de vermogensdissipatie en snelheid van digitale geïntegreerde schakelingen. We beperken ons meer specifiek tot standaard CMOS schakelingen. Hoewel zowel vermogensdissipatie als snelheid van de schakeling voor een groot gedeelte eerder in het ontwerpproces bepaald worden, richten we ons op problemen rond vermogensdissipatie en snelheid van digitale schakelingen tijdens logische synthese. Deze keuze is ingegeven door het feit dat tijdens logische synthese verschillende problemen betreffende de voorspelbaarheid en controleerbaarheid van vermogensdissipatie en snelheid van de schakeling bestaan. Deze problemen tijdens logische synthese maken op hun beurt de voorspelbaarheid en controleerbaarheid van vermogensdissipatie en snelheid van de schakeling op hogere niveaus van abstractie moeilijk.

We zullen de propagatie van glitches en de modellering van dynamische vermogensdissipatie en de kortsluitingsvermogensdissipatie bespreken. Dit werk ligt aan de basis van de rest van dit proefschrift. We zullen ook beknapt verschillende mathematische programmeringsmethoden bespreken. Deze zijn: lineair programmeren, lineair programmeren met gehele getallen, geometrisch programmeren en algemeen niet-lineair programmeren. Deze mathematische programmeringsmethoden zijn de gereedschappen met welke we de synthese problemen van dit proefschrift oplossen.

Het eerste synthese probleem dat we beschouwen is gate sizing. Gate sizing is het probleem van het optimalizeren naar een bepaald doel, en het daarbij toekennen van een schalingsfactor (sterkte van de gate) aan iedere gate in de schakeling teneinde dat doel te realiseren. We gebruiken gate sizing zowel voor het maximalizeren van de snelheid van de schakeling als voor het minimalizeren van de vermogensdissipatie onder de voorwaarde van het
realiseren van een bepaalde snelheid voor de schakeling. We formuleren alle onderdelen van het gate sizing probleem: de vertraging van de logische poort als een functie van de schalingsfactor, de relaties voor de vertraging op de paden in de logische schakeling en de bijdrage van de dynamische vermogensdissipatie en vermogensdissipatie door kortsluiting aan de functie die het doel van de optimalisatie beschrijft. We laten ook zien dat in sommige gevallen de in principe niet–lineaire relaties kunnen worden benaderd met een verzameling stuksgewijs lineaire voorwaarden. We onderzoeken het oplossen van de gate sizing formuleringen met verschillende mathematische programmeringsmethoden: lineair programmeren, niet–lineair programmeren en geometrisch programmeren. We bediscussiëren ook een methode om de transitiedichtheid te beperken door het onderdrukken van glitches gebaseerd op een combinatie van de insertie van buffers en gate sizing.

We onderzoeken het probleem van het bepalen van een bovengrens aan de maximale vermogensdissipatie van een CMOS schakeling. Dit probleem is van belang voor het ontwerpen van voedings– en aarde–lijnen. We formuleren een lineair program met gehele getallen (ILP) om de maximale vermogensdissipatie van een schakeling te bepalen. Aangezien zo'n ILP moeilijk is op te lossen, relaxeren we deze formulering tot een lineair program. Dit resultert in een overschatting van de maximale vermogensdissipatie. Dus het resultaat is een bovengrens aan de maximale vermogensdissipatie. We demonstreren onze methode op verschillende schakelingen.

Ook onderzoeken we het probleem van het bepalen van een ondergrens aan de maximale vermogensdissipatie teneinde de aanpak die een bovengrens oplevert te completeren. Een ondergrens aan de vermogensdissipatie is belangrijk om twee redenen: ten eerste om het bereik waarin de daadwerkelijke maximale vermogensdissipatie zich kan bevinden te bepalen en ten tweede om het ingangsvectorpaar dat de schatting van de maximale vermogensdissipatie realiseert te bepalen. We formuleren het probleem van het bepalen van een ondergrens aan de maximale vermogensdissipatie als een lineaire functie onder begrenzende voorwaarden die zowel lineair als niet–lineair zijn. Dit levert een niet–lineair programmeringsprobleem op. We lossen dit niet–lineaire programmeringsprobleem middels het standaard pakket LANCELOT op. We beschrijven ook hoe we het probleem in een dusdanige vorm kunnen gieten dat het efficiënt door LANCELOT opgelost kan worden. Omdat LANCELOT kan eindigen in een lokaal optimum in plaats van een globaal optimum, welk inherent is aan niet–lineaire programmeringsproblemen, leidt dit tot een onderschatting. Aldus levert deze aanpak een ondergrens. We beschrijven hoe het ingangsvectorpaar is te reconstrueren uit de oplossing van het niet–lineaire programmeringsprobleem, welke bestaat uit reële getallen.

Het laatste probleem dat we beschouwen is statische vertragingsanalyse. Normaal gesproken wordt statische vertragingsanalyse uitgevoerd op basis van scenario's voor het beste, typische en slechtste geval. Terwijl de analyses voor het beste en typische geval meestal te optimistisch zijn over de snelheid
van de schakeling, is het slechtste geval meestal te pessimistisch. We introduceren een methode van statistische statische vertraginganalyse waarbij we een normaal gedistribueerde verdeling voor de vertraging van de logische poorten aannemen. We hebben deze statistische methode gekozen, omdat veel fysische oorzaken van variatie in vertraging inherent statistisch zijn. Onze methode is aan de ene kant accurater dan het traditionele model, terwijl het aan de andere kant snel genoeg geëvalueerd kan worden om herhaald gebruikt te worden tijdens logische synthese. We tonen de toepasbaarheid van onze statistische statische vertraginganalyse voor gebruik tijdens logische synthese aan door toepassing in een gate sizing applicatie.
Power Dissipation and Timing in CMOS Circuits
## Contents

Summary ................................................................. v

Samenvatting ....................................................... vii

Contents .............................................................. xi

Acknowledgements .................................................. xv

1 CMOS Power Considerations ................................. 1
   1.1 Power dissipation and integrated circuit design .... 1
   1.2 Power savings at all levels of design abstraction ... 3
   1.3 Challenges in power modeling and optimization ... 6
   1.4 Overview of this thesis ................................. 7

2 Preliminaries ..................................................... 9
   2.1 Introduction ............................................... 9
   2.2 Delay modeling in CMOS circuits .................... 10
      2.2.1 Propagation delay and logic levels ............ 10
      2.2.2 Pin–to–pin delays ............................... 11
      2.2.3 Input and output slopes ......................... 12
      2.2.4 Glitches ........................................... 12
      2.2.5 Filtering effect of CMOS gates ................. 13
   2.3 Sources of power dissipation in CMOS circuits .... 20
      2.3.1 Dynamic power dissipation ..................... 20
      2.3.2 Short–circuit power dissipation .............. 21
   2.4 Glitch propagation ...................................... 21
   2.5 Transition density calculation ....................... 22
      2.5.1 Zero delay model ................................ 24
      2.5.2 General delay model ............................ 24
      2.5.3 Unit delay model ................................ 24
      2.5.4 Inertial delay model ........................... 24
   2.6 Linear programming .................................. 24
      2.6.1 Simplex solvers .................................. 26
      2.6.2 Interior point solvers ........................... 27
      2.6.3 Cholesky factorization .......................... 28
      2.6.4 Comparison of different LP solvers .......... 29
Power Dissipation and Timing of CMOS Circuits

2.6.5 (Mixed) integer linear programming .......... 30
2.7 Nonlinear programming .......................... 31
2.7.1 LANCELOT ................................. 31
2.7.2 Augmented Lagrange method .................. 31
2.7.3 Derivatives ................................ 32
2.8 Geometric programming .......................... 33
2.9 Quadratic programming ........................... 35

3 Gate Sizing ........................................ 37
3.1 Introduction ...................................... 37
3.2 Related Work ..................................... 38
3.3 Sizing a gate ..................................... 39
3.4 Modeling power in a gate sizing formulation ...... 42
3.5 Gate sizing ....................................... 43
3.6 Gate sizing using linear programming .......... 44
3.6.1 Linear programming formulation for gate sizing .. 44
3.6.2 Results with LP_solve .......................... 46
3.6.3 Results with PCx ............................... 48
3.6.4 Results with LOQO ............................. 50
3.7 Gate sizing using geometric programming .......... 52
3.8 Gate sizing using nonlinear programming ..... 57
3.9 Comparison of gate sizing using linear and nonlinear
programming ...................................... 62
3.10 Timing constraints for glitch removal .......... 63
3.11 Gate sizing for minimal power while removing
glitches ............................................. 64
3.12 Discussion ....................................... 67

4 Upper Bound Maximum Power ....................... 69
4.1 Introduction ...................................... 69
4.2 Related work ..................................... 69
4.3 Problem formulation .............................. 70
4.3.1 Calculating and storing the switching events .. 71
4.3.2 Power dissipation in CMOS circuits .......... 71
4.3.3 Maximum power estimation of CMOS circuits ... 73
4.4 Linear programming model ....................... 73
4.4.1 Objective function ............................ 74
4.4.2 One of four scenarios .......................... 74
4.4.3 Precedence constraints ......................... 74
## Contents

4.4.4 Inverters ........................................ 75
4.4.5 AND–gate .......................................... 75
4.4.6 OR–gate ........................................... 78
4.4.7 Complex gates ...................................... 78
4.5 Example .................................................. 79
4.6 Experimental results ................................. 81
4.7 Extension to maximum current estimation ....... 83
4.8 Discussion ............................................... 85

### 5 Lower Bound Maximum Power ............................ 87

5.1 Introduction ............................................. 87
5.2 Related work ............................................ 88
5.3 Problem formulation .................................... 89
5.3.1 Calculating and storing switching events revisited 89
5.3.2 Transformation of boolean functions into nonlinear constraints ........................................ 90
5.3.3 Objective function .................................. 90
5.3.4 Inverters .............................................. 91
5.3.5 Logic gates ........................................... 91
5.4 Example .................................................. 92
5.5 Nonlinear constrained optimization ................. 94
5.5.1 Maximum power estimation problem formulation 94
5.5.2 Solving the maximum power estimation problem 94
5.5.3 Conversion from the continuous to the Boolean domain ................................................. 95
5.6 Experiments .............................................. 95
5.7 Discussion ............................................... 97

### 6 Gate Sizing under a Statistical Delay Model ............ 99

6.1 Introduction ............................................. 99
6.2 Related work ............................................ 101
6.3 Theory of statistical calculations .................... 101
6.4 Validation of approximating the gate delay by a normal distribution .................................... 104
6.5 Validation of approximation of maximum of two normal distributed variables ....................... 107
6.6 Gate sizing using the statistical delay model ...... 114
6.7 Example .................................................. 116
6.8 Experimental results .................................... 118
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.9 Discussion</td>
<td>121</td>
</tr>
<tr>
<td>7 Concluding Remarks</td>
<td>123</td>
</tr>
<tr>
<td>7.1 Research goals and results</td>
<td>123</td>
</tr>
<tr>
<td>7.2 Suggestions for future work</td>
<td>125</td>
</tr>
<tr>
<td>A Stochastic maximum</td>
<td>129</td>
</tr>
<tr>
<td>References</td>
<td>135</td>
</tr>
<tr>
<td>Index</td>
<td>141</td>
</tr>
<tr>
<td>Biography</td>
<td>147</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to thank professor Jess for giving me the opportunity to become a researcher in his group, for giving me the freedom to choose my own directions and for his comments on the drafts of this thesis. I would also like to thank the reading committee for their valuable comments.

I would like to thank Michel Berkelaar for his constructive criticism, for providing interesting challenges and at the same time to leave me the freedom to approach and solve problems my way.

I would like to thank my roommate Jeroen Rutten for all the useful discussions we had. Explaining my work to him, usually provided me with useful insights in how to extend the work and how to better present it. I would also like to thank Koen van Eijk for always showing an interest in my work and also for providing 'symple', a tool to estimate transition densities.

I would like to thank Chandu Visweswariah for giving me the opportunity to spend time at the IBM T.J. Watson Research Center. I have learned a great deal from him about nonlinear programming and its uses in EDA. This thesis, especially the parts concerning the use of LANCELOT, would have certainly looked different without this experience.

Last but not least, I would like to thank my family for their continuous support.
Chapter

1 CMOS Power Considerations

1.1 Power dissipation and integrated circuit design

Since the introduction of the first integrated digital circuits, we have seen a shift in focus from first minimizing the area of the digital integrated circuit to optimizing or delivering a certain timing, and lately to minimizing the power dissipation. At first, area minimization was necessary in order to enlarge the chance for a circuit to be processed successfully. Then the focus shifted to timing minimization, or meeting very specific timing constraints in order to meet ever higher performance requirements and clock speeds. By that time, area became a less important factor since IC-fabrication processes had matured and the die area therefore was not that much of a limiting factor anymore. Currently, controlling the amount of power dissipation and minimizing the power dissipation of the integrated circuit has become an important factor, as well as ensuring other performance criteria.

Low power dissipation is important in portable and battery operated devices, to lengthen the time the application will run on a single battery charge or to limit the weight of the battery pack. However, battery operated devices are not the only application area where power dissipation is important. Also in other consumer electronics applications and microprocessors limiting power dissipation is a concern.

Power dissipation in portable and non-portable consumer electronics applications and microprocessors is important for several reasons:

- Higher dissipation of power leads to the warming up of the device. This will deteriorate the device’s performance or necessitate additional cooling measures to counteract performance degradation. These countermeasures in turn lead to higher costs and add weight to the application.
- Higher power dissipation in portable devices either leads to less active time on a single battery charge or the need for a higher capacity battery pack, which is heavier.
- There is a limit on the maximum current per pin. Since higher power dissipation means a higher supply current, higher power dissipation leads to more pins on the device.
• Higher power dissipation means higher currents going through the wires in the device. High currents can lead to electromigration of the metal in the power and ground wiring, which will shorten the device’s lifetime and impair its reliability.

In this thesis we will focus our attention to digital integrated circuits and more specifically to standard CMOS logic. Most highly integrated circuits in use today are digital circuits. Of these digital integrated circuits, most consist of standard CMOS logic. This does not mean that the ideas presented in this thesis could not be used for other styles of digital logic, but for those styles some adaptations may be necessary.

As we will make clear in this thesis the power dissipation in integrated digital CMOS circuits depends on several factors: the supply voltage, the capacitance being switched, the clock frequency and the number of transitions of a net per clock cycle. Except for perhaps the supply voltage, which is mostly determined by the chosen fabrication process generation [SIA97], all of the factors mentioned are subject to decisions made during the design process. We will now discuss these remaining factors.

Clock frequency and number of transitions of a net per clock cycle depend heavily on timing issues [Naj95]. Signals must propagate through the circuit between clocked latches within a certain time in order to achieve the desired clock frequency for the circuit. Also the number of transitions of a net per clock cycle depends very much on differences in propagation delay along different paths in the input cone of the net in question. Very accurate timing models are needed to calculate propagation delays and predict these timing related issues. However, in order to be able to effectively use such a timing model in a synthesis environment the timing model must not only be accurate, but it must also be such that propagation delays can be calculated quickly. The calculation speed is important since in the synthesis process multiple delay calculations will be necessary.

As we will see in this thesis a change in capacitance driven by a gate directly influences the power dissipation, because a change in this capacitance means that the charge transferred during a transition changes. This change in transferred charge directly impacts power dissipation. A change in capacitance driven by a gate also changes the propagation delay of the gate and this in turn influences the timing of the entire circuit. A change in propagation delay of a gate influences the delay along a path in the circuit. Differences in path propagation delays in turn lead to glitches. The number of transitions of a net per clock cycle is, at least partly, determined by the number of glitches. The sequence of events therefore leads from a change in capacitance to a changed number of transitions in a clock cycle. So the effect of a change of capacitance on power dissipation is less predictable and straightforward than
the general idea that more capacitance switched means more power dissipation. The problem becomes even greater because the capacitances, especially those of the wiring, cannot be estimated accurately before the layout is known. And a layout is not known until after the process of physical design, which follows logic synthesis.

In order to be able to deal with the available design space in the time and power domain for a given design effectively, we first have to be able to model timing and power efficiently and accurately enough at the given abstraction level. However, in order to perform synthesis, one first has to be able to estimate the propagation delay and power dissipation of a given design. Especially estimating power dissipation is not yet trivial. Existing power estimation models usually are either accurate, but not fast enough to deal with large designs, or can deal with large designs fast enough, but lack accuracy. The problems underlying this observation are mainly due to the lack of predictability of capacitance and timing at a higher abstraction level. At the level of logic synthesis in the design flow there is not a sufficiently accurate picture of the resulting layout to provide meaningful estimates of capacitances and therefore of accurate timing. The lack of accurate timing also means a lack of accuracy in the power estimate.

1.2 Power savings at all levels of design abstraction

With the growing complexity of digital integrated circuits we witness the creation of hierarchical levels of abstraction intended to structure the design process. We distinguish the following levels:

- system level synthesis
- architectural synthesis
- logic synthesis
- physical design

Figure 1.1 depicts the flow from specifying the design objective all the way to layout via the processes of system level synthesis, high level synthesis, logic synthesis and physical design. We will now describe each of the synthesis steps in more detail and will also discuss the level of accuracy of the power dissipation model required at each of these levels.
**Figure 1.1.** Overview of the main synthesis steps in the design process of a digital circuit.
**System level synthesis** is all about partitioning the specification of the design problem. This partitioning means distributing the functionality over several hardware components, for example several integrated circuits, as well as deciding which parts will be done in software and which in hardware. Important aspects in this partitioning process could be pin counts, packaging cost, the amount of flexibility required, also for future purposes, communication speeds required, cost issues depending on production volumes for the trade off between ASIC’s and general purpose components and so on.

**Architectural synthesis** takes the behavioral description of that part of the specification which is attributed to a specific part which will be realized in hardware and generates a register transfer level description, which consists of datapath and control units. The datapath consists of basic modules, such as adders, arithmetic–logic units, multipliers and register files. The data path is controlled by the control unit, which generates the control signals that that govern the datapath units. Important steps in architectural synthesis are module allocation, scheduling and binding. Module allocation determines the number and type of modules the architecture will consist of. Power trade–offs can be made here between for example the number of adders and multipliers, which each have a different power consumption, or between fast and high–power units or slower and less power–consuming units. Scheduling determines the execution time of the operations. Subsequently, binding determines on which unit a particular operation is performed.

**Logic synthesis** starts with a register transfer level description and delivers a description based on logic gates. The type of logic and the size of the library containing the set of logic gates depends on the design style. Examples of design style are full–custom design, standard–cell design and array–based design styles. Logic synthesis is generally regarded to consist of two major parts: combinational logic synthesis and sequential logic synthesis. The main difference between the two is that sequential logic synthesis also considers parts of the circuit containing memory elements, where combinational logic synthesis doesn’t.

**Physical design** performs the last steps in the synthesis process. Physical design generates the geometric patterns which define the physical layout of the integrated circuit from the gates and netlist. Physical design will translate the gate–level description via several steps, which include floorplanning, placement and routing. At this level of abstraction, extraction is performed to get the actual resistance, capacitance and induction of the circuit. A more detailed description of the steps entailing physical design can be found in [Len90].

The higher the abstraction level of the specification of the design the more choices for the implementation are still open to us. At the system level there
is still a choice in partitioning of the design, at architectural level we can still decide the complete architectural solution and at the logic level we can still decide the type of logic and the implementation in logic gates. Of course the greater the choice in implementation still open to us is, the greater gains in power implementation that can possibly be achieved.

Though higher gains in power dissipation are achievable at the higher abstraction levels of system level synthesis and architectural synthesis we focus on the domain of logic synthesis. This choice is given by the fact that though higher gains in power dissipation are achievable at the above mentioned higher levels of abstraction, those gains are only achievable if the back end of logic synthesis can be performed with a reasonable level of automation and the power dissipation is predictable and controllable at this level. In other words: the goal is not to have logic synthesis minimize power consumption but to make logic synthesis perform in a not too bad way and to deliver a circuit with a power dissipation, which can be predicted beforehand. In order to achieve this we will have to create some freedom beforehand, which might cost some more power, so that we can afterwards make corrections which improve the overall predictability.

1.3 Challenges in power modeling and optimization

Now that we have focussed our attention on logic synthesis and normal CMOS logic, we see the following problems and challenges ahead of us in estimating power dissipation and optimizing for low power at logic level:

- Estimating the average power dissipation in a circuit. This problem is relevant in determining the time the battery will last on a single charge and also the amount of heat that the IC-packaging will have to deal with. Though this problem has been addressed in numerous papers (see [Naj94a] for an overview), it is still problematic to estimate the average power accurately under a realistic delay model and for large circuits.
- Estimating the maximum power dissipation of a circuit in order to adequately design power and ground lines as well as finding the input vector transition that realizes the maximum power dissipation. The time interval in case of power estimation is typically one clock cycle.
- Estimating the maximum current through power and ground lines. This problem is related to the problem of estimating the maximum power. However, the estimation of the maximum current is not performed over an interval of a clock cycle.
- Avoiding gates which are oversized, and therefore consume too much power. This problem can be solved using gate sizing.
- A similar problem is making circuits meet their performance constraints using as little power as possible. Gate sizing can be one step in such an approach.
• The power dissipation of a circuit is not only due to useful transitions, but also for a large part due to glitches. Whether a glitch or even multiple glitches occur depends on differences in propagation delay along different paths in the circuit. These propagation delays depend on the capacitances and resistances of the circuit, which are not accurately known at this instance in the design process. This makes predicting power dissipation under a realistic delay model a very difficult task. We have to find a way to deal with this problem. This involves an accurate power dissipation model that is also easy to compute without much overhead.

• Since the power depends so heavily on accurate timing one could also try to model propagation delays in the circuit as stochastic variables. Such a statistical timing model can in future possibly be extended to a power dissipation model as well.

We will address the challenges mentioned above in the remainder of this thesis. As has been mentioned before, the problem of controlling the power dissipation of CMOS logic is for a large part dependent upon accurately modeling and controlling the timing of the circuit. This thesis will therefore address both timing and power modeling techniques. We will attack the optimization process of timing and power dissipation, which is of major concern in synthesis for low power dissipation. We model the optimization problems in the form of linear and nonlinear programming problems, which are therefore also an important part of this thesis.

1.4 Overview of this thesis

This thesis describes a number of ways of modeling timing and estimating power dissipation of CMOS gates at the logic level as well as optimizing for speed and power dissipation within these models.

In chapter 2 we discuss the traditional way of performing static delay modeling and some of the basic notions necessary for modeling power dissipation at the logic level. We also discuss several linear and nonlinear programming methods, which are used in the remaining chapters of this thesis as a basis for solving the gate sizing and power estimation problems. It is not our intention to give a complete overview of all aspects and details of linear and nonlinear programming, but to give just enough information for the reader to understand the basics and the advantages and disadvantages of each of the methods in general.

Gate sizing is the subject of discussion in chapter 3. We will look at sizing for minimal power subject to delay constraints and at sizing for minimal delay. We will use several different gate sizing formulations and discuss their
merits. We also investigate several different ways to solve the different gate sizing formulations.

Chapters 4 and 5 deal with problem of estimating the maximum power consumption of a CMOS circuit. In chapter 4 we present a method which estimates an upper bound on the maximum power by using linear programming. In chapter 5 we present a method using nonlinear programming which estimates a lower bound on the maximum power dissipation. The method is also able to calculate the input vector transition which realizes the lower bound maximum power estimate.

Chapter 6 introduces a statistical approach to the problem of static delay calculation. We are also able to use this statistical delay model during synthesis. We describe a gate sizing approach under the statistical delay model to illustrate the application as well as the applicability of a statistical approach to static delay calculation.

Finally, in chapter 7 we summarize the main contributions of this thesis. We will then present some concluding remarks regarding these ideas and their implementation. These concluding remarks will lead us to some suggestions regarding interesting topics for future research.
Chapter

2 Preliminaries

2.1 Introduction

This chapter deals with a selection of topics to give the reader some background on the basic work in the field of timing modeling, power modeling for CMOS, glitches, transition density calculation and linear and nonlinear programming. All these items are basic ingredients for the remainder of this thesis.

First we will discuss delay modeling of CMOS circuits in section 2.2 and all aspects that influence the delay of a CMOS logic gate. Then we will look into the sources of power dissipation of CMOS logic circuits in section 2.3. Section 2.3 discusses the different sources of power dissipation of CMOS logic gates. Some power is dissipated only when a gate is switched and some is dissipated constantly. In section 2.4 we take a look at how glitches are propagated in a more abstract way than in our consideration in section 2.2. Section 2.5 deals with transition density, a way to capture the number of transitions that occur within a clock cycle.

The remainder of this chapter, sections 2.6 through 2.9, deals with mathematical programming solvers for optimizing an objective function subject to constraints. Section 2.6 discusses linear programming. That is minimizing a linear objective function subject to linear equality constraints and nonnegative variables. We discuss both the simplex method as well as interior point methods to address this problem. We also discuss the advantages and disadvantages of the simplex method in contrast to interior point methods.

In section 2.7, the general nonlinear programming problem of minimizing a nonlinear objective function subject to nonlinear constraints and nonnegative variables is discussed. We also look at solving this programming problem by a standard program called LANCELOT. Geometric programming, a special case of nonlinear programming, with a theoretically more efficient solver is the topic of the discussion in section 2.8. Finally section 2.9 deals with quadratic programming, another special case of nonlinear programming. In quadratic programming an objective function, which next to linear
terms also contains quadratic terms, is solved, subject to linear constraints
and nonnegative variables. We shall see that solving a quadratic program is
in many ways very similar to solving a linear program.

2.2 Delay modeling in CMOS circuits

![Diagram of a general delay model]

**Figure 2.1.** General delay model

We assume a delay model with both pin–to–pin gate delays \( t_i \) and wire delays
\( t_{w,j} \). A pin–to–pin delay \( t_i \) is the propagation delay from arrival time \( T_i \) at
input pin \( i \) to output arrival time \( T_{out} \) at the output pin. See figure 2.1 for an
illustration. A wire delay \( t_{w,j} \) is the propagation delay from the output arrival
time \( T_{out} \) at the output pin to the arrival time \( T_{w,j} \) at the end of wire \( j \).
Different delays from each input to each output are allowed, as well as
different rise and fall times. We assume an arrival time \( T_i \) at input \( i \). We can
now calculate the arrival time at the output pin \( T_{out} \) and arrival time \( T_{w,j} \) at
the end of wire \( j \) with:

\[
T_{out} = \max_{i=1}^{n}(T_i + t_i) \tag{2.1}
\]

\[
T_{w,i} = T_{out} + t_{w,i}
\]

In contrast to the suggestion of figure 2.1 a gate or cell could have more than
one output each realizing different logic functions. In that case we can model
the delays of a gate with an instance of figure 2.1 for each output. So all delay
variables in figure 2.1 will become specific to the output under consideration
except for the input arrival times \( T_i \). We have to take care that synthesis
operations on the gate, such as gate sizing, have to be performed for each of
those instances.

2.2.1 Propagation delay and logic levels

Before we start talking about propagation delays it is important to describe
how we define logic levels and how we measure propagation delays of gates
and circuits. We will assume that $V_{dd}$ is the supply voltage of the gate. There are two logic levels, 1, also called high, and 0, also called low. The logic level is called high if the voltage level of the signal is larger than 50% of $V_{dd}$, the supply voltage of the circuit. A logic level is called low if the voltage level of the circuit is less than 50% of $V_{dd}$. We measure propagation delays from the time-instance of the input signal crossing the 50% threshold of $V_{dd}$ to the time-instance of the output signal crossing the 50% threshold of $V_{dd}$. Figure 2.2 depicts the measurement of the propagation delay, denoted by $t_p$, in case of the output of an inverter rising. This is the so-called "rising propagation delay". Similarly the "falling propagation delay" is defined. The propagation delay of the inverter is then given by the mean of the rising and falling propagation delays.

\[\text{\textbf{Figure 2.2}}. \quad \text{Voltage levels and measuring propagation delays of an inverter}\]

### 2.2.2 Pin-to-pin delays

The delay model of figure 2.1 also incorporates the possibility of dealing with different pin-to-pin propagation delays, depicted there by the propagation delays of the gate $t_i$. The propagation delays from input pins to output pins can differ due to charging paths to $V_{dd}$ or discharging paths to ground going through different transistors. Also charge sharing within the gate can effect the propagation delay. We measure the pin-to-pin propagation delays in a similar fashion as described for the inverter in section 2.2.1.
2.2.3 Input and output slopes

Rising and falling slopes at the output a of gate depend on both the capacitance that is driven by the gate and the steepness or slope of the changing signal at the inputs of the gate. In general, the faster the input change is the faster the output change is and the smaller the load driven by the gate is the faster the output change is. Multiple simultaneously changing inputs can either speed up the change when the input changes individually would have had the same logic effect, or slow down the change when their individual changes would have resulted in a different logic effect.

The rate of the input and output changes is often described as the slope of the signal. We measure the slope of the signal from the point where the rising signal passes the 10% \( V_{dd} \) mark to where it passes the 90% \( V_{dd} \) mark. For a falling signal we of course measure from 90% \( V_{dd} \) to 10% \( V_{dd} \). Figure 2.3 depicts how to measure the rise and fall times.

\[ \text{Vdd} \]
\[ 100\% \]
\[ 90\% \]
\[ \text{in} \]
\[ \text{out} \]
\[ 50\% \]
\[ 10\% \]
\[ 0\% \]
\[ t \]

**Figure 2.3.** Measuring rise and fall time of signals of an inverter

The slope is given in volt per time unit and can be calculated dividing the voltage difference of 80% \( V_{dd} \) by the rise or fall time.

2.2.4 Glitches

Glitches are spurious logic transitions in the circuit, which occur before all nets of the circuit stabilize to their new logic values. The term 'glitch' is restricted to the case when the logic level perceived by the successor gate changes. Glitches occur due to different propagation delays along different paths in the circuit. Different propagation delays in the input cone of a gate
means, that not all transitions at the input of that gate occur simultaneously. This again means that the output of that gate shows glitches.

Unfortunately glitches dissipate power, but do not contribute usefully to the function of a circuit. Furthermore, the amount of glitching at the output of a single gate depends very heavily on the propagation delays in the entire circuit.

**2.2.5 Filtering effect of CMOS gates**

CMOS gates exhibit a filtering effect. That means that if transitions at the input(s) of the gate which counter each others logic effect at the output of the gate occur closely enough in time, the output does not show a change in logic level. This filtering effect is sometimes also called the inertial delay of a gate. Looking at the gate at an electrical level, the output will show a change in output voltage, but this is not enough to cause a change in logic level perceived by subsequent gates. Figure 2.4 shows the filtering effect for a single inverter. We observe that as the input pulse width becomes smaller the output signal does not reach the 0 Volt level anymore, and crosses not even the 50% $V_{dd}$ level in the third drawing. In the latter case the gate exhibits the filtering effect.

The filtering effect experiments of figure 2.4 through 2.7 are performed using SPICE in a 1.0 micron technology with a capacitive load of 10 fF.
Figure 2.4. Filtering effect for an inverter
We can also perform this experiment for a chain of six inverters. Figure 2.5 shows the results for this experiment. All inverters are chosen such that their propagation delay is the same.

\[\text{Figure 2.5. Pulse width experiment for an inverter chain}\]

We can see in figure 2.5 that a pulse with a small width tends to die out after several stages of inverters. In the example of figure 2.5 the initial pulse isn't wide enough to cause a change in logic level at the output of the third inverter since the voltage level does not drop below 2.5 Volts. After the fifth inverter not even an electrical change in the output of the inverter is visible any more.
Now that we have demonstrated the principle of glitch propagation for an inverter chain, we naturally also want to know if the same principle holds for other logic gates. We will therefore look at a two-input NAND-gate. In order to investigate glitch propagation for a two-input NAND-gate we have to see either a pulse on one of its inputs, with the other input a stable 1, or a rising transition on one input before a falling transition occurs on the other.

We will first look at the case of glitch propagation through the two-input NAND-gate with one input equal to a stable 1. This case is similar to the case for an inverter. Actually the NAND-gate logically behaves as an inverter. Figure 2.6 depicts the propagation of glitches with several pulse widths through the NAND-gate.
Figure 2.6. Filtering effect on pulse propagation for a NAND–gate
We can see in figure 2.6 that a two-input NAND-gates also exhibits the filtering effect for pulses with a small width.

Next we will look at the case where we generate a glitch due to two opposing transitions at the inputs, which do not occur simultaneously. Figure 2.7 depicts the glitch generation experiment for the two-input NAND-gate for several different delays between the two transitions.

We can see in figure 2.7 that the two-input NAND-gate also exhibits the filtering effect, if the two transitions occur near enough in time.

We have seen that the filtering effect occurs for inverters with a pulse with a small enough width as well as for more complex gates for pulses with small enough widths and for input signals which would otherwise generate a glitch at the output, but for which the transitions are very near in time. In section 2.3 we will come to a more general understanding of the conditions under which a glitch propagates, partially propagates and is completely filtered out.
Figure 2.7. Filtering effect on pulse generation for a NAND–gate
2.3 Sources of power dissipation in CMOS circuits

There are three main sources of power dissipation in CMOS–gates:

- dynamic power dissipation
- short–circuit power dissipation
- dissipation due to leakage current

The leakage current in current CMOS–technologies is small [Sap95], compared to dynamic power dissipation and short–circuit current. The leakage current is however expected to grow as the smaller supply voltages in new technologies cause the threshold voltage to come down too. The problem of leakage current unfortunately is not one which we can effectively address in the context of logic synthesis, because its roots are in the process technology domain. We will therefore concentrate on the first two sources of power dissipation. Both dynamic power dissipation and short–circuit power dissipation occur during logic transitions. So, when the gate is stable, no power is used (except for the dissipation due to the leakage current). We will first look in more detail into dynamic power dissipation and then into short circuit power dissipation. We will assume the supply voltage $V_{dd}$ to be constant.

2.3.1 Dynamic power dissipation

The dynamic power dissipation in a single CMOS–gate is due to the loading and unloading of the capacitance driven by that gate. That capacitance consists of the internal capacitances of that gate, wire capacitances driven by that gate and the capacitances in the gate oxides of the transistors of the gates in the fanout cone. We will assume that the logic swing of a gate is equal to the supply voltage. The power dissipation in a single CMOS–gate can be calculated by the following equation.

$$P_{\text{dyn}} = \frac{1}{2} \cdot C_{\text{load}} \cdot V_{dd}^2 \cdot f \cdot D$$  \hspace{1cm} (2.2)

The variables in equation 2.2 have the following meaning:

- $P_{\text{dyn}}$ the dynamic power dissipation
- $C_{\text{load}}$ the load capacitance of the gate
- $f$ the clock frequency
- $D$ the transition density [Naj93] of the output of the gate

The transition density is a measure of the number of transitions during a clock cycle. In section 2.5 we will discuss the calculation of transition densities and the assumptions influencing their calculation in more detail.
2.3.2 Short-circuit power dissipation

Short-circuit power dissipation, like dynamic power dissipation, also occurs when a gate switches. During the transition there is a short time when both the NMOS as well as the PMOS transistor network conduct. Power is then dissipated. The amount of short-circuit power dissipation depends on how fast the gate can switch. This is influenced by the speed (rise or fall time) with which the input(s) of the gate switch, the gain factors of the transistors and the threshold voltage of the transistors. The frequently referenced article [Veen84] gives us a formulation for the short-circuit power dissipation of a CMOS–inverter:

\[
P_{\text{short-circuit}} = \frac{\beta}{12} \cdot (V_{dd} - 2V_T)^3 \cdot \frac{\tau}{T}
\]

(2.3)

The variables in equation 2.3 have the following meaning:

- \(P_{\text{short-circuit}}\) the short-circuit power dissipation
- \(\beta\) the gain factor of the gate
- \(V_{dd}\) the supply voltage
- \(V_T\) the threshold voltage of the transistors
- \(\tau\) the rise or fall time of a signal
- \(T\) the period of the signal with frequency \(f\), so \(T = \frac{1}{f}\)

In the derivation of equation 2.3 in [Veen84] it has been assumed that the gain factor and the threshold voltage of the PMOS– and NMOS–transistor network are the same. This is a reasonable assumption since such an assumption also leads to a gate which has similar rise and fall times. Both the assumption of equal gain factor and threshold voltage on the one hand and the assumption of equal rise and fall times are used extensively in the domain of logic synthesis.

2.4 Glitch propagation

CMOS–gates exhibit a filtering effect for signals with a period shorter than the propagation delay of the gate. In [Naj94] this effect is already mentioned and used to come to a more accurate estimation of the transition density when not working under a zero gate delay assumption. The filtering effect is, however, not completely described in this paper. The paper [Eis95] describes the filtering effect of CMOS–gates more thoroughly. In [Eis95] a distinction is made between three different cases:

- The width of the pulse is shorter than \(t_p\)
- The width of the pulse is between \(t_p\) and \(2 \cdot t_p\)
• The width of the pulse is longer than $2 \cdot t_p$

$t_p$ denotes the propagation delay of the gate (see figure 2.2). In the first case, when the pulse width $t_{pwi}$ is shorter than the propagation delay of the gate, the pulse will not propagate. In the last case when the pulse width $t_{pwi}$ is longer than two times the propagation delay the pulse will completely propagate with the same width. The most interesting case is of course the second one, when the pulse width is in between one and two times the propagation delay. The pulse will in this case propagate partially. According to [Eis95], a pulse with width $t_{pwi} \subseteq (t_p, 2t_p)$ will propagate with a width of $2(t_{pwi} - t_p)$. In the paper [Eis95] the filtering effect is only investigated for inverters. We expect the filtering effect to occur also, with a similar behavior, for more complex gates. The figures of section 2.2.5 on the filtering effect do at least strongly suggest so. The results of section 2.2.5 also suggest that the filtering effect occurs similarly for pulses created at a logic gate due to multiple changing inputs. In table 2.1 we have summarized the three different input pulse width cases and the corresponding output pulse width.

**Table 2.1** Output pulse width $t_{pwo}$ as function of input pulse $t_{pwi}$ width and propagation delay $t_p$

<table>
<thead>
<tr>
<th>$t_{pwi}$</th>
<th>$t_{pwo}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \leq t_{pwi} &lt; t_p$</td>
<td>$0$</td>
</tr>
<tr>
<td>$t_p \leq t_{pwi} &lt; 2t_p$</td>
<td>$2(t_{pwi} - t_p)$</td>
</tr>
<tr>
<td>$t_{pwi} &gt; t_p$</td>
<td>$t_{pwi}$</td>
</tr>
</tbody>
</table>

**2.5 Transition density calculation**

We will now introduce a measure of switching activity called transition density based on [Naj93]. Let $x(t)$, $t \in (-\infty, +\infty)$ be a function of time that takes the logic values 0 or 1. We can view $x(t)$ as a 0–1–stochastic process, that is strict-sense stationary and mean-ergodic. The equilibrium probability of $x(t)$, denoted by $P(x)$ is then:

$$P(x) = \lim_{T \to \infty} \frac{1}{T} \int_{-T}^{T} x(t)dt$$

(2.4)
We observe that $P(x)$ is simply the fraction of time that $x(t)$ is in the 1 state. If the number of transitions of $x(t)$ in the time interval $\left(-\frac{T}{2}, \frac{T}{2}\right]$ is called $n_x(T)$, we can define the transition density $D(x)$ of $x$ by:

$$D(x) = \lim_{T \to \infty} \frac{n_x(T)}{T}$$

(2.5)

We now discuss the calculation of transition densities in a circuit. We first have to define Boolean difference. For the Boolean variable $y$, the Boolean difference is:

$$\frac{\delta y}{\delta x} = y \mid x \oplus y \mid \bar{x}$$

(2.6)

where $y \mid x$ and $y \mid \bar{x}$ are the cofactors of $y$ relative to Boolean variable $x$ and its complement $\bar{x}$ respectively. The symbol represents $\oplus$ the exclusive–or operation. Now $P\left(\frac{\delta y}{\delta x}\right)$ is the probability that the Boolean difference of $y$ to $x$ evaluates to 1. The transition density at the output of a gate $y_j$, which is a function of inputs $x_i$, $i = 1, \ldots, n$ is then defined as:

$$D(y_j) = \sum_{i=1}^{n} P\left(\frac{\delta y_j}{\delta x_i}\right) D(x_i)$$

(2.7)

In equation 2.7 $D(y_j)$ is the transition density of signal $y_j$ which depends on the summation of the transition densities $D(x_i)$ of signals $x_i$ times the probability that the Boolean difference of $y_j$ to $x_i$ evaluates to 1.

BDDs (Binary Decision Diagrams [Bry86][Bry92]) are a commonly used data structure to store logic functions. A BDD–node can also store the transition densities corresponding to each net of the circuit. We use BDDs because a function $y$ is represented by a BDD through repeatedly taking the cofactor to the variables on which $y$ depends. The cofactors used in the Boolean difference of equation 2.6 are therefore readily available in this data structure. Both the calculation of the probability $P\left(\frac{\delta y_j}{\delta x_i}\right)$ as well as the transition densities can be performed while simply traversing the BDD in a depth first manner.

The transition density for a certain signal is dependent on the logic structure and the propagation delays of the circuit. We can assume several different delay models for the delays in a circuit, which lead to more complexity when a more realistic model is chosen. We distinguish four different delay models:
• zero delay model
• general delay model
• unit delay model
• inertial delay model

We will look into each delay model in more detail.

2.5.1 Zero delay model

A zero delay model implies the assumption of zero gate and wire delays in the circuit. This means that glitches are not taken into account in this model, as no delay differences exist in the zero delay model. The fact that a transition does not occur during every clock cycle due to the logic structure of the circuit can however be accounted for.

2.5.2 General delay model

A general delay model implies that actual propagation delays are taken into account. This means that glitches, due to differences in propagation delay along different paths in the circuit, will show up in the transition density.

2.5.3 Unit delay model

A unit delay model is a special case of the general delay model with only discrete timing events. That means that the propagation delays are defined in multiples of predefined units. As calculating the switching activity under this delay model is as difficult as with the general delay model and is not as realistic, we will not discuss it in greater detail.

2.5.4 Inertial delay model

The inertial delay model is an extension of the general delay model. So, propagation delays are taken into account in this model. The model is extended to include the filtering effect displayed by CMOS–gates, discussed in section 2.4. Calculating the transition densities of a circuit including the filtering effect was first described in [Naj94]. However it is not very clear in [Naj94] when a glitch or a short pulse is filtered out. We have therefore looked into the filtering effect more thoroughly in sections 2.2.5 and 2.4. It is not surprising that transition densities under the inertial delay model are even more complicated to compute than under the general delay model.

2.6 Linear programming

We have now arrived at the part of this chapter where we no longer discuss basic physical aspects concerning logic gates. Instead we will now discuss
several methods of numerical optimization. We start with linear programming.

Equation 2.8 states the linear programming problem, the problem of minimizing a linear objective function subject to linear equality constraints, while all the variables are nonnegative. In the context of mathematical programming it is customary to call the composite of an objective function together with a system of constraints a “program”.

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Ax = b, \ x \geq 0
\end{align*}
\]  
(2.8)

Note that linear inequalities can be easily rewritten to become linear equalities by adding a slack variable to each inequality. To each (primal) linear program (equation 2.8) there is a dual linear program. The dual linear program is given in equation 2.9. The solution of the dual program is the same as the solution of the primal program.

\[
\begin{align*}
\text{maximize} & \quad b^T y \\
\text{subject to} & \quad A^T y \leq c
\end{align*}
\]  
(2.9)

Of course we can rewrite equation 2.9 in order to exhibit equality constraints:

\[
\begin{align*}
\text{maximize} & \quad b^T y \\
\text{subject to} & \quad A^T y + z = c, \ z \geq 0
\end{align*}
\]  
(2.10)

Conventional mathematical analysis teaches us how we can find the local minima (maxima) of a function. We can find the global minimum (maximum) using the same method if the function is concave (convex). A linear program like in equation 2.8 is concave and the corresponding dual program as in equation 2.10 is convex.

We can also deal with the linear equality constraints, by using the method of Lagrange. The method of Lagrange involves the introduction of Lagrange multipliers. One Lagrange multiplier is introduced for each linear constraint. We can now rewrite equation 2.8 as follows:

\[
\begin{align*}
\text{minimize} & \quad L_p(x, y) = c^T x + y^T (Ax - b) \\
\text{subject to} & \quad x \geq 0
\end{align*}
\]  
(2.11)

A Lagrange multiplier is a new variable in the problem. Lagrange multipliers enable the representation of the objective function and the constraints in one new objective function. The Lagrange multipliers in equation 2.11 are assembled in the vector y. As all first order derivatives of equation 2.11 will
be zero at its minimum, \( \frac{\delta L_p}{\delta y_i} \) the derivative relative to Lagrange multiplier \( \delta y_i \) is also zero. This means that at the minimum of equation 2.11 the corresponding constraint is satisfied. There is a relation between the primal and dual forms of a linear program. The Lagrange multipliers \( y \) in equation 2.11 are the same variables as used in equations 2.9 and 2.10 for the dual linear program. If we would rewrite equation 2.10 using the method of Lagrange, the Lagrange multipliers introduced would be the same variables \( x \) as used in the primal linear program of equations 2.8 and 2.11.

There are basically two methods to solve linear programming problems as in equation 2.8: simplex solvers and interior point solvers. We look into some details of each of those types of solvers and compare their applicability for one simplex solver called LPsolve [Berk] and two interior point solvers called PCx [Czy97] and LOQO [Vand92].

### 2.6.1 Simplex solvers

The oldest and most well–known solver of linear programs is the simplex solver. The simplex method first converts all inequality constraints other than the nonnegativity constraints to equality constraints. In equation 2.8 we have already added slack variables to convert inequality constraints to equality constraints. The optimization problem is now written in the normal form. Next we make the conversion to the restricted normal form. In the restricted normal form we rewrite each equation such that each constraint has at least one variable with a positive coefficient that appears uniquely in this constraint. These variables are called basic variables. The remaining variables are called non–basic variables. We also rewrite the objective function such that it only contains non–basic variables.

For any problem written in the restricted normal form we can easily find a feasible basic vector. This vector might not give the optimal solution to the linear program. We find this basic vector by setting the non–basic variables to zero. Setting the non–basic variables to zero gives values for the basic variables which satisfy the constraints. Rewriting of the objective function is performed by substituting basic variables by the expression of non–basic variables of the corresponding constraint.

The simplex method now proceeds by making a series of exchanges between a basic and a non–basic variable and rewriting a constraint. We now substitute this constraint in the objective function in order to ensure that the objective function only contains non–basic variables. We however only make such an exchange if the objective function decreases, in case of a minimization problem, or increases, in case of a maximization problem. If no more
exchanges are possible that improve the objective function, this is the optimal value for the objective function. The basic vector now gives the solution to the optimization problem.

For more details on the simplex method we refer to [Pre92]. [Pre92] also describes dealing with degenerate feasible vectors in which no obvious exchanges are apparent.

2.6.2 Interior point solvers

We now use the observation that to each (primal) linear program as in equation 2.8 there is a dual linear program. The (optimal) solution of the dual program is the same as the solution of the primal problem. We insert the linear constraints into the objective function by introducing Lagrange multipliers for both the primal and the dual program. We also add logarithmic barrier functions to ensure all the variables in vectors \( x \) and \( z \) are greater than or equal to zero. We now obtain the set of equations 2.12 for the primal–dual linear program. So the primal–dual linear program is the combination of the primal and dual linear programs in Lagrangian form combined with logarithmic barrier functions and a scheme to control the barrier function control variable \( \mu \). This scheme is the additional set of constraints given by \( Xz = \mu e \). We use a logarithmic barrier function, because for large values of the variable to which the barrier function applies, the negative logarithm of this variable does not contribute much to the Lagrange function and has a fairly flat gradient. When the variable is near the barrier, 0, it contributes a lot to the Lagrange function and has a very steep gradient.

\[
L_p(x, y) = c^T x + y^T (Ax - b) - \mu \sum_{j=1}^{N} \ln(x_j) \quad (2.12)
\]

\[
L_D(x, y, z) = b^T y - x^T (A^T y + z - c) + \mu \sum_{j=1}^{N} \ln(z_j)
\]

\( Xz = \mu e \)

In equation 2.12 \( X \) is a diagonal matrix, which is given by \( \text{diag}(x_1, ..., x_N) \). The vector \( e \) is the unit vector, given by \( (1, ..., 1)^T \). Using Newton’s method to minimize \( L_p \), maximize \( L_D \), solve \( Xz = \mu \), we obtain equation 2.13. The barrier function control variable \( \mu \) in this system of equations is reduced each iteration. The vectors \( r_p = -Ax + b \) and \( r_D = A^T y + z - c \) are the values
of the primal and dual constraints of the problem for the current values of \( x, y, \) and \( z \). \( I \) is the identity matrix.

\[
\begin{bmatrix}
0 & A & 0 \\
-A^T & 0 & I \\
0 & Z & X
\end{bmatrix}
\begin{bmatrix}
\delta y \\
\delta x \\
\delta z
\end{bmatrix}
= \begin{bmatrix}
r_P \\
r_D \\
\mu e - Xz
\end{bmatrix}
\tag{2.13}
\]

We solve the nonlinear system of equations in equation 2.13 by using Newton's method. So each time we take a step in a Newton iteration we also update the barrier function control variable \( \mu \).

One of the problems with interior point methods is finding a starting point. This starting point has to be in the interior of the feasible area of the problem. The self–dual interior point method for linear programming as described in [Xu94] solves this problem by first solving the homogeneous problem, where \( r_P \) and \( r_D \) are 0, while choosing \( x = e, z = e \) and \( y = 0 \) initially.

A primal–dual algorithm for linear programming has an excellent criterion for termination. The difference between the objective function for the primal formulation and the objective function for the dual formulation should converge to zero. One can easily set a desired accuracy. The accuracy which has been reached up until the current number of iterations is defined by the gap between the primal and the dual objective function divided by the current value of the dual objective function. We keep iterating until the desired accuracy is reached. In many applications only an accuracy in a limited number of digits is required or even useful. This can lead to considerable speed–up as we have shown in an interior point linear programming problem for the purpose of gate sizing in [Jac97b].

\[
\frac{|c^T x - b^T y|}{|b^T y|} < \text{accuracy}
\tag{2.14}
\]

Another aspect of termination is terminating when the linear programming formulation is infeasible. The self–dual interior point method for linear programming as described in [Xu94] also identifies criteria to detect and terminate on infeasibility. We will not discuss infeasibility, because it is not relevant to the understanding of the remainder of the thesis.

An example of an interior point linear programming solver is PCx [Czy95], but many others exist. PCx uses the MPS–format as its input format. MPS is the defacto standard format for describing linear programming problems.

### 2.6.3 Cholesky factorization

One step in solving the Newton system of equations (equation 2.13) of a primal–dual linear program in a numerically stable manner involves
decomposing the positive definite matrix $AXZ^{-1}A^T$ in an upper and lower triangular matrix. This important step in solving equation 2.13 is performed using Cholesky factorization. Cholesky factorization factors the positive definite matrix, $AXZ^{-1}A^T$, in an upper and a lower triangular matrix, which are each others transpose. So $AXZ^{-1}A^T$ is rewritten in LDL$^T$, where L is the lower triangular matrix and D a diagonal matrix. An advantage of Cholesky factorization is that this factorization is numerically very stable and that the data structure for the sparse matrix representation can be computed before the start of the Newton iterations. This means that the repeated factorization of $AXZ^{-1}A^T$ during the Newton iterations only involves the numerical calculations as X and Z change.

The data structure used in Cholesky factorization can be very memory intensive, especially when A contains many dense columns, which create a large amount of fill–in in the triangular matrix. Because of its numerical stability and the predictability of the data structure of the factorization result, which eases reuse, Cholesky factorization is used by nearly all interior point methods. To be able to deal with dense columns, most interior point methods use special techniques. These techniques are able to deal with only a limited number of dense columns. We will not consider these techniques as we only aim to convey a general understanding of interior point solvers.

**2.6.4 Comparison of different LP solvers**

We have distinguished two basic types of solvers for linear programming problems: The simplex solver, and interior point solvers. In comparing the simplex solver with interior point solvers we observe that the interior point solvers can handle large LP problems, because of their better numerical stability. The better numerical stability is due to the fact that most numerical operations are performed on a positive definite matrix using Cholesky factorization, which is numerically very stable. However most simplex solvers, including LP_solve, solve either the primal or the dual equation system. Selection of the pivots in solving this set of equations greatly influences the numerical stability. Numerical problems prohibit the solving of large examples in the case of LP_solve. While other simplex solvers may incorporate methods more advanced than LP_solve to improve numerical stability, the restriction to either the primal or the dual set of equations has its drawbacks compared to interior point methods.

Interior point solvers however also have draw–backs. The aforementioned Cholesky factorization can cause a lot of fill–in in the factored triangular matrix. Even for large sparse linear programming problems, the factored matrix can become very dense. This means that interior point solvers require
of large amount of memory, compared to simplex solvers. Also, interior point
solvers require a lot more operations to calculate the factorization than a
simplex solver needs to solve the set of equations.

In summary small examples can most efficiently be solved by a simplex
solver. Interior point solvers are the best option for large examples, because
of their greater numerical stability. This feature enables the handling of large
scale problems. A limit on the use of interior point solvers, however, is their
memory requirement for large LP problems.

2.6.5 (Mixed) integer linear programming

We now briefly mention integer linear programming (ILP). Integer linear
programming is an extension to linear programming with the additional
constraint that all variables are non–negative integers. This makes solving
integer linear programs much more difficult than solving linear programs.
Solving an integer linear program basically entails the repeated solving of
linear programs in a branch and bound approach. Each time one of the
variables is bounded by the upper nearest integer and then the remaining
linear program is solved (one branch). The same variable is also bounded by
the lower nearest integer and the remaining linear program is solved (other
branch). The branching can be bound by keeping track of the best known
solution so far. In case the integer linear program is a minimization problem
the intermediate linear programs, in which some variables are fixed to an
integer and others are not, return a solution which is a lower bound on the
solution of the integer linear program. In case of a maximization problem
intermediate linear programs give an upper bound on the integer linear
program.

Integer linear programming can be very time consuming as the linear
program may worst case have to be solved $2^n$ times. In this case $n$ is the
number of (integer) variables. Since the number of times the linear program
has to be solved grows exponentially with the number of integer variables,
once can easily see that integer linear programming in general is not practical
for more than a few integer variables. For certain integer linear programs, for
which the constraints adhere to some special conditions and constructions it
has been shown that solving the linear program returns a solution for which
all integer variables of the integer linear program are integers. In these cases
solving the integer linear program is as easy as solving the corresponding
linear program. A discussion about which integer linear programs can be
solved in such an efficient way can be found in [Nem88].

Mixed integer linear programming is a mixture in which some variables are
non–negative integers, while others are nonnegative reals. A mixed integer
linear program (MILP) is just as hard as solving a general integer linear program (ILP). The branch and bound approach for the ILP can be used here on just the integer variables. This still leaves open the possibility of having to solve a LP $2^n$ times. In this case $n$ is the number of integer variables.

2.7 Nonlinear programming

Nonlinear programming is the problem of optimizing (minimizing or maximizing) a nonlinear objective function subject to nonlinear equality constraints and bounds on the variables of the problem. Of course, nonlinear inequality constraints can be transformed into equality constraints by adding a slack variable. The general nonlinear programming problem is defined as follows:

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad c_j(x) = 0, \quad 1 \leq j \leq m \\
& \quad l_i \leq x_i \leq u_i, \quad 1 \leq i \leq n
\end{align*}
\]  

(2.15)

In equation 2.15 $f(x)$ is the (nonlinear) objective function, $c_j(x)$ is one of the $m$ (nonlinear) constraints and $l_i$ and $u_i$ are the lower and upper bounds on variable $x_i$.

2.7.1 LANCELOT

The large scale nonlinear constraint optimization package LANCELOT [Con92] is specially designed for dealing with large instances of general nonlinear programming formulations as in equation 2.15. The nonlinear programming problems are described in a special format called SIF. A description of SIF can be found in [Con92]. Both the optimizer LANCELOT and the input specification format SIF are targeted to describing and solving the nonlinear programming problem. LANCELOT uses an augmented Lagrange method for dealing with the nonlinear (and, in general, non–convex) constraints.

2.7.2 Augmented Lagrange method

LANCELOT solves nonlinear programming problems by first transforming inequality constraints to equality constraints by means of slack variables. LANCELOT also explicitly distinguishes between nonlinear constraints and linear constraints. This distinction is also made between the linear and nonlinear parts of a single constraint. Lagrange's method is used to convert
the constrained programming problem to an unconstrained one. Due to the nonlinear constraints the programming problem is not necessarily convex (in case of minimizing the objective) or concave (in case of maximizing the objective). Using Newton's method, which solves a local optimization problem, finding a global optimum is not guaranteed. Therefore the problem has to be made convex (concave). This is achieved with the augmented Lagrange Method.

The augmented Lagrange method entails adding an additional term to the Lagrange formulation containing the square of the nonlinear constraints and a penalty parameter. This process ensures the existence of a penalty parameter for which the augmented Lagrange formulation is convex and which therefore causes Newton's method to converge to the global optimum.

The augmented Lagrange formulation is:

\[
\text{minimize} \quad \phi(x, \lambda, S, \mu) \tag{2.16}
\]

in which:

\[
\phi(x, \lambda, S, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i=1}^{m} s_{ii} c_i(x)^2 \tag{2.17}
\]

In equation 2.17 \( f(x) \) is the objective function, \( \lambda_i \) is the Lagrange multiplier for the \( i \)th constraint \( c_i(x) \), \( s_{ii} \) is a coefficient of the diagonal matrix \( S \) which contains the scaling factors for each of the constraints, and the scalar \( \mu \) is the penalty parameter. The penalty parameter \( \mu \) is a means by which LANCELOT controls the optimization process. So \( \mu \) is not optimized.

As soon as the optimum of the function is found given a certain value of the penalty parameter, the penalty parameter is decreased in order to emphasize the impact of the constraints on the function and therefore force LANCELOT to come up with a solution that satisfies the equality constraints. The penalty parameter is reduced iteratively instead of being chosen small initially in order not to have to many local optima in the function in which the optimization process can get stuck. The engineering work put into LANCELOT, apart from fine tuning the speed and accuracy of the numerical operations, is in the strategy behind controlling the penalty parameter. We will not discuss the intricacies of controlling the penalty parameter as this is not vital to the understanding of this thesis. Additional information can be found in [Con92].

### 2.7.3 Derivatives

LANCELOT is able to solve large scale nonlinear programs. The scale of the programming problem that can be solved depends on many factors, among which:
• the number of variables
• the number of constraints
• the total number of terms in the problem (both in the objective function and in the constraints)
• how many of these terms are nonlinear
• whether the nonlinear terms are smooth or not
• the availability of accurate derivatives

We have solved problems with several thousands of variables and several thousands of constraints, containing nonlinear terms. To do that, LANCELOT needs at least to be able to calculate the value of \( \phi(x, \lambda, S, \mu) \) for given values of the variables. LANCELOT also requires partial derivatives. The ability to provide first order derivatives greatly improves the performance and avoids, to a certain extent, getting stuck in local optima. For strongly nonlinear functions it is advisable to supply second order derivative information also. This will also serve to avoid local optima, as well as to speed up the convergence. LANCELOT will need fewer iterations, and subsequently fewer evaluations of the function \( \phi \) and fewer derivative evaluations.

It is also possible to calculate first and second order derivative information via a method of divided differences. This is time-consuming and numerically less stable. It is therefore advantageous to evaluate a derivative directly, through the evaluation of an analytical expression. If the functions in question are continuously differentiable it is preferable to make use of the analytical expressions for all the derivative terms.

### 2.8 Geometric programming

A special case of nonlinear programming is geometric programming. A general geometric programming formulation is given by:

\[
\begin{align*}
\text{minimize} \quad & g_0(t) \\
\text{subject to} \quad & g_k(t) \leq 1, \quad k = 1, 2, \ldots, p \\
& t_j > 0, \quad j = 1, 2, \ldots, m
\end{align*}
\]

where

\[
\begin{align*}
g_0(t) &= \sum_{i=1}^{n_g} c_i \prod_{j=1}^{m} t_j^{a_{ij}} \\
g_k(t) &= \sum_{i=n_k+1}^{n_g} c_i \prod_{j=1}^{m} t_j^{a_{ij}}, \quad k = 1, 2, \ldots, p
\end{align*}
\]

The exponents \( a_{ij} \in \mathbb{R} \) are arbitrary constants and the coefficients \( c_i \) are positive. Both the objective function of equation 2.18 as well as the
constraints consist of summations of so-called posynomials. Posynomials are products of the variables of the geometric programming formulation to any real number exponent. Only summations of posynomials with positive coefficients \( c_i \) are permissible in a geometric program of the form given in 2.18. If the coefficients \( c_i \) are not all positive the programming formulation is not (pseudo)-convex, and cannot be solved easily.

A geometric program of the form of equation 2.18 also has a dual program. The dual is a nonlinear objective function subject to linear equality constraints. However, in geometric programming the dual program does not use the Lagrange variables of the primal program as its variables, as we have seen for linear programming. We will just give the dual and refer to [Duff67] for a complete explanation of the relation between primal and dual geometric programs. The dual is of the following form:

\[
\begin{align*}
\text{maximize} & \quad \prod_{i=1}^{n_p} (c_i/x_i)^{x_i} \prod_{k=1}^{p} \lambda_k^{\lambda_k} \\
\text{subject to} & \quad \sum_{i=1}^{n_p} x_i = 1 \\
& \quad \sum_{i=1}^{n_p} x_i a_{ij} = 0, \quad j = 1, 2, ..., m \\
& \quad x_i \geq 0, \quad i = 1, 2, ..., n_p \\
\end{align*}
\]  

(2.19)

where

\[
\lambda_k = \sum_{i=n_{k-1}+1}^{n_k} x_i, \quad k = 1, 2, ..., p
\]

The dual geometric program of equation 2.19 has a nonlinear objective function and linear equality constraints. The negative logarithm of the objective function given in equation 2.20 happens to be a linearly constrained convex programming problem, which we can solve (see below).
minimize
\[ \sum_{i=1}^{n_p} x_i \ln \left( \frac{x_i}{c_i} \right) + \sum_{k=1}^{p} \sum_{i=n_{k-1}+1}^{n_p} x_i \ln \left( \frac{x_i}{c_i \lambda_k} \right) \]  \hspace{1cm} (2.20)

subject to
\[ \sum_{i=1}^{n_p} x_i = 1 \]
\[ \sum_{i=1}^{n_p} x_i a_{ij} = 0, \quad j = 1, 2, ..., m \]
\[ x_i \geq 0, \quad i = 1, 2, ..., n_p \]

where
\[ \lambda_k = \sum_{i=n_{k-1}+1}^{n_p} x_i, \quad k = 1, 2, ..., p \]

Looking at equation 2.20 we notice that a linear equality constraint is added for each posynomial, either in the objective function or in the constraints of the geometric program. We can also see that for each variable in each posynomial an additional entry in matrix \( A \) occurs. Therefore the size of a geometric program is dominantly dependent on the number of posynomials in the problem and the number of times each variable is used.

More information on geometric programming as well as solvers can be found in [Xu94] and [Tun95]. [Xu94] describes a primal–dual interior point method and [Tun95] an iterative procedure using linear programming to solve the geometric program.

### 2.9 Quadratic programming

Another special case of nonlinear programming that should be mentioned is quadratic programming. A quadratic program is defined as follows:

minimize \[ \frac{1}{2} x^T Q x + c^T x \]  \hspace{1cm} (2.21)

subject to \[ Ax = b, \quad x \geq 0 \]

Of course, also in the case of quadratic programming there exists a dual program to the primal program of equation 2.21. The dual program is:

maximize \[ b^T y - \frac{1}{2} x^T Q x \]  \hspace{1cm} (2.22)

subject to \[ A^T y + z = c, \quad x \geq 0, \quad z \geq 0 \]

We limit ourselves in this thesis to using and describing convex quadratic programming. Convex quadratic programming deals with quadratic pro-
grams as in equations 2.21 and 2.22 for a symmetric positive definite matrix \( Q \).

A positive definite matrix means that the determinant of the matrix is positive. If a matrix is symmetric and positive definite the matrix can be decomposed in an upper and lower triangular matrix, which are each others transpose, in a numerically stable way using Cholesky factorization. Note that the addition of the symmetric positive definite matrix \( AXZ^{-1}A^T \) due to the constraint matrix \( A \) and its transpose \( A^T \) is also symmetric and positive definite.

Like with linear programming (see section 2.6) we can use the method of Lagrange. So we can rewrite equation 2.21 as follows:

\[
\text{minimize} \quad \frac{1}{2} x^T Q x + c^T x + y^T (A x - b) \\
\text{subject to} \quad x \geq 0
\]  (2.23)

And we can rewrite the dual to:

\[
\text{maximize} \quad b^T y - \frac{1}{2} x^T Q x - x^T (A^T y + z - c) \\
\text{subject to} \quad x \geq 0, z \geq 0
\]  (2.24)

Using Lagrange's method we have been able to put the linear equality constraints into the objective function, like we have done for linear programming in section 2.6. We still have to deal with inequality constraints forcing the individual variables of \( x \) and \( z \) to be greater or equal to zero. We can enforce this similarly here, adding barrier functions for the individual variables of vectors \( x \) and \( z \). The equations for the primal–dual interior point method are then given by:

\[
L_p(x, y) = \frac{1}{2} x^T Q x + c^T x + y^T (A x - b) - \mu \sum_{j=1}^{N} \ln(x_j)
\]  (2.25)

\[
L_d(x, y, z) = b^T y - \frac{1}{2} x^T Q x - x^T (A^T y + z - c) + \mu \sum_{j=1}^{n} \ln(z_j)
\]

\[Xz = \mu e\]

We need matrix \( Q \) to be symmetric and positive semi–definite in order for the system to be solvable. The programming problem is only quadratic and solvable for a positive definite \( Q \). A program for which \( Q \) is zero is of course a linear program. The remainder of the solving of the quadratic program can be done in a similar way as the solving of the interior point linear program in section 2.6.2. An example of a convex quadratic interior point solver is LOQO [Vand92].
Chapter

3 Gate Sizing

3.1 Introduction

Gate sizing is the process of optimally assigning drive strengths to each of the individual gates of a circuit, given a certain cost function and constraints. Constraints refer to limits on the drive strength of the gate (widths of the transistors), precedence relations between gates and the implied constraints on the timing, and constraints imposed upon the relation between gate propagation delay and drive strength. Also constraints on the total propagation delay of the circuit may be given. The cost function is related to the goal of the optimization. A cost function may direct the optimization process towards minimal area, minimal power, minimal propagation delay (performance) of the circuit or any combination thereof.

Also other methods of increasing the drive strength of a gate exist: cloning and buffer insertion. Cloning is the process of duplicating identical gates and placing the gates in parallel in the netlist in order to increase the drive strength driving the net. Theoretically this is the same as simply duplicating the width of the gate. However cloning exhibits behavior different from gate sizing:

• there is additional wiring between the original gate and its clone;
• the clone may behave different from the original because of physical margins of the fabrication process;
• similarly the wiring around the clone may behave different from the one around the original gate because of the margins

Buffer insertion is the process of the insertion of buffers in a tree–like form in order to increase the drive strength. The capacitance of the original gate is distributed over the buffers in the leaves of the tree circuit.

A considerable amount of papers has been written on the subject of gate and buffer sizing. Section 3.2 will discuss the most notable of these papers and their contributions. In section 3.3 we present a model for a sizable gate and discuss in detail the relation between speed factor, load capacitance and delay of the gate. Section 3.4 deals with modeling the average power dissipation of a circuit enabling us to establish an objective function for the gate sizing problem.
We will then discuss linear programming, geometric programming and
general nonlinear programming approaches to gate sizing. Further we
present experiments using these programming approaches in sections 3.6,
3.7 and 3.8. In section 3.5 we provide an overview of the experiments and
which experiments can be compared to each other. These experiments enable
us to compare speed, accuracy and memory usage of the programming
approaches in section 3.9. In section 3.9 we will also make some general
remarks on the choice of model (linear, nonlinear, or geometric) and
implementation of the solvers for these models.

Section 3.10 discusses modeling glitches in the context of gate sizing. In
section 3.11 we present a gate sizing approach using nonlinear programming
dealing with glitches. This approach works around the problem of changes in
the transition densities under a realistic delay model while gate sizing, as a
result of changes in propagation delays.

3.2 Related Work

The idea of speeding a gate up by increasing the drive capability is
well-known. Probably the most-referenced paper in the context of gate sizing
is [Fish85], which describes a sizing approach know as TILOS. This paper
already observes that the transistor sizing problem, which is related to gate
sizing, can be modelled as a geometric programming problem. It does however
not solve the formulation using a geometric programming solver. The paper
[Ber90] introduces a model of a sizable CMOS gate. The paper observes that
the gate sizing problem is convex and subsequently makes a piecewise linear
(PWL) approximation. The PWL approximated problem can be solved using
a linear programming solver. The paper [Ber96] is an extension to the model
of [Ber90] which is capable of computing the entire area/power/delay
trade–off curve using PLATO, a linear complementary problem based
simulation package. We will not compare to the results obtained using
PLATO, because the linear complementary problem is not an optimization
problem formulation. The paper [Che95] also uses a linear program. In this
paper the nonlinear gate sizing formulation is iteratively linearized and
solved by a linear program.

All papers discussed so far use some kind of mathematical formulation to
model the relation between drive strength and gate propagation delay as we
will do in the remainder of this chapter. The paper [Jac99c] gives an overview
of possible gate sizing formulations and methods to solve them for gate sizing
problems based on mathematical formulations. However, approaches which
determine propagation delays as a result of a change in the drive strength
through simulation also exist. A state of the art example of such an approach
is [Con99]. Simulation of the circuits and calculating gradients through
simulation however takes considerable time. Generally such an approach
can only deal with circuits which have considerably less gates than
approaches using a mathematical relation between drive strengths and gate
propagation delay.

The last paper we want to take into consideration combines gate sizing and
clock skew optimization [Sath95]. In order to accomplish this nonlinear
optimization task, which is in itself not convex, the paper proposes an
approach which repeatedly solves a convex gate sizing problem (a gate sizing
problem which is formulated as a convex (nonlinear) optimization problem)
and then a linear program to optimize the clock skew. Though the paper
shows improvements for the examples presented, no definite results are
established about the convergence of the alternation between solving gate
sizing and optimizing clock skew. We discuss this paper here because the idea
of gate sizing with a nonlinear program and two-sided timing constraints for
the clock skew optimization (minimum and maximum propagation delays)
bears some resemblance with the gate sizing approach incorporating glitches,
which we will discuss in section 3.10 and 3.11. The work in sections 3.10 and
3.11 is an extension of that presented in [Jac96] and [Jac97a].

3.3 Sizing a gate

We can increase the ability of a gate to drive the capacitances connected to its
output by several means. The first method is by diminishing the load
capacitance. We can do this through either buffer insertion or doubling the
gate (cloning), connecting the outputs together. The other method is to
increase the width of the transistors of the gate thus reducing the resistance
which drives the load capacitance. We will now discuss a model of a
CMOS–gate which includes all the capacitances and resistances affected by
sizing. A standard CMOS–gate can be modeled as in figure 3.1. This is
basically the same gate model as used in [Ber92].


**Figure 3.1.** Model of a CMOS gate

The model of figure 3.1 contains the following resistances and capacitances and switches:

- $C_{\text{int}}$ the internal capacitances of this gate
- $R_{\text{PMOS}}$ the resistance of the PMOS transistor network of the gate
- $R_{\text{NMOS}}$ the resistance of the NMOS transistor network of the gate
- $t_{\text{RP}}$ a switch indicating if the PMOS network of the gate is conducting
- $t_{\text{RN}}$ a switch indicating if the NMOS network of the gate is conducting
- $C_{\text{wire}}$ the capacitance of the wires driven by this gate
- $C_{\text{inj}}$ the gate oxide capacitance of gate $j$ driven by this gate

We now introduce sizing factors, also called speed–factors, for a gate. In the gate models of figures 3.1 and 3.2 we will call the sizing factor for the gate under consideration $S_{\text{gate}}$ and for the gates driven by this gate $S_{j}$. We now introduce the sizing factors into the gate model in figure 3.2.
**Figure 3.2.** Sizable model of a CMOS gate

As we can see in figure 3.2 both the resistance of the PMOS and the NMOS network and the internal capacitance $C_{\text{int}}$ depend on $S_{\text{gate}}$. This means that the delay due to the internal capacitances of the gate will remain constant while sizing. The delay is represented in equation 3.1 by $t_{\text{int, gate}}$. In the context of gate sizing we generally assume that a gate is designed such that it has as much resistance in the PMOS network as in the NMOS network. This means that under similar circumstances (input slew, capacitance load) rise time and fall time of a gate are equal. We now introduce the equation relating the sizing factor $S_{\text{gate}}$ to the propagation delay of the gate:

$$t_{\text{p, gate}} = t_{\text{int, gate}} + c \cdot \frac{C_{\text{wire}} + \sum_{j \in \text{succ(gate)}} C_{\text{in,j}} S_j}{S_{\text{gate}}}$$ \hspace{1cm} (3.1)

We call equation 3.1 the “sizable gate propagation delay” equation (SGPD). The following variables are used in equation 3.1:

- $t_{\text{p, gate}}$ the propagation delay of the gate.
• \( t_{\text{int.gate}} \) the delay due to the internal capacitances of the gate.
• \( C_{\text{in,j}} \) the gate oxide capacitance of the input of gate \( j \) in the output cone (fanout) of this gate with \( S_j = 1 \).
• \( C_{\text{wire}} \) the capacitance of the wires driven by the gate.
• \( S_{\text{gate}} \) the speed factor of the gate.
• \( S_j \) the speed factor of gate \( j \).
• \( c \) denotes a constant of unit \( \text{ns/pF} \) relating time to capacitance.

The sizing of a gate does not affect the wire capacitances. The capacitances of the gate oxide scale with the sizing of the gate. In the upper part of the fraction in equation 3.1 we see the total load capacitance of a gate. The total load capacitance consists of the wire capacitance and the total capacitances of the gate oxide of the gates in the output cone that are driven. This total load capacitance is divided by the speed factor \( S_{\text{gate}} \).

### 3.4 Modeling power in a gate sizing formulation

When we perform gate sizing for minimal power dissipation we have to model the power dissipation in the objective function of the (non-)linear program. We will now look at the contribution of a single gate to the objective function for the entire circuit. The total objective function can then be assembled from the contributions of each individual gate. In the previous section we have characterized a gate by a single parameter: the speed factor \( S_{\text{gate}} \). The short-circuit power dissipation as given in equation 2.3 increases as the gate is assigned a larger width and of course scales with this speed factor \( S_{\text{gate}} \), because the gain factor \( \beta \) becomes larger as \( S_{\text{gate}} \) becomes larger. For each gate in the circuit we should therefore see a term \( c_{\text{scp}} \cdot S \) in the objective function where \( c_{\text{scp}} \) is a constant given by:

\[
c_{\text{scp}} = \frac{\beta_{\text{orig}}}{12} \cdot (V_{\text{dd}} - 2V_T)^3 \cdot \tau \cdot f \cdot D
\]  

(3.2)

Note that in equation 3.2 the measure \( \frac{1}{T} \) of equation 2.3 is replaced with frequency \( f \) to obtain an expression in similar variables as we will use for dynamic power dissipation. The gain factor \( \beta_{\text{orig}} \) denotes the gain factor when the speed factor is assumed to be 1. Also, the transition density \( D \) is inserted to obtain the right number of transitions. The transition density \( D \) is calculated using the zero delay model assumption. This is obviously a simplification of reality. For a discussion on the calculation of transition densities at the output of a gate see section 2.5. We will comment on the interaction of transition density calculation and gate sizing in section 3.11.
The transition density is only calculated once, prior to the gate sizing process. Equation 3.2 does not give the weight for the total short-circuit power dissipation contribution for a gate, but the average short-circuit power dissipation per second for a gate. As mentioned before, we assume $\beta_{\text{orig}}, V_{dd}, V_T, \tau$ and $f$ to be constants in the course of the gate sizing process.

Also the dynamic power dissipation of each gate has to be considered in the objective function. Similar to equation 3.2 for short-circuit power we can determine a constant $c_{\text{dyn}}$ for the dynamic power dissipation of a gate. Equation 3.3 gives the term that will appear in the objective function for the dynamic power dissipation.

$$c_{\text{dyn}} = \frac{1}{2} \cdot C_{\text{int}} \cdot V_{dd}^2 \cdot f \cdot D_{\text{gate}} + \sum_{j \in \text{pred}(\text{gate})} \frac{1}{2} \cdot C_{\text{int},j} \cdot V_{dd}^2 \cdot f \cdot D_j$$  \hspace{1cm} (3.3)

As we can see in equation 3.3 the equation for a single gate consists of several terms. The first term in equation 3.3 is due to the transition density $D_{\text{gate}}$ at the output of the gate whose internal capacitance $C_{\text{int}}$ scale with $S_{\text{gate}}$. The second term in equation 3.3 is due to the transition densities $D_j$ at the inputs $j$ of this gate. This is because the gate oxide capacitances $C_{\text{int},j}$ also scale with the same sizing factor $S_{\text{gate}}$.

In the objective function we will find the speed factors $S$ of each gate each multiplied by the corresponding factor $c_{\text{scp}} + c_{\text{dyn}}$ calculated for that gate.

The power model described in this section can deal with any delay model. However, since gate sizing changes propagation delays of gates in the circuit, which has an influence on the transition densities, the only delay model for power dissipation computation that is practical in combination with gate sizing is the zero delay model.

### 3.5 Gate sizing

We now look at several formulations of the gate sizing problem. In section 3.6 we formulate the gate sizing problem as a linear program. We use three different linear programming algorithms: LP_solve, PCx and LOQO. In section 3.7 we formulate the gate sizing problem as a geometric program. Finally in section 3.8 we look into gate sizing formulated as a general nonlinear programming problem. We solve this general nonlinear program with LANCELOT. We solve the gate sizing problem both minimizing the maximum circuit propagation delay (speed) and minimizing the power.
dissipation of the circuit (power). If we optimize for speed there are two objective functions possible: one minimizing a combination of the maximum circuit propagation delay and less importantly power and the other of just minimizing the maximum circuit propagation delay. Similarly when optimizing power there are also two objective functions possible: one minimizing a combination of power and less importantly maximum circuit propagation delay and the other just minimizing power. As we will become clear in section 3.6 the optimization of an objective function which is a combination of power and maximum propagation delay is a necessity when gate sizing using linear programming. Table 3.1 gives an overview of upcoming tables presenting results for gate sizing on several benchmarks circuits using the different objectives, formulations and solvers. The tables mentioned on the same row of table 3.1 are directly comparable. We will report power in tables 3.2 through 3.15 in pJ per clock cycle.

<table>
<thead>
<tr>
<th>objective</th>
<th>linear programming</th>
<th>nonlinear programming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>simplex</td>
<td>interior point</td>
</tr>
<tr>
<td>minimization of</td>
<td>LP_solve</td>
<td>PCx LOQO</td>
</tr>
<tr>
<td>speed</td>
<td>combination of $T_{\text{max}}$ and power</td>
<td>3.2</td>
</tr>
<tr>
<td>just $T_{\text{max}}$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>power</td>
<td>combination of power and $T_{\text{max}}$</td>
<td>3.3</td>
</tr>
<tr>
<td>just min power</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### 3.6 Gate sizing using linear programming

#### 3.6.1 Linear programming formulation for gate sizing

The gate sizing formulation for power minimization or speed optimization according to [Ber90] and [Ber92] is given in equation 3.4. The following variables are used additionally in the formulation:

- $T_{\text{out},i}$ the total delay at the output of gate $i$
- $T_{\text{i},i}$ the total delay at input $j$ of gate $i$
- $T_{\text{max}}$ the maximum circuit propagation delay
- $c_i$ the constant that incorporates $c_{\text{dyn}} + c_{\text{esp}}$ for gate $i$
- $c_t$ the constant weighing the circuit propagation delay
• 'limit' the maximum speed factor of the gate (generally taken the same for all gates)

Then the gate sizing problem can be phrased as follows:

\[
\text{minimize } \sum_{i \in \text{gates}} c_i S_i + c_T T_{\text{max}} \tag{3.4}
\]

for all gates \( i \) subject to

\[
t_{p,i} \geq t_{\text{int},i} + c \cdot \frac{C_{\text{wire},i} + \sum_{j \in \text{succ}(i)} C_{\text{in},j} S_j}{S_{\text{gate}}} \tag{3.4a}
\]

\[1 \leq S_i \leq \text{limit} \tag{3.4b}\]

\[T_{\text{out},i} \geq T_{j,i} + t_{p,i} \quad \text{for all inputs } j \tag{3.4c}\]

\[T_{\text{max}} \geq T_{\text{out},i} \tag{3.4d}\]

The mathematical formulation for gate sizing comprises the constraints for the SGPD which has been cast to an inequality. Also the speed factor \( S_i \) needs to be limited. The other two inequalities give the relations between gates with respect to their timing behavior. The output of a gate does not stabilize until a time \( T_{\text{out},i} \) which is larger than each individual arrival time \( T_{j,i} \) at input \( j \) added to the propagation delay of the gate \( t_{p,i} \). Similarly the propagation delay of the circuit \( T_{\text{max}} \) is larger than the times \( T_{\text{out},i} \) for all gates \( i \).

In the next equation the SGPD inequality of equation 3.4a for the propagation delay of a gate is PWL approximated. This is possible because the propagation delay of a single gate as a function of its sizing factor is a convex function and the propagation delay only needs to be bounded from below. We approximate each nonlinear gate propagation delay equation by n linear pieces. The constants \( k_j, l = 1, \ldots, 3n \) are introduced as a result of the PWL approximation. The total linear programming formulation for gate sizing is then:
minimize \[ \sum_{i \in \text{gates}} c_i S_i + c_t T_{\text{max}} \] (3.5)

for all gates \( i \) subject to

\[ t_{p,i} \geq k_1 - k_2 S_i + k_3 \sum_{j \in \text{succ}(i)} C_{\text{inj}} S_j \]

\[ \ldots \]

\[ t_{p,i} \geq k_{3n-2} - k_{3n-1} S_i + k_{3n} \sum_{j \in \text{succ}(i)} C_{\text{inj}} S_j \]

\[ 1 \leq S_i \leq \text{limit} \]

\[ T_{\text{out},i} \geq T_{j,i} + t_{p,i} \quad \text{for all inputs } j \]

\[ T_{\text{max}} \geq T_{\text{out},i} \]

The linear program for gate sizing yields the minimum value of the objective function. The linear program also delivers the values of variables \( S_i, t_{p,i} \) and \( T_{\text{out},i} \) for each gate for which the objective function is minimal. Also the resulting \( T_{\text{max}} \) is returned.

An approximation of each gate propagation delay equation in three linear pieces is reasonable, as is suggested in [Ber92]. We choose the upper bound for each sizing factor 'limit' to be 3, as is also suggested in [Ber92].

### 3.6.2 Results with LP_solve

Tables 3.2 and 3.3 display the results of solving several benchmark circuits using the simplex linear programming package LP_solve [Berk] (see also section 2.6.1). In table 3.2 we size in order to minimize the maximum propagation delay of the circuit (maximize the speed of the circuit).

In table 3.2 and the subsequent tables giving results for gate sizing, we list the name of the benchmark circuit in column 1 and the number of gates the circuit contains in column 2. If the gate sizing problem minimizes speed the actual propagation delay of the circuit is in column 3 and the power dissipation in column 4. If the gate sizing problem minimizes power, the delay constraint is in column 3, the power dissipation in column 4 and the resulting circuit propagation delay after sizing in column 5. The last two columns contain the memory used for solving the gate sizing formulation and the amount of CPU-time used for solving the gate sizing formulation. The experiments are performed on a Hewlett Packard K260 containing 512 MB of main memory, two PA8000 processors each running at 180 MHz. We have used just one processor for the experiments.
**Table 3.2** Minimization for speed using LP_solve

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay (ns)</th>
<th>power (pJ)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>58.98</td>
<td>742.28</td>
<td>0.8</td>
<td>2.7 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>51.97</td>
<td>2486.28</td>
<td>3.4</td>
<td>15.0 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>58.44</td>
<td>1377.41</td>
<td>1.6</td>
<td>8.2 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>70.48</td>
<td>4135.17</td>
<td>6.3</td>
<td>46.0 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

We can see in table 3.2 that LP_solve is able to successfully complete the four smallest benchmark circuits. The larger benchmark circuits are not completed due to the susceptibility of LP_solve to numerical instability in the solving of the set of linear equations (see also section 2.6).

Table 3.3 gives results for minimizing power. In this and the upcoming gate sizing formulations, we will assume a zero delay model for switching activity calculation. As gate sizing changes the timing in a circuit, the switching activity under a general or inertial delay model would change. As such changes in transition densities occur stepwise, we cannot deal with them in a gate sizing formulation. We have calculated the transition densities using the method described in section 2.5. For the primary inputs we have assumed a transition density of 0.5. For the circuit C6288, which is a multiplier, we do not weigh the transition densities at the output of the gates. This assumption is necessary as this circuit cannot be represented by a Binary Decision Diagram (BDD) in a reasonable amount of memory. Therefore we can not compute the transition density for this circuit. There is also a constraint set on the maximum propagation delay of the circuit in this set of experiments. This delay constraint is set slightly higher than the minimal propagation delay results for the circuits, as presented in tables 3.2, and for the interior point linear programming solvers in tables 3.4 and 3.6.
<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay constraint (ns)</th>
<th>power (pJ)</th>
<th>delay (ns)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>62.0</td>
<td>548.26</td>
<td>62.00</td>
<td>1.3</td>
<td>2.9 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>53.0</td>
<td>2085.82</td>
<td>53.00</td>
<td>2.1</td>
<td>14.4 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>60.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>73.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>120.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>155.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>230.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>250.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>540.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>255.0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

In Table 3.3 we observe that LP_solve can only compute the gate sizing formulations for the circuits C432 and C499 successfully. The computation for the remaining circuits fails due to numerical instability. We can see that, for the successfully completed cases, the CPU–times are similar for the power minimization and for the minimization of the circuit propagation delay. The power dissipation figures in Table 3.3 are smaller than in Table 3.2, since the propagation delays required in Table 3.3 are slightly more relaxed compared to those in Table 3.2.

### 3.6.3 Results with PCx

We now solve the formulations, which are the same as used for Table 3.2, using the interior point solver PCx [Czy97] (see also section 2.6.2). Table 3.4 displays the results of gate sizing for a minimal propagation delay for several benchmark circuits.
TABLE 3.4  Minimization for speed using PCx

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay (ns)</th>
<th>power (ns)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>58.98</td>
<td>742.28</td>
<td>2.4</td>
<td>1.5 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>51.99</td>
<td>2489.02</td>
<td>4.3</td>
<td>2.3 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>58.44</td>
<td>1379.13</td>
<td>3.9</td>
<td>2.4 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>70.54</td>
<td>4147.58</td>
<td>5.8</td>
<td>3.1 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>118.17</td>
<td>5726.31</td>
<td>9.0</td>
<td>8.8 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>148.07</td>
<td>12157.27</td>
<td>14.7</td>
<td>19.0 s</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>223.71</td>
<td>21372.07</td>
<td>22.4</td>
<td>41.7 s</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>241.37</td>
<td>39689.23</td>
<td>27.8</td>
<td>42.3 s</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>526.42</td>
<td>115550.05</td>
<td>18.5</td>
<td>23.8 s</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>243.32</td>
<td>83812.37</td>
<td>39.0</td>
<td>1 m 3.1 s</td>
</tr>
</tbody>
</table>

The first observation comparing table 3.4 to table 3.2 is that in table 3.4 the gate sizing problems for all benchmark circuits were successfully completed. PCx also solves the formulations considerably faster than LP_solve. However, PCx uses more memory solving the formulations. This is the consequence of the Cholesky factorization, where for gate sizing a large amount of fill in occurs. The Cholesky factorization however also yields a larger numerical stability. All benchmark circuits are therefore completed successfully.

The minimal circuit propagation delays calculated using PCx are exactly the same as the ones calculated by LP_solve with the exception of C499. This difference can be attributed to the numerical instability of LP_solve.

In table 3.5 we use the same formulation as in table 3.3 to minimize the power dissipation under delay constraints. The delay constraints are chosen slightly larger than the minimal delays calculated in table 3.4 and are the same constraints as chosen in the experiments of table 3.3.


<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay constraint (ns)</th>
<th>power (pJ)</th>
<th>delay (ns)</th>
<th>memory (Mb)</th>
<th>CPU-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>62.0</td>
<td>548.26</td>
<td>62.00</td>
<td>2.4</td>
<td>1.2 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>53.0</td>
<td>2085.82</td>
<td>53.00</td>
<td>3.9</td>
<td>2.7 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>60.0</td>
<td>1283.88</td>
<td>60.00</td>
<td>3.5</td>
<td>1.9 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>73.0</td>
<td>3370.91</td>
<td>73.00</td>
<td>5.3</td>
<td>3.2 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>120.0</td>
<td>5442.35</td>
<td>120.00</td>
<td>8.1</td>
<td>7.0 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>155.0</td>
<td>11222.33</td>
<td>155.00</td>
<td>13.2</td>
<td>14.0 s</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>230.0</td>
<td>19695.44</td>
<td>230.00</td>
<td>20.5</td>
<td>35.6 s</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>250.0</td>
<td>38148.60</td>
<td>250.00</td>
<td>25.0</td>
<td>33.4 s</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>540.0</td>
<td>100859.75</td>
<td>540.00</td>
<td>18.5</td>
<td>17.5 s</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>255.0</td>
<td>76145.41</td>
<td>255.00</td>
<td>35.1</td>
<td>47.0 s</td>
</tr>
</tbody>
</table>

In table 3.5 we observe that all benchmark circuits are all completed successfully and all delay constraints are adhered to. In comparing to the results with LP_solve (see table 3.3) we see that PCx performs faster, with a slightly higher memory use. The power dissipation figures are the same for PCx and LP_solve for the completed examples.

3.6.4 Results with LOQO

LOQO [Vand92] is a convex quadratic programming solver, which can solve linear programs. In that case, the matrix Q, which contains the factors for the quadratic terms, is zero (see also equation 2.21). We again repeat the experiment minimizing the maximum propagation delay of the circuit under the same conditions as previously, but now using LOQO. Table 3.6 shows the results.
Table 3.6  Minimization for speed using LOQO

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay (ns)</th>
<th>power (pJ)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>58.98</td>
<td>742.30</td>
<td>1.7</td>
<td>0.8 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>51.97</td>
<td>2486.22</td>
<td>3.6</td>
<td>6.3 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>58.44</td>
<td>1377.47</td>
<td>2.6</td>
<td>3.5 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>70.48</td>
<td>4135.06</td>
<td>4.2</td>
<td>7.1 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>118.2</td>
<td>5697.12</td>
<td>6.8</td>
<td>19.0 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>148.1</td>
<td>12480.54</td>
<td>9.8</td>
<td>39.0 s</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>223.7</td>
<td>20982.61</td>
<td>20.0</td>
<td>4 m 38.8 s</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>241.4</td>
<td>39316.26</td>
<td>18.3</td>
<td>1 m 18.6 s</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>526.3</td>
<td>108266.90</td>
<td>13.7</td>
<td>48.8 s</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>243.3</td>
<td>81290.05</td>
<td>29.6</td>
<td>2 m 47.2 s</td>
</tr>
</tbody>
</table>

As table 3.6 shows, LOQO is, like PCx, able to successfully solve the gate sizing problems for all benchmark circuits. LOQO tends to use slightly less memory than PCx, but is generally slower than PCx. The calculated maximum propagation delay results using LOQO correspond exactly to the figures calculated with PCx, within the accuracy delivered by both solvers. The power dissipation figures for LOQO are however different from the ones calculated with PCx. The power dissipation figures for LOQO tend to be less than for PCx. The differences can be explained by the fact that the PWL approximated model allows sizing factors to be higher than necessary. Another aspect is in the fact that many sizing factors in the objective function have a weight $c_i$ (see equation 3.4) which is small compared to the weight $c_t$ for the maximum propagation delay, which can lead to numerical inaccuracies.

In table 3.7 we show the results of the power dissipation minimization using LOQO. The propagation delay constraints are chosen to be the same as in the experiments using LP_solve and PCx.
### Table 3.7 Minimization for power using LOQO

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay constraint (ns)</th>
<th>power (pJ)</th>
<th>delay (ns)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>62.0</td>
<td>548.27</td>
<td>62.00</td>
<td>1.1</td>
<td>0.8 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>53.0</td>
<td>2085.77</td>
<td>53.00</td>
<td>3.5</td>
<td>7.4 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>60.0</td>
<td>1283.87</td>
<td>60.00</td>
<td>2.5</td>
<td>2.5 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>73.0</td>
<td>3370.91</td>
<td>73.00</td>
<td>4.4</td>
<td>6.2 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>120.0</td>
<td>5442.11</td>
<td>120.00</td>
<td>6.4</td>
<td>12.4 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>155.0</td>
<td>11221.51</td>
<td>155.00</td>
<td>9.7</td>
<td>27.3 s</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>230.0</td>
<td>19693.68</td>
<td>230.00</td>
<td>14.8</td>
<td>2 m 51.9 s</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>250.0</td>
<td>38130.36</td>
<td>250.00</td>
<td>20.1</td>
<td>58.4 s</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>540.0</td>
<td>100857.61</td>
<td>540.00</td>
<td>13.6</td>
<td>37.7 s</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>255.0</td>
<td>76122.83</td>
<td>255.00</td>
<td>29.6</td>
<td>2 m 1.5 s</td>
</tr>
</tbody>
</table>

We observe that LOQO is also able to deal with all benchmark circuits. All delay constraints are adhered to. The power dissipation figures in the case using LOQO and in the case using PCx only differ in the fourth decimal. This can be contributed to the accuracy requirements for the solvers, which are set to avoid excessive run–times. LOQO uses slightly less memory, sometimes at the expense of a larger run–time than PCx.

### 3.7 Gate sizing using geometric programming

The paper [Fish85] already indicates that the problem of transistor sizing can be described by a formulation based on posynomials (see section 2.8). This is also the case for the related problem of gate sizing, where all transistors of a gate are sized by the same factor. The basic gate sizing model we use (presented in section 3.3) also leads to a gate sizing formulation based on posynomials. So, the gate sizing problem has a linear objective function containing only positive terms, which conforms to the description of posynomials. It has linear inequality constraints containing only positive terms for the precedence relations of the gates, which can also be described by posynomials. And it also contains the SGPD equations for each gate, but these also consist of posynomials. The whole gate sizing problem can therefore be modelled by a geometric program.

In order to derive the geometric programming formulation for gate sizing we take the gate sizing formulation of equation 3.4 and rewrite it as follows:
\[
\text{minimize} \quad \sum_{i \in \text{gates}} c_i S_i + c_i T_{\text{max}} \tag{3.6}
\]

for all gates \(i\) subject to

\[
t_{\text{int},i} t_{p,i}^{-1} + c \cdot \left( C_{\text{wire},i} S_i^{-1} t_{p,i}^{-1} + \sum_{j \in \text{succ}(i)} C_{\text{in},j} S_j S_i^{-1} t_{p,i}^{-1} \right) \leq 1.
\]

\[
S_i^{-1} \leq 1
\]

\[
\lim S_i \leq 1
\]

\[
T_{j,i} T_{\text{out},i}^{-1} + t_{p,i} T_{\text{out},i}^{-1} \leq 1 \quad \text{for all inputs } j
\]

\[
T_{\text{max}} T_{\text{out},i} \leq 1
\]

This leaves us with a formulation conforming to the one presented in section 2.8. We now present four sets of experiments. In the first two sets we minimize the maximal propagation delay of the circuit. The experiments presented in table 3.8, use an objective function which contains the maximal propagation delay and also incorporates the sizing factors. The sizing factors are incorporated in order to select them as small as possible, as was done in previous experiments using the linearized model. Since we are still optimizing for speed we have to stress the importance of \(T_{\text{max}}\) in the objective function. We achieve this by weighing \(T_{\text{max}}\) by the number of gates in the circuit: \#gates. The main difference between this formulation and equation 3.5 is the nonlinear (but geometric) formulation for the SGPDs. We can therefore expect slightly different results compared to the PWL approximated case.
Table 3.8: Minimization of \( \#gates \times T_{\text{max}} + \sum_{i \in \text{gates}} c_i S_i \) using geometric programming for speed

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay (ns)</th>
<th>power (pJ)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>60.53</td>
<td>768.82</td>
<td>2.4</td>
<td>17.6 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>50.16</td>
<td>2527.07</td>
<td>5.6</td>
<td>1 m 34.8 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>57.94</td>
<td>1389.52</td>
<td>7.8</td>
<td>5 m 36.1 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>67.98</td>
<td>4294.06</td>
<td>12.7</td>
<td>8 m 51.6 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>116.24</td>
<td>5800.92</td>
<td>29.5</td>
<td>58 m 4.2 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>–</td>
<td>–</td>
<td>92.5</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>–</td>
<td>–</td>
<td>162</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>–</td>
<td>–</td>
<td>262</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>–</td>
<td>–</td>
<td>120</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>–</td>
<td>–</td>
<td>&gt;512</td>
<td>–</td>
</tr>
</tbody>
</table>

Looking at Table 3.8 we observe that the five largest benchmark circuits are not completed successfully. For the largest benchmark circuit this is due to the fact that the formulation takes too much memory. The other four benchmark circuits do fit in memory, but take an excessive amount of time to complete and are stopped after more than four hours of CPU–time. We also observe that using the nonlinear (geometric) formulation instead of the PWL approximated one, the resulting delays are slightly larger than in the PWL approximated model. The power dissipation for the geometric programming formulation is also slightly higher than for the PWL approximated case.

The second set of experiments minimizes the maximum propagation delay. In contrast to the previous set of experiments, we now only include the maximum propagation delay in the objective function. The results of this set of experiments are presented in Table 3.9.
Table 3.9 Minimization of $T_{\text{max}}$ using geometric programming for speed

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay (ns)</th>
<th>power (pJ)</th>
<th>memory (Mb)</th>
<th>CPU-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>60.53</td>
<td>881.27</td>
<td>1.2</td>
<td>1.5 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>50.16</td>
<td>2704.02</td>
<td>1.8</td>
<td>2.5 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>57.94</td>
<td>2282.31</td>
<td>2.1</td>
<td>4.0 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>67.98</td>
<td>4576.16</td>
<td>4.1</td>
<td>16.4 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>116.24</td>
<td>8883.74</td>
<td>8.0</td>
<td>1 m 46.0 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>145.92</td>
<td>21291.71</td>
<td>10.3</td>
<td>1 m 41.9 s</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>221.06</td>
<td>33134.10</td>
<td>26.1</td>
<td>21 m 57.8 s</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>238.96</td>
<td>75975.99</td>
<td>22.7</td>
<td>5 m 42.7 s</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>518.01</td>
<td>134947.02</td>
<td>14.4</td>
<td>1 m 7.6 s</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>–</td>
<td>–</td>
<td>128</td>
<td>&gt;4h</td>
</tr>
</tbody>
</table>

If we compare table 3.9 to table 3.8, we observe that in case of the geometric programming solver, deleting the sizing factors in the objective function saves a lot of memory. We are therefore now able to complete the optimization for minimal circuit propagation delay of all but one of the benchmark circuits. The largest circuit now fails, not due to memory use, but due to excessive computation time. We observe no difference in the calculated maximal propagation delays, but the power dissipation is larger in this set of experiments, as it is not included in the objective function.

The last two sets of experiments using the geometric programming solver minimize the power dissipation of the sized circuit. In the experiments presented in table 3.10 we also take into account the maximum propagation delay, with a very small weight factor, in order to ensure that we calculate the smallest propagation delay possible, while optimizing for minimal power dissipation. This experiment is comparable to the experiment for the minimization for power in the PWL approximated case. We have not PWL approximated the constraints for the gate delays in this set of experiments.
**Table 3.10** Minimization for power \( \sum_{i \in \text{gates}} c_i S_i + 0.01 \times T_{\text{max}} \) using geometric programming

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay constraint (ns)</th>
<th>power (pJ)</th>
<th>delay (ns)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>62.0</td>
<td>650.83</td>
<td>62.00</td>
<td>2.4</td>
<td>15.3 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>53.0</td>
<td>1919.33</td>
<td>53.00</td>
<td>5.6</td>
<td>2 m 8.7 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>60.0</td>
<td>1283.63</td>
<td>60.00</td>
<td>7.8</td>
<td>4 m 12.9 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>73.0</td>
<td>3203.73</td>
<td>73.00</td>
<td>12.7</td>
<td>7 m 43.4 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>120.0</td>
<td>5347.67</td>
<td>120.00</td>
<td>29.5</td>
<td>53 m 1.5 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>155.0</td>
<td>–</td>
<td>–</td>
<td>92.5</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>230.0</td>
<td>–</td>
<td>–</td>
<td>162</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>250.0</td>
<td>–</td>
<td>–</td>
<td>262</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>540.0</td>
<td>–</td>
<td>–</td>
<td>120</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>255.0</td>
<td>–</td>
<td>–</td>
<td>&gt;512</td>
<td>–</td>
</tr>
</tbody>
</table>

When looking at table 3.10, we again observe that the gate sizing for the five largest benchmark is not completed successfully. With the largest benchmark circuit this is due to excessive memory use and with the other four due to excessive computation time. The five benchmark circuits for which the gate sizing was completed successfully all adhere to the delay constraints posed. Power is slightly higher than in the case of the experiments with the PWL approximated SGPDs.

In our last set of experiments using the geometric programming solver, we only include the power dissipation in the objective function. So we delete the maximum propagation delay compared to the previous experiment. The gate sizing problem is subject to constraints on the maximum propagation delay.
TABLE 3.11 Minimization for power (\(\sum_{i \in \text{gates}} c_i S_i\)) using geometric programming

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay constraint (ns)</th>
<th>power (pJ)</th>
<th>delay (ns)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>62.0</td>
<td>650.83</td>
<td>62.0</td>
<td>2.4</td>
<td>15.5 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>53.0</td>
<td>1919.33</td>
<td>53.0</td>
<td>5.6</td>
<td>2 m 9.6 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>60.0</td>
<td>1283.63</td>
<td>60.0</td>
<td>7.8</td>
<td>3 m 58.8 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>73.0</td>
<td>3203.74</td>
<td>73.0</td>
<td>12.7</td>
<td>7 m 41.6 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>120.0</td>
<td>5347.67</td>
<td>120.0</td>
<td>29.5</td>
<td>54 m 6.9 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>155.0</td>
<td>-</td>
<td>-</td>
<td>92.5</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>230.0</td>
<td>-</td>
<td>-</td>
<td>162</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>250.0</td>
<td>-</td>
<td>-</td>
<td>262</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>540.0</td>
<td>-</td>
<td>-</td>
<td>120</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>255.0</td>
<td>-</td>
<td>-</td>
<td>&gt;512</td>
<td>-</td>
</tr>
</tbody>
</table>

Like in table 3.10, we observe in table 3.11 that the gate sizing for the five largest benchmarks is not completed successfully for the same reasons as before. We also observe that the successfully completed benchmark circuits adhere to the delay constraints posed. We cannot observe any difference in the resulting power figures, but there are differences in the amount of CPU–time required to solve the formulation, though no trend is obvious.

In this section on gate sizing using geometric programming we have observed that only the experiments presented in table 3.9 are successful for the majority of the examples. The gate sizing problem presented in table 3.9 is different in that the amount of posynomials is considerably smaller than in the other geometric programs, because there the objective function only contains \(T_{\text{max}}\).

3.8 Gate sizing using nonlinear programming

The gate sizing formulation using nonlinear programming is of course very similar to the formulation using linear programming. However, since we allow nonlinear equations we do not have to PWL approximate the SGPDs of equation 3.1. Equation 3.1 consists of several nonlinear terms. Although we allow nonlinear terms, we still want to avoid them if possible, for reasons of efficiency. We therefore multiply both sides of equation 3.1 for gate \(i\) with \(S_i\),
the sizing factor for gate $i$. We derive the following equation for the gate propagation delay of gate $i$:

$$t_{p,i}S_i = t_{\text{int},i}S_i + c \cdot (C_{\text{wire},i} + \sum_{j \in \text{succ}(i)} C_{\text{in},j}S_j)$$  \hspace{1cm} (3.7)

Equation 3.7 has with $t_{p,i}S_i$ just one nonlinear term. The complete gate sizing formulation using nonlinear programming then becomes:

$$\text{minimize} \quad \sum_{i \in \text{gates}} c_i S_i + c_t T_{\text{max}}$$  \hspace{1cm} (3.8)

for all gates $i$ subject to

$$t_{p,i}S_i = t_{\text{int},i}S_i + c \cdot (C_{\text{wire},i} + \sum_{j \in \text{succ}(i)} C_{\text{in},j}S_j)$$

$$1 \leq S_i \leq \text{limit}$$

$$T_{\text{out},i} \geq T_{j,i} + t_{p,i} \quad \text{for all inputs } j$$

$$T_{\text{max}} \geq T_0$$

We can solve the minimization problem of equation 3.8 using the large scale nonlinear optimization package LANCELOT [Con92]. Note that in contrast to the previous formulations in equations 3.5 and 3.6 we bound the gate propagation delay by an equality constraint. This saves one slack variable per gate in the formulation. We can use this equality constraint because LANCELOT is able to deal with equality constraints (see section 2.7). LANCELOT internally even converts inequality constraints to equality constraints before solving. The gate propagation delay is simply a function of the sizing factor and can therefore be modeled as the equality constraint in equation 3.8.

We again perform four sets of experiments as has been the case with the geometric programming model of the previous section. In table 3.12 we present the first set of results for minimizing the maximum propagation delay. In the objective function we incorporate both the maximum propagation delay as well as the power dissipation. We take power dissipation into account in order to obtain the minimal power consumption corresponding to our objective of minimized maximum circuit propagation delay. Because power dissipation is incorporated in the objective function, we weigh the maximum propagation delay in the objective function by the number of gates (#gates). This way our objective remains optimization for speed.
Table 3.12 Minimization for speed of $\sum_{i \in \text{gates}} c_i S_i + \#\text{gates} \times T_{\text{max}}$ using LANCELOT

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay (ns)</th>
<th>power (pJ)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>60.53</td>
<td>768.26</td>
<td>1.8</td>
<td>39.6 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>50.16</td>
<td>2527.01</td>
<td>3.5</td>
<td>33.0 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>57.81</td>
<td>1390.91</td>
<td>8.2</td>
<td>22 m 22.9 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>67.98</td>
<td>4294.05</td>
<td>4.6</td>
<td>30.2 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>–</td>
<td>–</td>
<td>3.1</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>–</td>
<td>–</td>
<td>4.1</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>–</td>
<td>–</td>
<td>4.9</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>–</td>
<td>–</td>
<td>6.9</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>–</td>
<td>–</td>
<td>5.4</td>
<td>&gt;4h</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>–</td>
<td>–</td>
<td>9.0</td>
<td>&gt;4h</td>
</tr>
</tbody>
</table>

As can be observed in table 3.12 LANCELOT is not able to solve the gate sizing problem for all benchmark circuits successfully. For the six largest circuits the formulation is not solved due to an excessive run–time of over four hours. When comparing the resulting minimal propagation delays to the ones obtained by geometric programming under the same conditions (table 3.8) we observe absolutely no difference. The power figures are similar as in the geometric programming case. LANCELOT uses considerably less memory than the geometric programming method, but no significant trend can be seen regarding the run–times.

The next set of experiments minimizes the maximum propagation delay, but, unlike in the previous set, power dissipation is not in the objective function. The experiment is similar to the one of table 3.9 using the geometric programming solver.
When we look at table 3.13 we observe that now, in contrast to table 3.12, we are able to successfully compute the gate sizing problem for all benchmark circuits. The minimal propagation delays calculated in table 3.13 are marginally higher than those calculated in table 3.12. The explanation for this small difference lies in numerical issues involving the adding of all the weighed sizing factors and the large constant factor (#gates) for the maximum propagation delay in the experiments of table 3.12. When we compare the power figures, we also see that they are slightly higher than in the case of table 3.12. The change in objective function from table 3.12 to table 3.13 does not make much difference in the use of memory. This change in objective function decreases the run–times, with the exception of C499.

We can also compare the results of table 3.13 to those of table 3.9. We can see that LANCELOT is also able to complete the largest benchmark circuit successfully in contrast to the geometric program. For the benchmark circuits which were completed the geometric program however performs faster at the expense of a larger memory use. The propagation delays do not differ much, but there are some differences in the power dissipation figures. A trend in these figures is not visible.

We will now look into two sets of experiments minimizing power dissipation under a delay constraint. In table 3.14 we present a set of experiments in which not only the power dissipation, but also the maximum propagation delay is part of the objective function. This is a similar set–up as presented for the experiment of table 3.10 for the geometric programming case.
Table 3.14 Minimization for power $\sum_{i \in \text{gates}} c_i S_i + 0.01 \times T_{\text{max}}$ using LANCELOT

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay constraint (ns)</th>
<th>power (pJ)</th>
<th>delay (ns)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>62.0</td>
<td>648.12</td>
<td>62.00</td>
<td>1.8</td>
<td>10.7 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>53.0</td>
<td>1917.48</td>
<td>53.00</td>
<td>3.5</td>
<td>17.6 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>60.0</td>
<td>1283.37</td>
<td>60.00</td>
<td>8.2</td>
<td>1 m 2.1 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>73.0</td>
<td>3201.90</td>
<td>73.00</td>
<td>4.6</td>
<td>52.7 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>120.0</td>
<td>5339.59</td>
<td>120.00</td>
<td>3.1</td>
<td>6 m 19.0 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>155.0</td>
<td>11091.10</td>
<td>155.00</td>
<td>4.1</td>
<td>29 m 16.1 s</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>230.0</td>
<td>19431.86</td>
<td>230.00</td>
<td>4.9</td>
<td>50 m 46.6 s</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>250.0</td>
<td>37910.78</td>
<td>250.00</td>
<td>6.9</td>
<td>69 m 54.3 s</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>540.0</td>
<td>99421.20</td>
<td>540.00</td>
<td>5.4</td>
<td>56 m 12.3 s</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>255.0</td>
<td>75439.81</td>
<td>255.00</td>
<td>9.0</td>
<td>210 m 41.5 s</td>
</tr>
</tbody>
</table>

Table 3.14 shows us that the results delivered by LANCELOT all satisfy the delay constraints posed. If we compare table 3.14 to the similar experiment using the geometric programming formulation presented in table 3.10, we observe that using general nonlinear programming we are able to compute the results for all benchmark circuits in contrast to the geometric formulation. Also, we use less memory and less CPU–time to come to these results in this case. The power figures for the benchmarks completed in both cases are similar. Any differences in the power dissipation figures can be attributed to the required accuracy demanded of the different solvers.

The last set of experiments minimizes the power dissipation subject to a delay constraint. The maximum circuit propagation delay is not part of the objective function. The experiment is comparable to the one presented in table 3.11 for the geometric programming case.
**Table 3.15** Minimization for power $\sum_{i \in \text{gates}} c_i S_i$ using LANCELOT

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>delay constraint (ns)</th>
<th>power (pJ)</th>
<th>delay (ns)</th>
<th>memory (Mb)</th>
<th>CPU–time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>62.0</td>
<td>648.19</td>
<td>62.00</td>
<td>1.8</td>
<td>11.0 s</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>53.0</td>
<td>1917.47</td>
<td>53.00</td>
<td>3.5</td>
<td>19.4 s</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>60.0</td>
<td>1283.35</td>
<td>60.00</td>
<td>8.2</td>
<td>58.7 s</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>73.0</td>
<td>3201.90</td>
<td>73.00</td>
<td>4.6</td>
<td>43.1 s</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>120.0</td>
<td>5339.60</td>
<td>120.00</td>
<td>3.1</td>
<td>7 m 54.3 s</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>155.0</td>
<td>11091.07</td>
<td>155.00</td>
<td>4.1</td>
<td>30 m 1.0 s</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>230.0</td>
<td>19431.85</td>
<td>230.00</td>
<td>4.9</td>
<td>49 m 0.9 s</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>250.0</td>
<td>37910.79</td>
<td>250.00</td>
<td>6.9</td>
<td>74 m 36.6 s</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>540.0</td>
<td>99421.15</td>
<td>540.00</td>
<td>5.4</td>
<td>55 m 8.2 s</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>255.0</td>
<td>75439.88</td>
<td>255.00</td>
<td>9.0</td>
<td>248 m 32.1 s</td>
</tr>
</tbody>
</table>

All experiments presented in table 3.15 are completed successfully and satisfy the delay constraints posed. Comparing the results of table 3.15 to the corresponding experiments using the geometric programming solver presented in table 3.11 we observe that LANCELOT uses less memory and less CPU–time. The power figures are comparable. When we compare the results of table 3.15 to those of table 3.14 we observe that memory use, power figures and resulting delay values are the same or comparable. However, the experiments of table 3.15, especially the larger ones, use considerably less CPU–time.

### 3.9 Comparison of gate sizing using linear and nonlinear programming

Now that we have presented results for gate sizing using both simplex and interior point linear programming approaches, as well as geometric and nonlinear programming approaches, we can summarize the main observations.

From the presented results we can conclude that for regular gate sizing an approach approximating the SGPDs in a PWL way is by far the fastest. Such an approach has the most potential to deal with increasingly larger problems sizes. This is all under the provision that an interior point solver (PCx, LOQO) is used to solve the resulting linear program. A geometric programming solver has its limitations both in memory usage and required run–time. An approach using the general nonlinear programming package LANCELOT.
is also able to deal with all the benchmark circuits, using the nonlinear SGPD constraints directly, but at the expense of a considerably larger run-time. However, this approach uses a less memory than the PWL approximation approach.

In case of minimizing power, the lowest power dissipation figures are obtained using the PWL approximation approach and the interior point solvers. The lower figures are in part due to the PWL approximation of the SGPDs. In the case no PWL approximation takes place the best power figures are in most cases obtained for the experiments with LANCELOT, though there are no major differences.

In case of minimizing the circuit propagation delay the fastest results in general can be obtained using LANCELOT, because PWL approximation of the SGPDs is not necessary. LANCELOT is also able to solve the problem of sizing all benchmark circuits for speed. The interior point solvers are also able to solve all benchmarks for speed, but due to the PWL approximation of the SGPDs they both tend to be a little slower.

### 3.10 Timing Constraints for Glitch Removal

We will now present an approach to handle glitches in the context of gate sizing and to deal with transition density changes as result of gate sizing. Our approach consists of removing glitches. The best way to ensure that glitches are removed is, trivially, to make sure that they do not occur. If all the path delays of the different paths in the circuit are balanced no gate will generate a glitch. But the paths do not have to be completely balanced. A small margin is allowable, because of the filtering effect. We can see in table 2.1 that a glitch will not occur if the arrival times of signals do not differ more than the propagation delay \( t_p \) of the gate. We introduce the following constraints:

\[
\forall j \neq k \quad T_{j,i} \geq T_{k,i} - t_{p,i} \tag{3.9}
\]

Here \( T_{j,i} \) and \( T_{k,i} \) are the total delay times at the inputs of gate \( i \). For simplicity, we neglect the fact that propagation delays can differ in case of rising and falling transitions. Also we assume the propagation delays from all inputs of the gate to the output of the gate to be identical. The total delay at the output \( T_{\text{out},i} \), however, is still only bounded from below by the precedence constraints. We therefore introduce the following constraints:

\[
\forall j \in \text{pred}(i) \quad T_{\text{out},i} \geq T_{j,i} + t_{p,i} \tag{3.10}
\]
So, we have to include $T_{\text{out},i}$ in the formulation as well to provide an upper bound. We write equation 3.10 to become equation 3.11, where $T_{j,i}$ is the total delay time at input $j$ and $T_{\text{out},i}$ is the total delay at the output.

$$\forall j \in \text{pred}(i) \ T_{j,i} \geq T_{\text{out},i} - 2t_{p,i} \quad (3.11)$$

The formulation in equation 3.11 makes the formulation in equation 3.9 obsolete, because the $T_{\text{out},i}$ is common to all inequalities with the input times $T_{j,i}$ for a gate.

We now have to adapt the gate sizing problem statement to incorporate the additional constraints. We also have to address the problem of solving the adapted gate sizing problem.

### 3.11 Gate sizing for minimal power while removing glitches

Equation 3.12 gives the gate sizing formulation for minimum power while removing glitches. We have copied the formulation for gate sizing for minimal power of equation 3.8, while adding the additional constraints of equation 3.11 to remove glitches.

$$\text{minimize } \sum_{i \in \text{gates}} c_i S_i \quad (3.12)$$

for all gates $i$ subject to

- $t_{p,i}S_i = t_{\text{int},i}S_i + c \cdot (C_{\text{wire},i} + \sum_{j \in \text{suc}(i)} C_{\text{inj},j}S_j)$
- $T_{\text{out},i} \geq T_{j,i} + t_{p,i}$ for all inputs $j$
- $T_{j,i} \geq T_{\text{out},i} - 2t_{p,i}$ for all inputs $j$
- $T_{\text{max}} \geq T_{\text{out},i}$
- $1 \leq S_i \leq \text{limit}$

Note that we have again multiplied each SGPD by $S_i$ to minimize the number of nonlinear terms. The formulation in equation 3.11 makes the formulation in equation 3.10 obsolete, because the $T_o$ is common to all inequalities with the input times $T_j$ for a gate. Therefore, only equation 3.11 will appear in the gate sizing formulation of equation 3.12. We will solve equation 3.12 by using LANCELOT.

We can see that the constraints in equation 3.12 are linear except for the term $t_{p,i}S_i$ in the constraint for the SGPDs. We should also note that this nonlinear
term is in an equality constraint. The aforementioned equality constraint and the constraint to remove glitches make that the gate sizing formulation in equation 3.12 is not posynomial (see section 2.8). The problem is also not convex, so a linearization of the SGPDs as in section 3.6 is also out of the question. Also, for specific instances the formulation according to equation 3.12 can be infeasible if excessive differences in propagation delays exist. Adding buffers alleviates such a problem.

The gate sizing formulation of equation 3.12 bares a resemblance to one in [Sath95] for the combined gate sizing and clock skew optimization. The resemblance is in the fact that there are also upper and lower bounds on the propagation delays through the circuit, as is the case in equation 3.12. The formulation in [Sath95] is solved by iteratively solving a linear and a geometric program. The paper does not give any clue about the convergence of this iteration. The approach described in [Sath95] does not guarantee that after solving the linear program all constraints of the geometric program are satisfied and vice versa. So while it is conceivable to use LANCELOT to solve the combined gate sizing and clock skew optimization problem, we therefore cannot use the iterative procedure suggested by [Sath95] to solve the problem posed in equation 3.12.

Setting up an experiment to demonstrate removing glitches through gate sizing is not as straightforward as for gate sizing as described in the previous sections. Because of the two-sided timing constraints it is not always possible to satisfy all the constraints in a general case. Propagation delays through the circuit may differ to such a large extent that the range of gate sizes permitted is never large enough to compensate for these differences. Also the timing constraints to two different gates in the fanout cone of a certain gate may be such that the timing constraints of both gates can never be satisfied.

In order to ensure the two cases described in the previous paragraph do not occur, we have to make sure beforehand that the timing constraints are feasible. One way to ensure this is by technology mapping in such a way that no large differences in propagation delays are possible. This is still an open problem. Another way is to insert buffers to delay fast paths and to speed up slow paths. We pursue the option of buffer insertion. The buffers we introduce are also sizable gates.

We now preprocess the circuit. We add buffers to equalize the logic level count along all paths. This ensures that the only difference in propagation delay is due to different loads to be driven by different gates. We are very likely able to even out those delays through gate sizing. Figure 3.3 illustrates the way in which we introduce the additional buffers. The triangles represent non–inverting buffers.
We introduce a buffer for each logic level where there is no gate (see figure 3.3). In the resulting circuit there is a gate in every path at each logic level. The next problem is to size for differences in load capacitance.

We will now perform a set of experiments to show the applicability of the method. We size for minimal power under the additional constraints posed to eliminate glitches, but no maximum propagation delay for the circuit is posed. The results of the experiments are presented in table 3.16. We use the benchmark circuits of the previous experiments, yet now the number of gates is larger as a consequence of the inserted buffers.
**Table 3.16** Sizing for minimal power $\sum_{i \in \text{gates}} c_i S_i$ while eliminating glitches

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>power (pJ)</th>
<th>memory (Mb)</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>1124</td>
<td>3742.53</td>
<td>2.8</td>
<td>1 m 51.0 s</td>
</tr>
<tr>
<td>C499</td>
<td>1412</td>
<td>5604.76</td>
<td>3.8</td>
<td>9 m 1.4 s</td>
</tr>
<tr>
<td>C880</td>
<td>1130</td>
<td>3904.22</td>
<td>2.8</td>
<td>6 m 34.4 s</td>
</tr>
<tr>
<td>C1355</td>
<td>1788</td>
<td>7898.99</td>
<td>7.0</td>
<td>20 m 19.5 s</td>
</tr>
<tr>
<td>C1908</td>
<td>5047</td>
<td>30926.85</td>
<td>11.2</td>
<td>220 m 4.9 s</td>
</tr>
<tr>
<td>C2670</td>
<td>3606</td>
<td>24553.80</td>
<td>8.4</td>
<td>69 m 27.1 s</td>
</tr>
</tbody>
</table>

We have not presented results for C3540 to C7552 since these circuits become considerably larger than their originals as a consequence of the buffer insertion. The example C1908 with the buffers is already larger than the example C7552 without buffers. Sizing C3540 to C7552 with buffers inserted takes an excessive amount of time. We have observed for the experiments of table 3.16 that most of the sizing factors are larger than their minimum of 1. So gate sizing is still necessary after buffer insertion in order to balance delays along different paths in the circuits.

We have also performed the sizing experiment with the same benchmark circuits without buffer insertion. The timing constraints could not be satisfied for any of the benchmark circuits. So changing the structure of the circuit, in our experiment by buffer insertion, but possibly through a different mapping, is necessary in order to satisfy all timing constraints to avoid all glitches. We therefore conclude that both buffer insertion and gate sizing are necessary to remove glitches.

Now that we can equalize delays along different paths in the circuit through the combination of buffer insertion and gate sizing, it is reasonable to assume a switching activity for each signal in the circuit as calculated under a zero delay model.

### 3.12 Discussion

We have looked into modeling a sizable CMOS–gate and discussed all effects sizing has on the performance (propagation delay), capacitances, both internal and external, and resistances. We have discussed how to model power dissipation as an objective function. We are able to take into account capacitances as well as switching activity. The incorporation of switching activity, however, is still limited to a zero delay model. Short–circuit power
dissipation can also be incorporated in the model. We have shown the terms for dynamic power dissipation and short-circuit power dissipation are dependent on the speed factor of the gate.

Also, we have presented an overview, including experiments on benchmark circuits, of several mathematical programming approaches to the gate sizing problem. From the comparison we conclude that for straightforward gate sizing an approach that uses a PWL approximation of the SGPDs is by far the fastest and the most scalable, provided an interior point solver is used. An approach using a geometric programming solver has its limitations both relative to memory usage and run-time. The general nonlinear programming package LANCELOT is also able to deal with all the benchmark circuits. It can handle the nonlinear SGPDs directly. However, it exhibits larger runtimes.

We have also shown that a zero delay model assumption for the calculation of the switching activity of the output of the gate before sizing can be accurate, provided after sizing glitches do not occur in the circuit. We can ensure this by introducing proper timing constraints. We have also adjusted a gate sizing formulation to include these additional constraints and even out small differences in propagation delays. We are able to solve this problem using LANCELOT. We have also introduced buffers into the circuit to even out excessive differences in propagation delays. We successfully demonstrated the combined approach of buffer insertion and gate sizing to remove glitches while minimizing power.
Chapter 4  Upper Bound Maximum Power

4.1 Introduction

Estimating the maximum power dissipation in VLSI circuits is considered to be important in connection with the reliability of circuits. Underestimating, the maximum power dissipation greatly reduces the reliability of the circuit. Because we do not want to underestimate the maximum power in this case we are interested in an upper bound. The estimation involves searching for two consecutive input vectors such that the transition maximizes the power dissipated by the circuit. The related problem of estimating the maximum current is of importance to the design of power and ground lines in a VLSI circuit.

We present a novel linear programming method, which estimates an upper bound on the maximum power. In order to establish a linear program we distinguish four scenarios at the time of a possible switch. Our approach is able to deal with arbitrary delays and accepts arbitrary power models for a switch of a gate.

This chapter is organized as follows: First we will look at related work in section 4.2. Then we will establish the problem of estimating the maximum power for CMOS circuits in section 4.3. In section 4.4 we explain the way we transform the maximum power estimation problem into a linear program. We will look at an example in section 4.5. Section 4.6 presents some experiments and results for the upper bound estimation of the maximum. We go on to extend the upper bound maximum power estimation method to estimating an upper bound on the maximum current in section 4.7. Finally, in section 4.8 we draw some conclusions.

4.2 Related work

Several approaches to estimating the maximum power in a circuit have been proposed in the past. A “weighted max–satisfiability” approach is proposed in [Deva92]. This method, according to the paper, tends to overestimate the actual power consumption. The largest example on which results are
reported has 703 gates. Most examples are much smaller. In [Krip92] and [Krip93] an upper bound estimation method for the related problem of estimating the maximum current is described using the propagation of “signal uncertainty waveforms”. A “signal uncertainty waveform” consists of an enumeration of time intervals with probabilities attached for each of the four transitions stable high, stable low, rising and falling in which that transition might occur. This approach cannot handle correlations between signals at all in [Krip92] and only partially in [Krip93]. So the method may grossly overestimate the maximum current since it doesn’t recognize invalid combinations of signals. The method described in [Teng95] is also a “signal uncertainty waveform” technique. In this paper the maximum transition count at the output of each gate is estimated. So the method considers only the output of one gate at a time and does not consider weighing the gates. The method suffers from inaccuracy for large circuit depths. The susceptibility for large circuit depths seems to be a common problem for all upper bound maximum power and current estimation approaches. A large circuit depth is a problem as the worst case number of possible switching events grows exponentially with the number of logic levels. The work we present in this chapter is based on our papers [Jac97a] and [Jac99b].

Next to the upper bound maximum power estimate, it is also interesting to calculate a lower bound on the maximum power to obtain a feeling of the amount of overestimation in case of the upper bound and the amount of underestimation in case of the lower bound. We will discuss estimating a lower bound on the maximum power in chapter 5. Then we will also discuss in more detail the existing work on the estimation of a lower bound on the maximum power.

4.3 Problem formulation

In this section we first describe an efficient way to calculate and store all the time–instances at which a gate can switch. Next, we discuss some aspects of power consumption of CMOS circuits and then the problem of maximum power estimation in these circuits. We will consider a single clock cycle.
4.3.1 Calculating and storing the switching events

![Diagram of time events](image)

<table>
<thead>
<tr>
<th>t</th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
<td>↑</td>
<td>1</td>
<td>↓</td>
</tr>
</tbody>
</table>

**Figure 4.1.** Storing time-entries for upper bound maximum power estimation

To be able to calculate the number of transitions that occur at the output of each gate we have to calculate the time-instances at which each gate could make a transition as a result of the propagation of the input signals along different paths in the circuit. We store, in an ordered list, the time-instances at which the output of a gate could possibly make a transition. We call an entry in our list a time-entry. We call the list a time-entry list. We introduce n(g), the number of entries in the list for a gate g. For each primary input of the circuit the time-entry list has one time-entry for time-instant zero. For all other gates we merge the time-entry lists with time-instances of the fanin gates, keeping them ordered and adding the propagation delay of the gate. This results in a time-entry list with time-instances of all possible output transitions of the gate. Figure 4.1 gives an example. We see that the signal does not necessarily have to make a transition, but can also remain constant as is the case here for t₂ and t₄. This is an approach which differs from the “time-sequences” used in the estimation of a lower bound in [Wan97]. This paper considers the time-instances at which a new logic level might be assumed. So the use of “time-sequences” results in one more entry into the list than the use of a time-entry list.

4.3.2 Power dissipation in CMOS circuits

The power used in a CMOS circuit depends on both the capacitance being switched and the number of transitions. We only take into account dynamic power dissipation (see section 2.3.1) in the calculation of the amount of energy dissipated in one transition, since this is the major source of power dissipation in CMOS circuits. We first look into the energy dissipated due to one charging transition. We assume a gate model as in figure 3.1. We now
denote the resistance of the PMOS transistor network \( R_{\text{PMOS}} \) by R. We also denote the total of the capacitances and \( \sum_{j \in \text{successor(gate)}} C_{\text{inj}} \) driven by the gate by C. We assume both R and C to be linear. A rising transition at the output of the gate involves charging the capacitance C through resistor R. We also assume that the capacitance C was initially completely discharged with a potential of 0 V and will eventually be completely charged to \( V_{dd} \). Now if the gate switches, the NMOS transistor network stops conducting and the PMOS network starts conducting, we can view this as applying an input step \( V_{dd} \) at time 0. The voltage \( V_{out} \) at the output of the gate is now given by:

\[
V_{out}(t) = V_{dd}(1 - e^{-\frac{t}{RC}}) \tag{4.1}
\]

The voltage difference over the resistor is then:

\[
V_R(t) = V_{dd} - V_{out}(t) = V_{dd}e^{-\frac{t}{RC}} \tag{4.2}
\]

The current through the resistor is:

\[
I_R(t) = \frac{V_R(t)}{R} = \frac{V_{dd}}{R}e^{-\frac{t}{RC}} \tag{4.3}
\]

Now the energy dissipated in the resistor, assuming the capacitance is eventually charged to \( V_{dd} \), is given by:

\[
E = \int_{0}^{\infty} I_R(t)V_R(t)dt = \int_{0}^{\infty} \frac{V_{dd}^2}{R}e^{-\frac{2t}{RC}}dt = \frac{1}{2} \cdot C \cdot V_{dd}^2 \tag{4.4}
\]

A similar derivation can be given to compute the energy due to discharging the capacity driven by the gate. So the energy consumption at a gate \( g \) due to one transition is:

\[
E(g) = \frac{1}{2} \cdot C_{\text{load}}(g) \cdot V_{dd}^2 \tag{4.5}
\]

For some gate \( g \) we now consider the time-entry list. The number of actual transitions, \( N(g) \), at the output of a gate \( g \) is given by:

\[
N(g) = \sum_{i=0}^{n(g)-1} (f_g(i_-) \oplus f_g(i_+)) \tag{4.6}
\]

In equation (4.6) \( f_g(i_-) \) is the logic level at the output of the gate just before the possible switching event and \( f_g(i_+) \) just after. This way we only count
those time-entries yielding a value change at the output of gate \( g \). The total power dissipation of a circuit in a clock cycle is then:

\[
P_{\text{total}} = \sum_{g \in \text{gates}} E(g) \cdot N(g)
\]

(4.7)

### 4.3.3 Maximum power estimation of CMOS circuits

In order to model the maximum power dissipation in CMOS circuits we have to make some assumptions. First of all, we assume that all switches of the primary inputs are synchronized on the rising clock edge of each clock cycle. So we have one input vector transition at the start of the clock cycle. Secondly, we assume that given some gate each switching event consumes the same amount of energy. This is also for the purpose of the experiments. Should more accurate energy information be known, we can easily incorporate this in our formulation. Thirdly, we do not yet take into account the filtering effect of CMOS gates.

### 4.4 Linear programming model

While normal Boolean logic distinguishes two logic levels, we distinguish four different situations at a time-instance a possible transition takes place. We define the following four different scenarios:

- no transition takes place and the logic level stays 0 (we denote this by \( \varnothing \))
- no transition takes place and the logic level stays 1 (we denote this by \( \uparrow \))
- a transition from 0 to 1 takes place (we denote this by \( \uparrow \))
- a transition from 1 to 0 takes place (we denote this by \( \downarrow \))

We will establish an Integer Linear Program (ILP) to compute the exact maximum power for a given circuit. We will therefore first introduce the objective function. Then we will discuss some necessary constraints, which stem from the use of the four scenarios. Subsequently we will deal with inverters. The main work of this section lies in dealing with the constraints for AND-gates. For an AND-gate we establish constraints for all combinations of scenarios of inputs and output that violate the behavior of an AND-gate. We will conclude this section with a discussion of dealing with more complex gates.

For each scenario for each the time-entry and for each gate we introduce a variable \( x_{a,b,c} \). In \( x_{a,b,c} \) the \( a \) identifies the gate to which the variable \( x_{a,b,c} \) belongs, \( b \) identifies the time-entry out of the time-entry list of gate \( a \) and \( c \) identifies one of the four scenarios. For the ILP-formulation the variable \( x_{a,b,c} \) is defined on \( \{0,1\} \). The ILP is however very difficult to solve (see section
2.6.5). We therefore relax the ILP to a linear program (LP). In the LP the variable $x_{a,b,c}$ is a real variable defined on $[0,1]$. The relaxation to a LP results in an overestimation of the maximum power dissipation. So the method we describe is an upper bound method.

### 4.4.1 Objective function

The objective function for maximizing the power consumption contains the variables that denote a possible switch, being $x_{g,i,\downarrow}$ and $x_{g,i,\uparrow}$. The objective function, which we want to maximize, is:

$$
\sum_{g \in \text{gates}} \sum_{i=0}^{n(g)-1} \frac{1}{2} V_{dd}^2 C_{load}(g)(x_{g,i,\downarrow} + x_{g,i,\uparrow})
$$

So for each gate and for each time-entry of that gate, we enter the variables for the rising and the falling scenario in the objective function weighed by the capacitance switched and $\frac{1}{2} V_{dd}^2$.

### 4.4.2 One of four scenarios

In equation 4.9 $g$ denotes the gate and $i$ the time-entry. For the variables to the four scenarios belonging to gate $g$ and time-entry $i$ the following constraints hold:

$$
\forall_{g \in \text{gates}} \forall_{0 \leq i < n(g)} (x_{g,i,0} + x_{g,i,\downarrow} + x_{g,i,\uparrow} + x_{g,i,1} = 1)
$$

The constraints of equation 4.9 state that for a given time-entry for a given gate one and only one of the four scenarios can apply. We can change this equality constraint into an inequality constraint. The statement then changes to no more than one of the four scenarios can apply. This will give a less accurate estimate, as the choice for none of the scenarios is left open. This additional choice for a given time-entry may leave open more possibilities for other time-entries, leading to more over-estimation. If equation 4.9 is an equality constraint the linear program may be more difficult to solve [Nem88] than if it is an inequality constraint.

### 4.4.3 Precedence constraints

We also have to introduce precedence constraints to ensure that the logic level does not change inbetween time-entries. For example, if we have a rising transition for gate $g$ at time-entry $i - 1$, there cannot be a rising transition immediately afterwards at time-entry $i$, as the logic level after the first
transition is not the same as before the second transition, and there are no transitions inbetween. The corresponding constraints are:

$$\forall g \in \text{gates} \forall 0 \leq i < n(g)(x_{g,i,1} + x_{g,i,\uparrow} + x_{g,i+1,\uparrow} + x_{g,i+1,0} = 1) \quad (4.10)$$

$$\forall g \in \text{gates} \forall 0 \leq i < n(g)(x_{g,i,0} + x_{g,i,\downarrow} + x_{g,i+1,\downarrow} + x_{g,i+1,1} = 1)$$

It is also possible to make equation 4.10 an inequality constraint. This gives a looser upper bound estimate, but may be a less difficult problem to solve.

4.4.4 Inverters

For inverters and inverted functions, we do not introduce additional real variables, but replace the variables in the constraints according to table 4.1. We add the load for the inverter to the predecessor gate of the inverter in the objective function.

**Table 4.1** Replacement of variables for INVERT

<table>
<thead>
<tr>
<th>Q</th>
<th>$x_{g,0}$</th>
<th>$x_{g,\uparrow}$</th>
<th>$x_{g,\downarrow}$</th>
<th>$x_{g,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>$x_{g,1}$</td>
<td>$x_{g,\downarrow}$</td>
<td>$x_{g,\uparrow}$</td>
<td>$x_{g,0}$</td>
</tr>
</tbody>
</table>

4.4.5 AND–gate

Equations 4.12 to 4.21 list all the constraints for all time–entries for all inputs of one AND–gate. These constraints are such that they cover those situations of combinations of input and output scenarios that cannot exist for an AND–gate. In equations 4.12 to 4.21 $x_{a,b,c}$ again denotes the continuous variable for gate $a$ at time–entry $b$ for scenario $c$. In the following equations $\text{pred}(g)$ denotes the set of inputs to gate $g$ and $t_g$ the propagation delay of gate $g$.

We have to relate a time–entry of an input to the corresponding time–entry of the output of the gate. We therefore introduce a function $T_g(i)$, which returns the time–instance corresponding to time–entry $i$ of gate $g$. We also introduce a function $T_g^{-1}(t)$, which returns the time–entry corresponding to time $t$ for gate $g$. We can now compute the time–entry $y_g$ at the output for time–entry $i$ of input $p$ by:

$$y_g = T_g^{-1}(T_p(i) + t_g) \quad (4.11)$$

We will briefly explain the meaning of each equation. In these explanations we make a distinction between the situation that an input or output “is” zero or one, and the situation that it “stays” zero or one. We use “is” for the situation that denotes a Boolean level at a time where a transition cannot
occur. So the Boolean level is determined by the scenario immediately prior to the current time. We therefore have to account for a falling or stable zero scenario appearing in the constraint for a Boolean level zero and a rising or stable one scenario for a Boolean level one. We use “stay” for the situation that denotes a Boolean level at a time where a transition could have occurred, but doesn’t. In this case we have to account for a stable zero scenario for a Boolean level zero and a stable one scenario for a Boolean level one in the constraint.

If an input \( p \) to an AND–gate \( g \) stays zero or is falling (denoted by \( x_{p,i,0} \) and \( x_{p,i,\downarrow} \)), the output of the AND–gate will never be rising (denoted by \( x_{g,y_p,\uparrow} \)):

\[
\forall_{p \in \text{pred}(g)} \forall_{0 \leq i < n(p)} (x_{p,i,0} + x_{p,i,\downarrow} + x_{g,y_p,\uparrow} \leq 1)
\]  

(4.12)

If an input \( p \) to an AND–gate \( g \) stays zero or is rising (denoted by \( x_{p,i,0} \) and \( x_{p,i,\uparrow} \)), the output of the AND–gate will never be falling (denoted by \( x_{g,y_p,\downarrow} \)):

\[
\forall_{p \in \text{pred}(g)} \forall_{0 \leq i < n(p)} (x_{p,i,0} + x_{p,i,\uparrow} + x_{g,y_p,\downarrow} \leq 1)
\]  

(4.13)

If an input \( p \) to an AND–gate \( g \) stays zero or is falling or rising (denoted by \( x_{p,i,0}, x_{p,i,\downarrow} \) and \( x_{p,i,\uparrow} \)), the output of the AND–gate will never stay one (denoted by \( x_{g,y_p,\downarrow} \)):

\[
\forall_{p \in \text{pred}(g)} \forall_{0 \leq i < n(p)} (x_{p,i,0} + x_{p,i,\downarrow} + x_{p,i,\uparrow} + x_{g,y_p,\downarrow} \leq 1)
\]  

(4.14)

In the next constraints we have to relate a time–entry of the output of the gate \( p \), which is an input of gate \( g \), to a time–entry of the output of gate \( g \). In the following equation \( y_p \) is the time–entry of input \( p \), corresponding to time–entry \( i \) of the output of gate \( g \).

\[
y_p = T_p^{-1}(T_g(i) - t_g)
\]  

(4.15)

If an input is zero (denoted by \( x_{p,y_p,0} \) and \( x_{p,y_p,\downarrow} \)), the output of an AND–gate will never be rising or falling or stay one (denoted by \( x_{g,i,\uparrow}, x_{g,i,\downarrow} \) and \( x_{g,i,\downarrow} \)):

\[
\forall_{0 \leq i < n(g)} \forall_{p \in \text{pred}(g)} \forall_{T_p(j) = T_g(i) - t_j}
\]

\[
(x_{p,y_p,0} + x_{p,y_p,\downarrow} + x_{g,i,\uparrow} + x_{g,i,\downarrow} + x_{g,i,\downarrow} \leq 1)
\]  

(4.16)

In the next constraint we again have to relate a time–entry of the output of the gate \( q \), which is an input of gate \( g \), to a time–entry of the output of gate \( g \). Similarly to equation 4.15 we give:

\[
y_q = T_q^{-1}(T_g(i) - t_g)
\]  

(4.17)

If one possibly switching input is either stable one, rising or falling (denoted by \( x_{q,y_q,1}, x_{p,y_p,\uparrow} \) and \( x_{p,y_p,\downarrow} \)), the remaining possibly switching inputs are
stable one (denoted by \( x_{q,Y_{q}^1} \)) and the non–switching inputs are all one (denoted by \( x_{q,Y_{q}^1} \) and \( x_{q,Y_{q}^1} \)), the output of the AND–gate can never stay zero (denoted by \( x_{g,i,0} \)):

\[
\forall 0 \leq i < n(g) \forall p \in \{p \in \text{pred}(g) \mid \forall j, T_d(j) = T_d(i) - t_e\} \left\{ x_{g,i,0} + \sum_{q \in \text{pred}(g)} x_{q,Y_{q}^1} + \sum_{q \in \{q \in \text{pred}(g) \mid \forall j, T_d(j) = T_d(i) - t_e\}} x_{q,Y_{q}^1} + x_{p,Y_{p}^r} + x_{p,Y_{p}^f} \leq 1 \right\}
\] (4.18)

If all non–switching inputs are one (denoted by \( x_{p,Y_{p}^1} \) and \( x_{p,Y_{p}^f} \)) and all switching inputs are falling (denoted by \( x_{p,Y_{p}^r} \)), the output of the AND–gate can never stay zero (denoted by \( x_{g,i,0} \)):

\[
\forall 0 \leq i < n(g) \left\{ x_{g,i,0} + \sum_{[p \in \text{pred}(g) \mid \forall j, T_d(j) = T_d(i) - t_e]} (x_{p,Y_{p}^1} + x_{p,Y_{p}^f}) + \right. \\
\left. \sum_{[p \in \text{pred}(g) \mid \exists j, T_d(j) = T_d(i) - t_e]} x_{p,Y_{p}^f} \leq I(g) \right\}
\] (4.19)

In the next constraint the number of inputs to the gate plays a part. We therefore introduce \( I(g) \), which is the number of inputs of gate \( g \).

If all non–switching inputs are one (denoted by both \( x_{p,Y_{p}^1} \) and \( x_{p,Y_{p}^f} \), because the logic level after, not the scenario at the possible switching time is important) and all possibly switching inputs are rising (denoted by \( x_{p,Y_{p}^r} \)), the output of the AND–gate can never stay zero (denoted by \( x_{g,i,0} \)):

\[
\forall 0 \leq i < n(g) \left\{ x_{g,i,0} + \sum_{[p \in \text{pred}(g) \mid \forall j, T_d(j) = T_d(i) - t_e]} (x_{p,Y_{p}^1} + x_{p,Y_{p}^f}) + \right. \\
\left. \sum_{[p \in \text{pred}(g) \mid \exists j, T_d(j) = T_d(i) - t_e]} x_{p,Y_{p}^r} \leq I(g) \right\}
\] (4.20)

In the next constraint the number of possibly simultaneously switching inputs to the gate at a certain time–entry plays a part. We therefore introduce \( S_i(g) \), which is the number of simultaneously switching inputs of gate \( g \) at time–entry \( i \).
If all possibly switching inputs stay one (denoted by $x_{p,y_p}$), the output of the AND–gate will never be falling or rising (denoted by $x_{g,i,\downarrow}$ and $x_{g,i,\uparrow}$):

$$\forall 0 \leq i < n(g) \left\{ \sum_{[p \in \text{pred}(g) \mid \exists T_g(i) = T_g(i) - t_g]} x_{p,y_p} + x_{g,i,\downarrow} + x_{g,i,\uparrow} \leq S_i(g) \right\} \quad (4.21)$$

The index $y_p$ is the index of the time–entry for gate $p$ at time $T_g(i) - t_g$.

### 4.4.6 OR–gate

To describe an OR–operation we make use of DeMorgan’s rule and the fact that we already have complementary variables for all gates at every time–entry available. So, we replace all real variables of equations 4.12 to 4.21 according to table 4.1 to get the set of constraints for an OR–gate.

### 4.4.7 Complex gates

Complex gates can be decomposed into several AND–, OR– and INVERT–operations and can therefore be modelled as several sets of constraints. Of course, the internal variables of a complex gate can not be found in the objective function. In case of a complex gate we only add the propagation delay of the gate to each entry in the time–entry list for the output of the gate. For internal Boolean relations we do not need to take account of propagation delays, so the following relation between time–entries holds:

$$y_p = T_p^{-1}(T_g(i)) \quad (4.22)$$

Internally we do, of course, merge the lists of time–entries. We also add precedence constraints and a constraint on the simultaneous occurrence of the four scenarios for the variables internal to the complex gate.
4.5 Example

We now look at an example. Figure 4.2 gives a small network. The gates in this network are given by the capital letters A to E. The primary inputs are given by the lowercase letters a to d. The values between braces give the time-entries assuming a unit delay. In equation 4.23 the corresponding objective function for upper bound estimation of the maximum power is given. Equation 4.23 gives the constraints to the problem, which are ordered according to their origin. Please note that no additional variables are used for inverters A and D. We have left out the constraints for gate B for time 1, for gate C for time 0, and for gate E for times 0,1, and 2, as they are similar to the set of constraints for gate B at time 0.
Maximize: $C_{\text{load}}(E) \cdot (x_{E,0,\uparrow} + x_{E,0,\downarrow} + x_{E,1,\uparrow} + x_{E,1,\downarrow} + x_{E,2,\uparrow} + x_{E,2,\downarrow})$

$+ (C_{\text{load}}(B) + C_{\text{load}}(D)) \cdot (x_{B,0,\uparrow} + x_{B,0,\downarrow} + x_{B,1,\uparrow} + x_{B,1,\downarrow})$

$+ C_{\text{load}}(C) \cdot (x_{C,0,\uparrow} + x_{C,0,\downarrow}) + C_{\text{load}}(A) \cdot (x_{a,0,\uparrow} + x_{a,0,\downarrow})$

One of four scenarios

\begin{align*}
\text{Precedence constraints} & \\
\text{Constraints for gate B at time–entry 0} & \\
\text{Constraints for gate C at time–entry 0} & \\
\text{Constraints for gate E at time–entry 0} & \\
\text{Constraints for gate E at time–entry 1} & \\
\text{Constraints for gate E at time–entry 2} & \\
\end{align*}

The number of variables needed to describe a maximum power estimation problem grows quite rapidly as equation 4.23 shows. However, the problem remains tractable as is shown in section 4.6. As is easily seen in this example the linear program has a very specific form, where all variables in the constraints only appear with a constant 1. The right hand side of the constraints is also an integer constant, though not necessarily 1. However the form of our linear programming description is not so special that it always has an integral solution [Nem88]. An integral solution would mean an exact solution. The reconstruction of an input vector which maximizes the power is only possible (and trivial) in case of an integral solution. In general an
upper bound estimate does not have corresponding input vectors, as an overestimation can not be realized in the circuit.

4.6 Experimental results

For our experiments we used estimated capacitances consisting of floating point values. We initially assume the constraints of equations 4.9 and 4.10 to be equality constraints. We mark the columns for these results with \( = \). We also present results where these constraints are changed to inequality constraints. These are presented in the columns marked with \( \leq \). The first set of experiments is performed on benchmark circuits with unconstrained logic depth under a unit delay model. We are able to calculate an upper bound power estimate for six out of the ten benchmarks (table 4.2). The columns marked CPU give the CPU–time consumed and the columns marked % give our upper bound maximum power estimate divided by the power estimate for a transition occurring for every time–entry of every gate in the circuit. The smaller the number is, the tighter is our upper bound estimate of the maximum power. We also give some data about the benchmarks circuits in this table. The linear program for the four largest benchmarks (C3540, C5315, C6288 and C7552) becomes too large to handle due to their enormous (and unrealistic) logic depth (denoted by #levels in table 4.2 and 4.3).

**Table 4.2** Improvement of the upper bound maximum power estimates and CPU times for the original circuits under UNIT delay model

<table>
<thead>
<tr>
<th></th>
<th>#input</th>
<th>#gate</th>
<th>#levels</th>
<th>#entry</th>
<th>#var</th>
<th>#constr</th>
<th>( \leq )</th>
<th>%</th>
<th>CPU</th>
<th>%</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>36</td>
<td>240</td>
<td>26</td>
<td>1257</td>
<td>5028</td>
<td>16559</td>
<td>94.8</td>
<td>284 s</td>
<td>68.5</td>
<td>412 s</td>
<td></td>
</tr>
<tr>
<td>C499</td>
<td>41</td>
<td>436</td>
<td>28</td>
<td>3075</td>
<td>12300</td>
<td>37481</td>
<td>96.9</td>
<td>86 s</td>
<td>84.8</td>
<td>102 s</td>
<td></td>
</tr>
<tr>
<td>C880</td>
<td>60</td>
<td>523</td>
<td>27</td>
<td>1978</td>
<td>7912</td>
<td>23520</td>
<td>92.2</td>
<td>132 s</td>
<td>80.8</td>
<td>221 s</td>
<td></td>
</tr>
<tr>
<td>C1355</td>
<td>41</td>
<td>644</td>
<td>29</td>
<td>4675</td>
<td>18700</td>
<td>59009</td>
<td>95.5</td>
<td>429 s</td>
<td>85.1</td>
<td>461 s</td>
<td></td>
</tr>
<tr>
<td>C1908</td>
<td>33</td>
<td>1100</td>
<td>52</td>
<td>7761</td>
<td>31044</td>
<td>107627</td>
<td>94.1</td>
<td>195 m</td>
<td>72.9</td>
<td>262 m</td>
<td></td>
</tr>
<tr>
<td>C2670</td>
<td>233</td>
<td>1987</td>
<td>54</td>
<td>5491</td>
<td>21964</td>
<td>67570</td>
<td>90.5</td>
<td>26 m</td>
<td>77.2</td>
<td>44 m</td>
<td></td>
</tr>
</tbody>
</table>

For the second set of experiments, we therefore make the benchmarks more realistic by pipelining them in order to achieve a maximal logic depth of 10 for each stage. Table 4.3 depicts the number of inputs, gates, logic levels, time–entries, variables and constraints for the circuits used in the experiments of table 4.4.
We conduct the experiments using both a unit delay model with integer delay values (denoted in tables 4.3 and 4.4 with unit) and a real delay model with floating point values for the delays (denoted with float). In table 4.4 we again present CPU–times and the improvement in the upper bound estimate. Using a real delay model instead of a unit delay model results in a much larger number of time–entries and therefore in a much larger number of variables (see table 4.3) and constraints. Solving the maximum power estimation problem for an arbitrary (real) delay model is therefore more difficult.

We have conducted both sets of experiments described above also in the case where we change equations 4.9 and 4.10 into inequality constraints (see table 4.4 and 4.3 in the columns marked with \( \leq \)). This problem is numerically less difficult to solve (compare columns marked CPU), but gives a looser upper bound. All experiments are conducted on one 180 Mhz processor of a Hewlett Packard K260 with 512 MB of memory using LOQO (see sections 2.6 and 2.9).
TABLE 4.4 Improvement of the upper bound maximum power estimates and CPU times for the benchmark circuits with the logic depth limited to 10 per stage under both UNIT and FLOAT delay model

<table>
<thead>
<tr>
<th>Circuit</th>
<th>UNIT ≤%</th>
<th>CPU s</th>
<th>UNIT =%</th>
<th>CPU s</th>
<th>FLOAT ≤%</th>
<th>CPU s</th>
<th>FLOAT =%</th>
<th>CPU s</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>95.1</td>
<td>3</td>
<td>82.3</td>
<td>4</td>
<td>91.2</td>
<td>5</td>
<td>68.4</td>
<td>6</td>
</tr>
<tr>
<td>C499</td>
<td>98.7</td>
<td>7</td>
<td>82.3</td>
<td>8</td>
<td>96.3</td>
<td>8</td>
<td>81.0</td>
<td>9</td>
</tr>
<tr>
<td>CS80</td>
<td>89.4</td>
<td>7</td>
<td>77.5</td>
<td>10</td>
<td>89.9</td>
<td>31</td>
<td>67.3</td>
<td>34</td>
</tr>
<tr>
<td>C1355</td>
<td>92.4</td>
<td>11</td>
<td>78.4</td>
<td>16</td>
<td>91.1</td>
<td>11</td>
<td>77.4</td>
<td>15</td>
</tr>
<tr>
<td>C1908</td>
<td>90.6</td>
<td>14</td>
<td>76.6</td>
<td>20</td>
<td>88.6</td>
<td>20</td>
<td>68.1</td>
<td>27</td>
</tr>
<tr>
<td>C2670</td>
<td>89.2</td>
<td>16</td>
<td>82.8</td>
<td>20</td>
<td>87.1</td>
<td>23</td>
<td>77.8</td>
<td>28</td>
</tr>
<tr>
<td>C3540</td>
<td>87.7</td>
<td>37</td>
<td>72.3</td>
<td>46</td>
<td>87.0</td>
<td>53</td>
<td>65.1</td>
<td>62</td>
</tr>
<tr>
<td>C5315</td>
<td>89.1</td>
<td>73</td>
<td>80.6</td>
<td>98</td>
<td>86.7</td>
<td>118</td>
<td>71.2</td>
<td>152</td>
</tr>
<tr>
<td>C6288</td>
<td>90.6</td>
<td>111</td>
<td>83.9</td>
<td>145</td>
<td>88.0</td>
<td>291</td>
<td>75.8</td>
<td>392</td>
</tr>
<tr>
<td>C7552</td>
<td>88.6</td>
<td>103</td>
<td>77.9</td>
<td>126</td>
<td>83.8</td>
<td>178</td>
<td>67.9</td>
<td>182</td>
</tr>
</tbody>
</table>

In table 4.4 we can see that for our benchmark examples we are able to estimate an upper bound on the maximum power in a reasonable amount of time. The largest amount of CPU–time is used for C6288, which only needs 392 seconds. Comparing table 4.2 and 4.4 we can see that limiting the logic depth to a realistic value makes the problem of estimating an upper bound on the maximum power tractable. We can also see in the table that our method improves the upper bound estimate by at least 19%. Further comparisons on the quality of the estimates will be made in chapter 5.

4.7 Extension to maximum current estimation

We can extend the method for estimating the upper bound on the maximum power dissipation to calculate an upper bound on the maximum current. The maximum current is simply the maximum power dissipation addressed on a certain time–interval. We call this time–interval a “window”. The window is small compared to the clock–cycle–time of the circuit under consideration. We have to calculate the maximum current over a window, because in our model a transition incurs an instantaneous transfer of charge. If we do not use this window the current is a delta pulse, given our model.

So, in order to calculate an upper bound on the maximum current in a circuit, the maximum power estimation problem has to be solved repeatedly, for each window of switching events. Therefore we will include only the capacitance loads for switching events within the window in the objective function of
equation 4.8. The capacitance values for switching events outside the window will be taken to be zero.  

We will now investigate the number of solutions of the maximum power estimation problem required to estimate an upper bound on the maximum current. We will therefore tabulate the number of windows for several benchmark circuits under both unit and floating point delay values with limitations on the number of logic levels per stage. The results are presented in table 4.5. We also give the average and maximum number of possible switching events in a window. The window size is taken to be 1% of the clock–cycle–time of the circuit. We have chosen this window size, because it is smaller than the clock–cycle–time, and also about a factor 10 smaller than the gate propagation delays. This window size also seems of a reasonable order of magnitude over which to calculate the current.

**Table 4.5** The number of windows for maximum current estimation for unit and float delay values, with a logic depth limited to 10 per stage

<table>
<thead>
<tr>
<th></th>
<th>#input</th>
<th>#gate</th>
<th>#levels</th>
<th>UNIT</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#win</td>
<td># ave.</td>
<td># max</td>
<td>#win</td>
<td># ave</td>
<td># max</td>
<td>#win</td>
<td># ave</td>
<td># max</td>
<td></td>
</tr>
<tr>
<td>C432</td>
<td>36</td>
<td>240</td>
<td>26</td>
<td>8</td>
<td>53.5</td>
<td>86</td>
<td>113</td>
<td>4.5</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>C499</td>
<td>41</td>
<td>436</td>
<td>28</td>
<td>7</td>
<td>90.9</td>
<td>168</td>
<td>35</td>
<td>19.8</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>C880</td>
<td>60</td>
<td>523</td>
<td>27</td>
<td>9</td>
<td>103.2</td>
<td>150</td>
<td>234</td>
<td>7.7</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>C1355</td>
<td>41</td>
<td>644</td>
<td>29</td>
<td>9</td>
<td>135.6</td>
<td>240</td>
<td>59</td>
<td>21.7</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>C1908</td>
<td>33</td>
<td>1100</td>
<td>52</td>
<td>9</td>
<td>210.1</td>
<td>281</td>
<td>276</td>
<td>9.0</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>C2670</td>
<td>233</td>
<td>1987</td>
<td>54</td>
<td>9</td>
<td>364.4</td>
<td>502</td>
<td>262</td>
<td>14.8</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>C3540</td>
<td>50</td>
<td>2641</td>
<td>72</td>
<td>9</td>
<td>514.6</td>
<td>723</td>
<td>476</td>
<td>16.4</td>
<td>271</td>
<td></td>
</tr>
<tr>
<td>C5315</td>
<td>178</td>
<td>3624</td>
<td>63</td>
<td>9</td>
<td>694.9</td>
<td>947</td>
<td>478</td>
<td>21.4</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>C6288</td>
<td>32</td>
<td>2448</td>
<td>125</td>
<td>9</td>
<td>632.8</td>
<td>922</td>
<td>122</td>
<td>69.1</td>
<td>462</td>
<td></td>
</tr>
<tr>
<td>C7552</td>
<td>207</td>
<td>4925</td>
<td>53</td>
<td>9</td>
<td>1007.2</td>
<td>1419</td>
<td>543</td>
<td>24.1</td>
<td>376</td>
<td></td>
</tr>
</tbody>
</table>

As we can see for the benchmark circuits in table 4.5 the number of windows under the more realistic floating point delay model can be considerable. The amount of possible switching events per window is also too large to make estimating the current in a window less difficult than solving the linear program for upper bound maximum power estimation. So, at first sight the presented approach for estimating an upper bound on the maximum power cannot be converted to an approach for estimating an upper bound on the maximum current in a straightforward way. Not all is lost however. Usually a maximum current estimate is only interesting for a single row of cells (in a standard cell layout approach) sharing the same power and ground lines. Such a row of cells does not consist of more than a few hundred gates. So the number of windows to consider is limited. This together with the limitation
of the maximum logic depth per stage can go a long way in making the upper bound maximum current estimation problem tractable.

4.8 Discussion

In this chapter we have presented a method capable of estimating an upper bound on the maximum power of CMOS circuits. The result is useful for the design of power and ground lines. The method entails the construction and solving of a linear program. In order to make the problem tractable and more realistic, we have limited the logic depth per pipeline stage of the circuits. We have presented results on several benchmark circuits of up to a several thousand gates. We are able to solve even our largest benchmark using floating point delays within 7.5 minutes.

We also suggested the extension of the method for upper bound maximum power estimation to the problem of estimating an upper bound on the maximum current. This is a more difficult problem as it entails the repeated solving of a linear program similar to the estimation of the upper bound on the maximum power. We have suggested that this problem may also become tractable by making an additional assumption on the size of the circuit (number of gates) simultaneously under consideration.
Chapter 5

Lower Bound Maximum Power

5.1 Introduction

In the previous chapter we have introduced a method for estimating an upper bound on the maximum power dissipation in CMOS circuits. In this chapter we complement the upper bound power estimation method with a lower bound maximum power estimation method. The estimation involves searching for two consecutive input vectors which maximize the power dissipated by the circuit.

We propose a novel method to establish such an estimate. Our method translates the combinational problem of finding two consecutive input vectors maximizing the power dissipation to an optimization problem using real variables. The resulting problem comprises of a nonlinear objective function to be maximized, accompanied by both linear and nonlinear equality constraints. We are also able to retrieve the input vector transition corresponding to the lower bound maximum power estimate. Because the method renders a valid input vector pair to the power estimate, the power estimate is either equal to or lower than the maximum power dissipation of the circuit. The method can underestimate the maximum power, because a nonlinear optimization problem can get stuck in a local maximum.

This chapter is organized as follows. First we will look at previous and related work in section 5.2. Then we will establish the problem of estimating a lower bound on the maximum power for CMOS circuits in section 5.3. In section 5.4 we present an example of the nonlinear programming model for estimating a lower bound on the maximum power dissipation of a CMOS circuit. In section 5.5 we reiterate this nonlinear programming formulation and propose solving the nonlinear program using LANCELOT. We also explain the way in which we reconstruct the Boolean input vector pair inducing the estimated power dissipation since the solution provided by LANCELOT consists of reals. Section 5.6 presents some experiments and corresponding results for the lower bound estimation of the maximum power using this nonlinear programming solver. Finally, in section 5.7 we draw some conclusions.
5.2 Related work

We will now discuss existing work on estimating a lower bound on the maximum power of CMOS circuits. A number of methods for estimating a lower bound on the maximum power are described in [Krst97], [Man97], [Wan96] and [Wan97]. Of these, [Krst97] describes an Automated Test Pattern Generation (ATPG) method and a probabilistic method, [Man97] an ATPG based technique for maximizing weighted switching activity and [Wan96] another ATPG based technique.

The probability based approach of [Krst97] keeps a list of times for each gate also containing a probability (in [Krst97] called desirability) of this gate having a stable 0, stable 1, rising or falling transition at a given time. These desirabilities are propagated backwards into the circuit towards the primary inputs. For each of the primary inputs one can then determine which transition (stable 0, stable 1, rising or falling) is most likely (desirable) to maximize power.

[Wan97] describes a technique to estimate a lower bound on the maximum power dissipation based on unconstrained non-linear optimization. This technique maps the Boolean domain onto a continuous domain of real values in the range of [0, 1] and solves the nonlinear optimization problem. The method has just one, very large, nonlinear objective function, because every Boolean relation for the gates has to be included in the objective function. Our method, described in this chapter, uses nonlinear constraints for the Boolean relations for the gates. Also, [Wan97] uses a gradient based (first order derivative) heuristic to solve the unconstrained nonlinear optimization problem, rather than a nonlinear programming method using nonlinear equality constraints and explicit first and second order derivatives. The work we present in this chapter is based upon our paper [Jac98b].
5.3 Problem formulation

5.3.1 Calculating and storing switching events revisited

\[
\begin{array}{c|c|c|c|c|c}
& t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \\
\hline
-\infty & 0 & 1 & 0 & 0 & 1 & 0 \\
\end{array}
\]

**Figure 5.1.** Storing time–entries for lower bound maximum power estimation

In this section we establish the problem of lower bound maximum power estimation. The first step is accomplished by slightly adapting some structures introduced in the previous chapter. We again use the time–entry list. However, we now store the time at which the output of a gate takes on a new Boolean value instead of a scenario. For the primary inputs of the circuit the time–entry list therefore has two entries, one for the time–instance minus infinity and one for the time–instance zero, in that order. For the other gates we again merge the time–entry lists for the gates fanin, while keeping them ordered and while adding the propagation delay of the gate. This results in a list with time–instances at which the output of the gate reaches a new (Boolean) value. An example is given in figure 5.1. When we compare figure 5.1 to figure 4.1, we see the additional time–entry for minus infinity in figure 5.1. We also see that figure 5.1 stores a logic level, while figure 4.1 stores a scenario. The number of entries in the list for a gate g is given by \( n(g) \). Note that \( n(g) \) is one larger than in the previous chapter, because of the additional storing of the time–entry for minus infinity.

Because we have slightly changed our time–entry lists, we also have to make some changes to the definition of the number of transitions. The number of transitions \( N(g) \) at the output of a gate \( g \) in this chapter is given by:

\[
N(g) = \sum_{i=0}^{n(g)-1} (f_g(i) \oplus f_g(i + 1))
\]  
(5.1)
In equation 5.1 \( f_g(i) \) is the Boolean value at the output of gate \( g \) at time–entry \( i \). The total power consumption of a circuit again is:

\[
P_{\text{total}} = \sum_{g \in \text{gates}} E(g) \cdot N(g)
\]  

(5.2)

The energy \( E(g) \) dissipated due to one transition is defined in equation 4.5.

### 5.3.2 Transformation of Boolean functions into nonlinear constraints

Below we describe the transformation of the Boolean functions for the gates into nonlinear constraints containing continuous variables. This is a mapping from the Boolean domain \([0, 1]\) onto the continuous domain \([0, 1]\) of real numbers. For each time–entry \( i \) of each gate \( g \) in the network other than inverters and buffers, we have introduced \( f_g(i) \), the Boolean value at the output of the gate. We also introduce \( \overline{f_g(i)} \), the Boolean value of its complement. For the Boolean value \( f_g(i) \) we introduce the corresponding real variable \( x_{g,i} \), which has a range of \([0, 1]\). For the complementing Boolean value \( \overline{f_g(i)} \) we introduce the corresponding real variable \( z_{g,i} \), also in the range of \([0, 1]\). We now have to ensure that \( x_{g,i} \) and \( z_{g,i} \) cannot both be simultaneously 0 or 1. We therefore include the linear equality constraint \( x_{g,i} + z_{g,i} = 1 \) for each gate and for each time–entry.

### 5.3.3 Objective function

For each gate and time–entry we substitute \( (f_g(i) \oplus f_g(i + 1)) \) from equation 5.1 by the nonlinear expression \( (x_{g,i} - x_{g,i+1})^2 \) and include it in the objective function. This quadratic formulation for the eXclusive–OR function performs better for our purposes compared to the formulation \( x_{g,i} + x_{g,i+1} - 2x_{g,i}x_{g,i+1} \) which is used in [Wan97]. This is because the first order derivatives of \( x_{g,i}^2 \) and \( x_{g,i+1}^2 \) out of \( (x_{g,i} - x_{g,i+1})^2 \) are not constants, and the second order derivatives are non–zero. The solution technique we use performs better if meaningful first and second order derivatives are available. For \( x_{g,i} + x_{g,i+1} - 2x_{g,i}x_{g,i+1} \) the first order derivatives of \( x_{g,i} \) and \( x_{g,i+1} \) would be constants and the second order derivatives would be zero. In the Boolean domain both expressions \( f_g(i) + f_g(i + 1) - 2f_g(i)f_g(i + 1) \) and \( (f_g(i) - f_g(i + 1))^2 \) perform identically, since \( f_g(i) \cdot f_g(i) \) equals \( f_g(i) \).
5.3.4 Inverters

Since we have already introduced a variable \( x_{g,i} \) for the Boolean value at the output of the gate \( g \) at a certain time–entry \( i \) and a variable \( z_{g,i} \) for its complement, we do not need to add additional variables and constraints for inverters and buffers. We have also already added the constraint to relate these two variables. We account for propagation delays caused by buffers and inverters. We also take into account the capacitance of buffers and inverters and contribute that capacitance to their predecessor in the circuit.

5.3.5 Logic gates

For an AND–gate \( g \) at time–entry \( i \) the following Boolean relation holds:

\[
\left\{ \prod_{p \in \text{pred}(g)} f_p(y_p) \right\} = f_g(i) \quad (5.3)
\]

In this equation \( \text{pred}(g) \) denotes the set of inputs to gate \( g \). The index \( y_p \) is given by the index of the time–entry for gate \( p \) at time \( T_g(i) - t_g \). Here \( t_g \) is the propagation delay of gate \( g \) and \( T_g(i) \) is the function giving the time corresponding to time–entry \( i \) of gate \( x \). The inverse function \( T_g^{-1}(t) \) gives the time–entry for gate \( g \) corresponding to time \( t \). So the following equation holds:

\[
y_p = T_p^{-1}(T_g(i) - t_g) \quad (5.4)
\]

The constraint for the relation for an AND–gate given in equation 5.3 then is:

\[
\left\{ \prod_{p \in \text{pred}(g)} x_{p,y_p} \right\} = x_{g,i} \quad (5.5)
\]

To describe an OR–operation we make use of DeMorgan’s rule. This gives the following Boolean relation:

\[
\left\{ \prod_{p \in \text{pred}(g)} f_p(y_p) \right\} = f_g(i) \quad (5.6)
\]

The corresponding constraint for this relation then is:

\[
\left\{ \prod_{p \in \text{pred}(g)} z_{p,y_p} \right\} = z_{g,i} \quad (5.7)
\]

Complex gates can be decomposed into several AND–, OR– and INVERT–operations and can therefore be modelled as several constraints. Of course, the
internal variables for a complex gate resulting from this decomposition are not represented in the objective function, because transitions internal to a gate do not constitute a switching of the gate. In case of a complex gate we only add the propagation delay of the gate to each entry in the time–entry list for the output of the gate. For the internal relations the following relation between time–entries holds:

\[ y_p = T_p^{-1}(T_g(i)) \]  

The constraints for the internal AND–, OR– and INVERT–operations now simply follow equations 5.5 and 5.7. Of course, for each of the internal Boolean relations of a gate we also have to add the constraint \( x + z = 1 \). We also define a set \( \text{internal}(g) \) as the set of all internal variables as a result of the decomposition of the complex gate \( g \) including the output of the gate.

### 5.4 Example

![Diagram](image)

**Figure 5.2.** A small example for the generation of a NLP to the maximum power estimation problem

We now look at an example. Figure 5.2 gives a small network. The gates in this network are denoted by the capital letters A to E. The primary inputs are denoted by the lowercase letters a to d. The values between braces are time–entries of the time–entry list assuming a unit delay.

We now introduce a real variable \( x_{g,i} \) bounded to \([0,1]\), as discussed in previous sections. This variable \( x_{g,i} \) represents \( f_g(i) \). Similarly we introduce a real variable \( z_{g,i} \) bounded to \([0,1]\) representing \( \overline{f_g(i)} \). In equation 5.9 the
objective function estimating the maximum power for the example of figure 5.2 is given.

\[
\text{Maximize } \frac{1}{2} V_{\text{dd}}^2 C_{\text{load}}(A) \left( (x_{a,-\infty} - x_{a,0})^2 \right) + \\
\frac{1}{2} V_{\text{dd}}^2 C_{\text{load}}(B) \left( (x_{B,-\infty} - x_{B,1})^2 + (x_{B,1} - x_{B,2})^2 \right) + \\
\frac{1}{2} V_{\text{dd}}^2 C_{\text{load}}(C) \left( (x_{C,-\infty} - x_{C,1})^2 \right) + \\
\frac{1}{2} V_{\text{dd}}^2 C_{\text{load}}(E) \left( (x_{E,-\infty} - x_{E,2})^2 + (x_{E,2} - x_{E,3})^2 + (x_{E,3} - x_{E,4})^2 \right)
\]  

Equation 5.10 gives the linear equality constraints binding the variables \( x_{g,i} \) and the complementing variables \( z_{g,i} \), for all gates and for all time-entries in the circuit. We note that no additional variables are used for inverters A and D.

\[
x_{a,-\infty} + z_{a,-\infty} = 1 \\
x_{a,0} + z_{a,0} = 1 \\
x_{b,-\infty} + z_{b,-\infty} = 1 \\
x_{c,0} + z_{c,0} = 1 \\
x_{c,-\infty} + z_{c,-\infty} = 1 \\
x_{c,0} + z_{c,0} = 1 \\
x_{d,-\infty} + z_{d,-\infty} = 1 \\
x_{d,0} + z_{d,0} = 1 \\
x_{B,-\infty} + z_{B,-\infty} = 1 \\
x_{B,0} + z_{B,0} = 1 \\
x_{B,1} + z_{B,1} = 1 \\
x_{B,2} + z_{B,2} = 1 \\
x_{C,-\infty} + z_{C,-\infty} = 1 \\
x_{C,0} + z_{C,0} = 1 \\
x_{C,1} + z_{C,1} = 1 \\
x_{E,-\infty} + z_{E,-\infty} = 1 \\
x_{E,0} + z_{E,0} = 1 \\
x_{E,1} + z_{E,1} = 1 \\
x_{E,2} + z_{E,2} = 1 \\
x_{E,3} + z_{E,3} = 1 \\
x_{E,4} + z_{E,4} = 1
\]
Equation 5.11 gives the nonlinear constraints representing the logic gates (except the inverters) of the circuit of figure 5.2.

\[
\begin{align*}
z_{a,- \infty} x_{b,- \infty} &= z_{B,- \infty} \\
z_{a,- \infty} x_{b,0} &= z_{B,1} \\
z_{a,0} x_{b,0} &= z_{B,2} \\
x_{c,- \infty} x_{d,- \infty} &= z_{C,- \infty} \\
x_{c,0} x_{d,0} &= z_{C,1} \\
x_{B,- \infty} z_{C,- \infty} &= x_{E,- \infty} \\
x_{B,- \infty} z_{C,1} &= x_{E,2} \\
x_{B,1} z_{C,1} &= x_{E,3} \\
x_{B,2} z_{C,1} &= x_{E,4}
\end{align*}
\] (5.11)

5.5 Nonlinear constrained optimization

5.5.1 Maximum power estimation problem formulation

The lower bound maximum power estimation problem as introduced in section 5.3 can be summarized as:

\[
\text{Maximize } \sum_{g \in \text{gates}} \frac{1}{2} V_{dd}^2 C_{\text{load}}(g) \cdot \sum_{i=1}^{n(g)-1} \left(x_{g,i} - x_{g,i+1}\right)^2
\] (5.12)

subject to \( \forall g \in \text{gates} \ \forall h \in \text{internal}(g) \ \forall 1 \leq i \leq n(h) \ \forall x_{h,i} + z_{h,i} = 1 \)

\[
\forall g \in \text{gates} \ \forall h \in \text{internal}(g) \ \forall 1 \leq i \leq n(h) \ \left( \prod_{p \in \text{pred}(h)} \{x, z\}_{p,y_p}\right) = x_{h,i}
\]

In equation 5.12 \{x, z\} is either x or z depending on the Boolean function to be realized, and \(y_p\) corresponds to equation 5.4. Equation 5.12 is a nonlinear constrained maximization problem on continuous variables in the range \([0, 1]\). We first discuss a way to solve this continuous optimization problem and then how to make the conversion back from the continuous domain to the Boolean domain.

5.5.2 Solving the maximum power estimation problem

We use LANCELOT[Con92] to solve the problem described in equation 5.12. LANCELOT makes use of an Augmented Lagrange method to deal with the (nonlinear) constraints. We provide not only the objective function and the
constraints but also the first and second order derivatives of the objective function and the constraints in this approach. We do this by generating a SIF-file [Con92] which specifies the objective function, the constraints and their first and second order derivatives. For a more detailed description of the nonlinear optimization package LANCELOT and of the methods used by it we refer to [Con92] and section 2.7. Because we are maximizing an objective function which is a square of the difference of two variables both in the range [0,1], nearly all variables are either 0 or 1, once the optimization has finished. As a starting point for all continuous variables in the optimization problem we choose 0.0 and we have defined no additional bounds on those variables.

5.5.3 Conversion from the continuous to the Boolean domain

When the maximization process is completed we have an optimal solution with real variables to the maximum power estimation problem. We still want to convert the solution back to the Boolean domain. We perform this conversion by rounding the real variables in the range [0,1] to their nearest integer value, being either 0 or 1. Most variables are already at or near 0.0 or 1.0 respectively. This means that one of the variables is either 1.0 or 0.0 and the complementary variable is smaller than 1E–3 or larger than 9.999E–1 respectively. It is clear that rounding in this case does not change the solution.

For large examples it occurs that some variables are not near 0.0 or 1.0. The complementary variables \(x_{g,i}\) and \(z_{g,i}\) then still add up to 1.0, but both of them stabilize at 0.5. Usually there are less than 10 of such variables (5 pairs of complementary variables). So a tractable approach may be to fix such variables to either 0 or 1 in all possible combinations and solve the remaining nonlinear problems. A drawback of this approach is that LANCELOT takes a long time to terminate on such inferior solutions containing variables stabilizing on 0.5. This appears to render this approach impractical.

5.6 Experiments

For the experiments, we map several circuits from the MCNC benchmark set on a library containing complex AOI–gates. We have limited the library to 3,3–AOI–gates, in order to limit the maximum number of inputs for a cell. In table 5.1 we give the number of gates (#gate) in each circuit, the maximum number of levels of gates (#level) in each circuit and the number of variables (#var) used by our approach to model the maximum power estimation problem. We also give the number of inputs (#in) to each circuit and the number of linear (#L) and nonlinear (#NL) constraints used to model the maximum power estimation problem. Table 5.1 shows us that the number of
variables required to model the problem depends not only on the number of gates in the circuit, but also heavily on the number of logic levels in the circuit. The number of variables is a good measure for the size of the optimization problem, but does not say everything about the time it takes to solve problem. We can see this if we compare the number of variables with the CPU–times required for solving the problem. So the structure of the circuit has also an influence on the CPU–time required to solve the problem.

**Table 5.1** Number of inputs, gates, levels, variables and constraints for the benchmark circuits

<table>
<thead>
<tr>
<th></th>
<th>#in</th>
<th>#gate</th>
<th>#level</th>
<th>UNIT</th>
<th>FLOAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#var</td>
<td>#L</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#var</td>
<td>#L</td>
</tr>
<tr>
<td>con1</td>
<td>7</td>
<td>23</td>
<td>7</td>
<td>156</td>
<td>78</td>
</tr>
<tr>
<td>dc1</td>
<td>4</td>
<td>54</td>
<td>9</td>
<td>296</td>
<td>148</td>
</tr>
<tr>
<td>9sym</td>
<td>9</td>
<td>84</td>
<td>17</td>
<td>428</td>
<td>214</td>
</tr>
<tr>
<td>5xp1</td>
<td>7</td>
<td>113</td>
<td>11</td>
<td>852</td>
<td>426</td>
</tr>
<tr>
<td>alu2</td>
<td>10</td>
<td>132</td>
<td>14</td>
<td>770</td>
<td>385</td>
</tr>
<tr>
<td>dc2</td>
<td>8</td>
<td>140</td>
<td>13</td>
<td>918</td>
<td>459</td>
</tr>
<tr>
<td>duke2</td>
<td>22</td>
<td>686</td>
<td>20</td>
<td>4768</td>
<td>2384</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8578</td>
<td>4289</td>
</tr>
</tbody>
</table>

For the circuits we use both a unit delay model (table 5.2) and a real delay model with floating point values for the delays (table 5.3). The floating point delay model clearly results in a much larger number of time–entries. This can be seen in table 5.1 as each time–entry results in two variables. We also use capacitance estimates represented by floating point values. We then create the input specification file for LANCELOT and start the optimization for each of the circuits. We also estimate the upper bound power estimate by the method of the previous chapter. The result of the upper bound power estimation is in the column marked upper bound in both tables. The results of our lower bound power estimation approach are in the column marked lower bound. We compare the lower bound estimates and the upper bound estimates in the column marked with %. This percentage is calculated according to equation 5.13.

\[
\text{percentage} = \frac{\text{lower bound estimated power}}{\text{upper bound estimated power}} \times 100\%
\]  

(5.13)

From the percentage we can see how accurate our estimations of the maximum power are. The closer the percentage is to 100%, the smaller is the gap between upper and lower bound estimate. As we know that the actual maximum power should be between the upper and lower bound estimates the higher the percentage the more accurate our estimation methods are.
TABLE 5.2 Maximum power estimate, iterations and CPU–time for MCNC–circuits of function of the penalty parameter using a UNIT delay model

<table>
<thead>
<tr>
<th>Circuit</th>
<th>upper bound</th>
<th>lower bound</th>
<th>%</th>
<th>iterations</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>con1</td>
<td>7.65</td>
<td>5.63</td>
<td>74</td>
<td>31</td>
<td>0.4 s</td>
</tr>
<tr>
<td>dc1</td>
<td>19.58</td>
<td>13.61</td>
<td>70</td>
<td>23</td>
<td>0.9 s</td>
</tr>
<tr>
<td>9sym</td>
<td>33.56</td>
<td>17.96</td>
<td>54</td>
<td>65</td>
<td>6.3 s</td>
</tr>
<tr>
<td>5xp1</td>
<td>58.26</td>
<td>36.67</td>
<td>63</td>
<td>115</td>
<td>33.5 s</td>
</tr>
<tr>
<td>alu2</td>
<td>65.40</td>
<td>27.44</td>
<td>42</td>
<td>72</td>
<td>16.4 s</td>
</tr>
<tr>
<td>dc2</td>
<td>67.36</td>
<td>28.67</td>
<td>43</td>
<td>36</td>
<td>7.1 s</td>
</tr>
<tr>
<td>duke2</td>
<td>714.98</td>
<td>315.72</td>
<td>44</td>
<td>92</td>
<td>6 m 4.1 s</td>
</tr>
</tbody>
</table>

TABLE 5.3 Maximum power estimate, iterations and CPU–time for MCNC circuits as a functions of the penalty parameter using a FLOAT delay model

<table>
<thead>
<tr>
<th>Circuit</th>
<th>upper bound</th>
<th>lower bound</th>
<th>%</th>
<th>iterations</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>con1</td>
<td>9.48</td>
<td>4.67</td>
<td>49</td>
<td>50</td>
<td>1.1 s</td>
</tr>
<tr>
<td>dc1</td>
<td>24.58</td>
<td>11.93</td>
<td>49</td>
<td>45</td>
<td>4.0 s</td>
</tr>
<tr>
<td>9sym</td>
<td>81.71</td>
<td>40.87</td>
<td>50</td>
<td>255</td>
<td>1 m 35.0 s</td>
</tr>
<tr>
<td>5xp1</td>
<td>91.19</td>
<td>37.28</td>
<td>41</td>
<td>148</td>
<td>1 m 27.0 s</td>
</tr>
<tr>
<td>alu2</td>
<td>106.57</td>
<td>47.19</td>
<td>44</td>
<td>101</td>
<td>40.7 s</td>
</tr>
<tr>
<td>dc2</td>
<td>97.05</td>
<td>42.42</td>
<td>44</td>
<td>143</td>
<td>1 m 25.9 s</td>
</tr>
<tr>
<td>duke2</td>
<td>1209.5</td>
<td>345.32</td>
<td>29</td>
<td>510</td>
<td>283 m 56.7 s</td>
</tr>
</tbody>
</table>

The column marked % in tables 5.2 and 5.3 shows the difference between upper and lower bound power estimates. We observe the general trend that as the circuit becomes larger the upper and lower bound estimate are, even relatively, further apart. This is both due to a less tight upper bound estimate as well as due to a less tight lower bound estimate.

As tables 5.2 and 5.3 show we are able to calculate a lower bound maximum power estimate with corresponding input vectors for the circuit duke2. The circuit duke2 has 22 primary inputs as can see in table 5.1. That means that there are $2^{22}$ input vector combinations. So calculating the maximum power through simulation is out of the question.

5.7 Discussion

In this chapter we have presented a method to estimate a lower bound on the maximum power of CMOS circuits. Our method establishes this problem as
a nonlinear optimization problem. Unlike some existing methods for computing the lower bound power estimate our nonlinear optimization problem also has linear and nonlinear constraints. The optimization problem is solved by LANCELOT. The resulting estimate is accompanied by the input vector pair realizing this power estimate in the circuit. Though the method is successful, it is not yet able to supply power estimates for circuits comparable in size to the ones used for the upper bound power estimates in the previous chapter. This is however not surprising as nonlinear optimization problems are much more difficult to solve than linear programs. However, the assumption of a limited logic depth, as in the previous chapter, may provide a way to handle larger circuits.

The main weakness of the presented method appears to be lack of convergence of LANCELOT for large examples. This problem manifests itself in that after many iterations most variables are either 0.0 or 1.0, but some stabilize on 0.5 (together with their complements). In our examples usually less than 10 variables (5 pairs) exhibit this behavior.
Chapter

6 Gate Sizing under a Statistical Delay Model

6.1 Introduction

This chapter is about gate sizing under a statistical delay model. Gate sizing refers to the process of assigning drive strengths (speed/sizing factors) optimally to the individual gates for a given cost function and constraints. For a discussion on gate sizing under a traditional delay model see chapter 3.

We will first discuss static timing analysis using a statistical delay model. Then we will discuss how to use this statistical delay model in the gate sizing step. In static delay analysis, the circuit is modeled as a network of delay–inducing elements. The most simple form is the Boolean network [Bra87] of a circuit, with delays assigned to the gates. If this network is acyclic, we can perform a longest path analysis on the network in linear time, and derive the latest possible arrival time of signals at the primary outputs of the circuit. Mostly a “typical” delay value is quoted for each gate.

In reality, the delays of network elements such as gates and wires are not constants. There are basically two different types of deviation from the typical delay value of a gate or wire. The first type are those deviations, that apply to all gates in the circuit in a similar way. To this category belong effects like:

- changes in the global power supply voltage
- global temperature changes on the chip
- variations during the production process which result in performance changes

The other type of deviation is local in nature. In this category we can think of effects like changes in delay due to:

- the state of the gate (the Boolean values on the inputs of the gate)
- different input signal rise (or fall) times
- local supply noise
- local temperature changes
- cross talk on wires
• delay change due to local manufacturing imperfections

Traditionally, the problem of these variations is handled by introducing three different delay values for each delay inducing element: a minimum, or best–case delay, a typical delay, and a maximum, or worst–case delay. We can now perform our delay calculations 3 times, and obtain best–case, typical and worst–case values for the delay at the outputs of the circuit. For the global deviations described above, this seems a perfectly reasonable way to handle the delay variations, as all delay inducing elements are affected in a similar way. For the local deviations, however, this does not seem reasonable at all. While one gate may become slower for a certain variation, another may become faster. Modeling these local variations by calculation with worst–case values for all gates, is very likely to result in an overly pessimistic value for the delay. In order to handle these local delay variations more realistically, in this chapter we propose to use stochastic variables for the delays, and to use statistical calculations to derive probability distributions for the output delays.

All above mentioned effects are physical in nature. One might also consider to model uncertainties in delay statistically reflecting the inaccurate knowledge of the physical features at the current state of the synthesis process. For example as the delay may be unknown because actual routing of the wires has not yet been performed.

We will also discuss the use of the new statistical delay model in the gate sizing step. As we will see, it is of paramount importance to have analytical expressions for the mean and standard deviation of the maximum–operation available. These expressions are analytical functions of the mean and standard deviation of the operandi. The availability of these functions is the key to successful operation of gate sizing under the statistical delay model. First of all the analytical functions deliver a speed–up over Monte–Carlo simulations. Secondly the analytical results open up the possibility of having analytical first and second order derivatives of the objective function and constraints to the variables (among which are the drive strengths) of the gate sizing problem. This again makes it possible to solve the gate sizing problem under a statistical delay model for large instances. We will solve the gate sizing problem using LANCELOT [Con92].

In this chapter we will first discuss related work in section 6.2. Then we will look into the theory of statistical delay calculation in section 6.3. Section 6.4 will deal with validating the choice of a normal distribution for the propagation delay distribution of a gate and with the repeated approximations of the resulting delay distributions with a normal distribution. In section 6.6 we will formulate the gate sizing problem assuming statistical delays. Section 6.7 illustrates the gate sizing formulation with an example.
In section 6.8 we present results on the performance of the gate sizing method using a statistical delay for circuits up to a few thousand gates. Finally in section 6.9 we discuss the merits of statistical delay calculation and gate sizing using the statistical delay model.

6.2 Related work

In the recent past, a few attempts have been made to perform static timing analysis in a statistical way. The first attempt known to the author is described in [Hit82a] and [Hit82b]. Unfortunately, the details of how the statistical calculations were performed are not revealed in the papers. Comparisons between normal and statistical results are also not given. A more recent attempt is described in [Jyu94], where the statistical properties were obtained with Monte–Carlo simulations. Monte–Carlo simulations take a long time to compute, because for accurate results a large number of simulations is required. Monte–Carlo simulations are therefore not practical in a synthesis process.

An even more recent approach is described in [Ber97a] and [Ber97b]. These papers approximate the mean and standard deviation of the maximum of two stochastic variables both normally distributed. The papers describe how this maximum of two normal distributions resembles a normal distribution. The idea of using normal distributions for all distribution functions stems from these papers. However, in these papers the mean and standard deviation of the distribution function resulting from applying the maximum operator on two normal distribution is obtained by sampling. This does not allow for a formal model of gate sizing. We have therefore derived analytical expressions in [Jac99a] and [Jac2000], in order to enable gate sizing using the statistical delay model. The work presented in this chapter is based on [Ber97b], [Jac99a] and [Jac2000].

6.3 Theory of statistical calculations

We model the schedule time of a signal $T$ as a stochastic variable, which we assume normally distributed with a mean $\mu_T$ and a standard deviation $\sigma_T$. We will also model the delay of a gate $t_p$ as a stochastic variable, with mean $\mu_{t_p}$ and standard deviation $\sigma_{t_p}$. In [Berk97b] it has already been shown that the shape of the distribution for the delay elements is almost irrelevant if we are only interested in the delay distribution of the entire circuit (see section 6.4).
Traditional delay calculation calculates the delay at the output of a gate with two inputs “1” and “2” by:

\[
T_m = \max(T_1, T_2)
\]

\[
T_{\text{out}} = T_m + t_p
\]

\(T_{\text{out}}\) refers to the delay at the output of the gate, while \(T_1\) and \(T_2\) refer to the delay at the two inputs. This calculation involves two operations: A maximum—operation and an addition.

We will now investigate performing the addition and maximum—operation for statistical variables. For two statistically independent normally distributed stochastic variables \(A\) and \(B\) we can calculate the mean \(\mu_C\) and standard deviation \(\sigma_C\) of stochastic variable \(C\), defined as \(A + B\), by:

\[
\mu_C = \mu_A + \mu_B
\]

\[
\sigma_C^2 = \sigma_A^2 + \sigma_B^2
\]

We also have to perform the maximum—operation with stochastic values. To see what happens in this case, we will concentrate on calculating \(C = \max(A, B)\), with \(A\), \(B\) and \(C\) stochastic variables. \(A\) and \(B\) are normally distributed with means \(\mu_A\) and \(\mu_B\) and standard deviations \(\sigma_A\) and \(\sigma_B\). For any value \(x\) we can write:

\[
P(C \leq x) = P((A \leq x) \cap (B \leq x))
\]

(6.3)

Assuming statistical independence of \(A\) and \(B\), we can write equation 6.3 as:

\[
P(C \leq x) = P(A \leq x) \cdot P(B \leq x)
\]

(6.4)

Assuming statistical independence implies an approximation in the case of reconverging paths in the circuit. In section 6.5 we will discuss the error of making this approximation. We will use the notation \(F_A\) for the distribution function of the stochastic variable \(A\). We will also use the notation:

\[
P(A \leq x) = F_A(x) = \int_{-\infty}^{x} f_A(u)du
\]

(6.5)

in which \(f_A\) for a normal distribution is given by:

\[
f_A(x) = \frac{1}{\sigma_A \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{x - \mu_A}{\sigma_A} \right)^2}
\]

(6.6)

Using this notation, and taking the derivative on both sides of equation 6.4, we obtain:

\[
f_C(x) = F_A(x)f_B(x) + f_A(x)F_B(x)
\]

(6.7)
We refer the reader to [Ber97a] and [Ber97b] where it is shown that the resulting probability density function \( f_C \) for stochastic variable \( C \) is very similar to a normal distribution. Given any probability density function, we can always calculate the mean and standard deviation [Pap91]. In case of the normal distribution the probability density function is completely characterized by the mean and standard deviation. We now compute \( \mu_C \) which is a function of just \( \mu_A, \mu_B, \sigma_A \) and \( \sigma_B \):

\[
\mu_C = E[x_C] = \frac{\sqrt{\sigma_A^2 + \sigma_B^2}}{\sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{x_A - x_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)} + \mu_A \Phi \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right) + \mu_B \Phi \left( \frac{\mu_B - \mu_A}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)
\]

for which \( \phi(x) \) is defined by:

\[
\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}u^2} du
\]

We also give \( E[x_C^2] \):

\[
E[x_C^2] = (\mu_A + \mu_B) \frac{\sqrt{\sigma_A^2 + \sigma_B^2}}{\sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{x_A - x_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)} + \left( \sigma_A^2 + \mu_A^2 \right) \Phi \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right) + \left( \sigma_B^2 + \mu_B^2 \right) \Phi \left( \frac{\mu_B - \mu_A}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)
\]

from which we can calculate \( \sigma_C \) by:

\[
\sigma_C^2 = E[x_C^2] - \mu_C^2
\]

Thus \( \sigma_C \) is a function of \( \mu_A, \mu_B, \sigma_A \) and \( \sigma_B \). Appendix A provides the derivation of equations 6.8 and 6.10.
We now return to the intended application of delay calculation. Similar to equation 6.1 we can calculate the delay at the output of a gate using normally distributed stochastic variables:

$$\mu_{T_m} = \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{1}{2} \left( \frac{\mu_{T_1} - \mu_{T_2}}{\sqrt{\alpha^2_{T_1} + \alpha^2_{T_2}}} \right)^2 \right)$$

$$\mu_{T_1} \Phi \left( \frac{\mu_{T_1} - \mu_{T_2}}{\sqrt{\alpha^2_{T_1} + \alpha^2_{T_2}}} \right) + \mu_{T_2} \Phi \left( \frac{\mu_{T_2} - \mu_{T_1}}{\sqrt{\alpha^2_{T_1} + \alpha^2_{T_2}}} \right)$$

$$\sigma^2_{T_m} = \left( \mu_{T_1} + \mu_{T_2} \right) \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{1}{2} \left( \frac{\mu_{T_1} - \mu_{T_2}}{\sqrt{\alpha^2_{T_1} + \alpha^2_{T_2}}} \right)^2 \right) +$$

$$\left( \alpha^2_{T_1} + \mu^2_{T_1} \right) \Phi \left( \frac{\mu_{T_1} - \mu_{T_2}}{\sqrt{\alpha^2_{T_1} + \alpha^2_{T_2}}} \right) + \left( \alpha^2_{T_2} + \mu^2_{T_2} \right) \Phi \left( \frac{\mu_{T_2} - \mu_{T_1}}{\sqrt{\alpha^2_{T_1} + \alpha^2_{T_2}}} \right) - \mu_{T_m}^2$$

$$\mu_{T_{out}} = \mu_{T_m} + \mu_t$$

$$\sigma^2_{T_{out}} = \sigma^2_{T_m} + \sigma^2_t$$

### 6.4 Validation of approximating the gate delay by a normal distribution

In order to validate the approximation of the delay distribution of a single gate by a normal distribution we show that the larger the circuit is, the less relevant the shape of the distribution function is. In order to show this we calculate the distribution function of several circuits using both a uniform distribution for the gate delays as well as normal distributed gate delay with Monte–Carlo simulation. Figure 6.1 shows the resulting delay distribution for an inverter chain under both a uniform and a normal distribution for several stages of the inverter chain. The experimental results in figure 6.1 are computed with $10^6$ samples.
Figure 6.1. Shape of delay distribution function for an inverter chain after (1, 2, 3, and 5 inverters) under a uniform (dots) and normal (line) delay distribution for the individual gates.

We observe in figure 6.1 that already after a few stages the shape of the delay distribution function resembles a normal distribution for both the case with a uniform distribution, and the case with a normal distribution for the gate delays. This is in line with the “Central Limit Theorem” [Pap91], which states that in the limit adding a large number of independent statistical variables with continuous distribution functions, will yield a normal distribution regardless of the distribution functions of the individual variables.

Now we perform Monte–Carlo simulations using both a uniform delay as well as a normal distributions to model the gate delays in the circuit C7552. Figure 6.2 shows the distribution of circuit propagation delays for the Monte–Carlo simulations for this circuit based on $10^5$ samples.
We observe in figure 6.2 that the resulting distribution functions for both unit and normal distribution are very much alike. Both have approximately the shape of a normal distribution. We now calculate the mean and standard deviation of a number of benchmark circuits under both a unit and a normal distribution as well as the difference between them in percentages. Table 6.1 show the results. We also use $10^5$ samples for these experiments.
TABLE 6.1 Monte–Carlo simulation of circuits under uniform and normal gate delay distributions (percentages are calculated as the difference between results for uniform and normal distributions divided by the result for the normal distributions)

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>normal mean</th>
<th>normal sigma</th>
<th>uniform mean</th>
<th>uniform sigma</th>
<th>u–n/n %</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>100.50</td>
<td>4.483</td>
<td>100.42</td>
<td>4.418</td>
<td>-0.08 % -1.45 %</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>92.45</td>
<td>2.653</td>
<td>92.38</td>
<td>2.506</td>
<td>-0.08 % -5.58 %</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>101.28</td>
<td>3.968</td>
<td>101.33</td>
<td>3.968</td>
<td>0.07 % 0 %</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>135.21</td>
<td>3.254</td>
<td>135.12</td>
<td>3.083</td>
<td>-0.07 % -5.26 %</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>227.37</td>
<td>6.302</td>
<td>227.33</td>
<td>6.244</td>
<td>-0.03 % -0.92 %</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>317.63</td>
<td>9.094</td>
<td>317.70</td>
<td>8.940</td>
<td>0.02 % -1.69 %</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>507.38</td>
<td>12.214</td>
<td>507.27</td>
<td>12.160</td>
<td>-0.02 % -0.52 %</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>581.88</td>
<td>15.645</td>
<td>581.96</td>
<td>15.366</td>
<td>0.01 % -1.78 %</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>1252.67</td>
<td>20.819</td>
<td>1253.11</td>
<td>20.758</td>
<td>0.04 % -0.29 %</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>634.57</td>
<td>16.812</td>
<td>634.57</td>
<td>15.923</td>
<td>0 % -5.42 %</td>
</tr>
</tbody>
</table>

We can see that for all benchmarks circuits results are off only by less than 0.08 % for the mean and by less than 5.58 % percent for the standard deviation. The standard deviation in the uniformly distributed case tends to be less than the normally distributed case. We therefore conclude that for circuits with a non–trivial size the shape of the distribution of the delays of the individual gates has only a marginal influence on the mean and standard deviation of the delay distribution of the entire circuit.

6.5 Validation of approximation of maximum of two normal distributed variables

Consider equation 6.7. We take $F_A$ to be a probability density function for one normal distribution and $F_B$ for a second but different normal distribution. Then $f_A$ and $f_B$ are the corresponding normal distributions. We can easily see from equation 6.7 that $f_C$ is not exactly a normal distribution. However, $f_C$ resembles a normal distribution very closely, as we will show. Our approach calls for a normal distribution to be used in subsequent calculations. So we approximate $f_C$ with a normal distribution.

We will now show that the result of a maximum–operation of two normal distributed stochastic variables, is very much like a normal distribution using the figures 6.3 to 6.7. In each of these figures we have chosen the standard deviation of the delay distribution of inputs A and B to both be 0.25.
In figure 6.3 we start with a mean delay for both inputs of 1.0. This results in a curve for the resulting distribution function, which has a smaller standard deviation and a higher mean than the individual input distributions. We can see that the curve computed by sampling the inputs looks very similar to the normal distributed curve, of which mean and standard deviation were calculated by means of equations 6.8 through 6.11.

![Graph showing distribution functions](image)

**Figure 6.3.** Sampled and calculated distribution functions for maximum-operation with input distributions with mean 1.0 and standard deviation 0.25

Next we move the mean of input B to the slightly higher value of 1.1, while keeping the mean of input A on 1.0. We also keep the standard deviations to their value of 0.25. We can see that the resulting distribution for the MAX-operation is slightly moved to the right (higher mean) and that the distribution curve is slightly wider. The approximated normal distributed curve looks very similar to the calculated curve.
Figure 6.4. Sampled and calculated distribution functions for maximum-operation with input distributions with mean 1.0 and 1.1 and standard deviation 0.25

We again increase the mean delay of the distribution function of input B for figure 6.5. The mean delay of input B is now 1.3. The standard deviation of the resulting output curves is still smaller than the 0.25 of the input distributions. We can however see that the probability distribution curve for the maximum is not narrow as in figure 6.4. The mean of the resulting distribution function of figure 6.5 is also still larger than the means of the input distributions, but the difference from the mean of input B is smaller here than in figure 6.4. We again see that a normal distribution is a good approximation for the resulting curve.
We now increase the mean delay of input B to 1.5, while keeping the mean delay of input A and both standard deviations the same. We can see in figure 6.6 that the resulting curve looks more and more like the distribution function of the largest delay distribution function, that of input B. The resulting curve has again become wider and lower than in previous figures. An approximation by a normal distribution still looks very reasonable.
In our final figure 6.7 we increase the mean delay of input B to 2.0. The resulting curve for the MAX–operation is indistinguishable from the normal
distributed curve for input B, which has the largest mean delay of the two inputs.

Figures 6.3 through 6.7 show us that the approximation of the distribution function of the MAX-operation of two normally distributed independent inputs by a new normal distribution is quite acceptable. Furthermore, the figures show us that the calculation of the mean and standard deviation of the maximum according to equations 6.8 through 6.11 is accurate.

The question still open, is whether the assumption of statistical independence used in the calculation of the maximum will introduce a large error. We will therefore perform two sets of experiments. We will first present results for circuits which do not contain any reconverging paths. We will analytically calculate mean and standard deviation and also simulate these using Monte–Carlo simulations. The results are given in table 6.2. The circuits are tree–circuits. The first number in the name is the maximal logic depth and the second number the maximum number of inputs allowed. We used $10^5$ Monte–Carlo simulations for each of the experiments.

**Table 6.2** Results of calculating mean and standard deviation analytically and using Monte–Carlo simulations for tree–circuits (percentages are calculated as the difference between analytical and Monte–Carlo divided by Monte–Carlo)

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>analytical</th>
<th>Monte–Carlo</th>
<th>a–m/m %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mean</td>
<td>sigma</td>
<td>mean</td>
</tr>
<tr>
<td>T20_2a</td>
<td>6382</td>
<td>228.77</td>
<td>5.326</td>
<td>229.89</td>
</tr>
<tr>
<td>T20_2b</td>
<td>3270</td>
<td>138.63</td>
<td>3.671</td>
<td>139.14</td>
</tr>
<tr>
<td>T20_2c</td>
<td>2098</td>
<td>103.13</td>
<td>2.939</td>
<td>103.45</td>
</tr>
<tr>
<td>T13_3a</td>
<td>1951</td>
<td>79.96</td>
<td>2.630</td>
<td>80.34</td>
</tr>
<tr>
<td>T13_3b</td>
<td>1369</td>
<td>64.89</td>
<td>2.431</td>
<td>65.17</td>
</tr>
<tr>
<td>T13_3c</td>
<td>4512</td>
<td>145.04</td>
<td>4.263</td>
<td>145.91</td>
</tr>
<tr>
<td>T11_4a</td>
<td>2645</td>
<td>99.46</td>
<td>3.550</td>
<td>100.05</td>
</tr>
<tr>
<td>T11_4b</td>
<td>7711</td>
<td>237.73</td>
<td>7.514</td>
<td>239.55</td>
</tr>
<tr>
<td>T11_4c</td>
<td>7400</td>
<td>228.47</td>
<td>7.387</td>
<td>230.28</td>
</tr>
</tbody>
</table>

Next we perform the same experiment on circuits which have reconverging paths. The results are presented in table 6.3.
Table 6.3 Results of calculating mean and standard deviation analytically and using Monte–Carlo simulations for circuits with reconverging paths (percentages are calculated as the difference between analytical and Monte–Carlo divided by Monte–Carlo)

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>analytical mean</th>
<th>analytical sigma</th>
<th>Monte–Carlo mean</th>
<th>Monte–Carlo sigma</th>
<th>a–m/m %</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>240</td>
<td>108.15</td>
<td>1.631</td>
<td>100.50</td>
<td>4.483</td>
<td>7.51 %</td>
</tr>
<tr>
<td>C499</td>
<td>436</td>
<td>94.27</td>
<td>1.172</td>
<td>92.45</td>
<td>2.653</td>
<td>1.97 %</td>
</tr>
<tr>
<td>C880</td>
<td>513</td>
<td>104.48</td>
<td>2.526</td>
<td>101.28</td>
<td>3.968</td>
<td>3.16 %</td>
</tr>
<tr>
<td>C1355</td>
<td>644</td>
<td>139.63</td>
<td>1.234</td>
<td>135.21</td>
<td>3.254</td>
<td>3.27 %</td>
</tr>
<tr>
<td>C1908</td>
<td>1100</td>
<td>234.39</td>
<td>3.598</td>
<td>227.37</td>
<td>6.302</td>
<td>3.09 %</td>
</tr>
<tr>
<td>C2670</td>
<td>1987</td>
<td>329.17</td>
<td>4.458</td>
<td>317.63</td>
<td>9.094</td>
<td>3.63 %</td>
</tr>
<tr>
<td>C3540</td>
<td>2641</td>
<td>525.87</td>
<td>4.211</td>
<td>507.38</td>
<td>12.214</td>
<td>3.64 %</td>
</tr>
<tr>
<td>C5315</td>
<td>3624</td>
<td>606.96</td>
<td>7.142</td>
<td>581.88</td>
<td>15.645</td>
<td>4.31 %</td>
</tr>
<tr>
<td>C6288</td>
<td>2438</td>
<td>1325.10</td>
<td>10.374</td>
<td>1252.67</td>
<td>20.819</td>
<td>5.78 %</td>
</tr>
<tr>
<td>C7552</td>
<td>4925</td>
<td>658.56</td>
<td>5.968</td>
<td>634.57</td>
<td>16.812</td>
<td>3.78 %</td>
</tr>
</tbody>
</table>

We compare the results presented in tables 6.2 and 6.3. We observe that the analytically determined means for the tree–circuits in table 6.2 are slightly smaller (less than 1%) than the results determined by Monte–Carlo simulation. In table 6.3 the results for the analytically determined means are somewhat larger (2 to 8%) than the results determined by Monte–Carlo simulation. We also observe that the analytically determined standard deviations are always smaller than those determined by Monte–Carlo simulation. The standard deviations are off by less than 10 percent for the tree circuits, but by 35 to 66 % for the circuits with reconverging paths. Especially the difference for the standard deviations for the circuits with reconverging paths is considerable. We therefore conclude that reconverging paths, jeopardizing the assumption of statistical independence between inputs, have considerable influence on the quality of the results. It might be possible to take statistical dependence because of reconverging paths into account in future. This is however not straightforward and therefore a topic for future research. We will continue to use the assumption of statistical independence in the remainder of this chapter demonstrating gate sizing under the statistical delay model.
6.6 Gate sizing using the statistical delay model

We will now incorporate the statistical delay model in the gate sizing formulation. As was shown in chapter 3, the speed factor \( S_{gate} \) and the delay of the gate \( t_{p,gate} \) are related as follows:

\[
t_{p,gate}S_{gate} = t_{int,gate}S_{gate} + c \cdot \left( C_{wire,gate} + \sum_{j \in \text{suox(gate)}} C_{in,j}S_j \right)
\]  
(6.13)

We have written equation 6.13 such that there are as few nonlinear terms as possible, just as in equation 3.7. In equation 6.13 \( t_{int,gate} \) is a constant denoting the delay due to capacitances internal to the gate, \( c \) is a constant relating propagation delay to capacitance, \( C_{wire,gate} \) is a constant denoting the capacitance loading the gate (mainly in wiring), \( C_{in,j} \) is a constant denoting the gate oxide capacitances of the transistors of gate \( j \) driven by the current gate and \( S_j \) is the speed (sizing) factor of that gate. The speed factor can range from 1, meaning no speed–up, to “limit” times speed–up. In the gate sizing approach using the statistical delay model we take the mean of the gate delay \( \mu_{t_{p,gate}} \) equal to the delay \( t_{p,gate} \) in equation 6.13. The standard deviation \( \sigma_{t_{p,gate}} \) we define as a function of \( \mu_{t_{p,gate}} \):

\[
\sigma_{t_{p,gate}} = f(\mu_{t_{p,gate}})
\]  
(6.14)

In the following sections we choose function \( f \) such that the standard deviation \( \sigma_{t_{p,gate}} \) is 25% of \( \mu_{t_{p,gate}} \). We make these assumptions regarding the propagation delay distribution of the gate, because actual distributions in for example transistor and wire widths, doping levels, wiring loads and transistor lengths are not know at this time. So the propagation delay distribution of the gate can not be calculated based on distributions of \( S_{gate}, t_{int,gate}, c, C_{wire,gate}, C_{in,j}, \) and \( S_j \). We think that a standard deviation of 25% on the propagation delay of a gate is of the right order of magnitude as will be observed on an IC manufacturing line.

Now that we have discussed the equations relating delay to sizing, we need to discuss calculating the circuit delay. In our statistical approach we need to calculate the maximum arrival time at the inputs of a gate (see also section 2.2 equation 2.1). For each gate we calculate the mean and standard deviation of the maximum of the circuit delays at the inputs of the gate using equations 6.8, 6.10 and 6.11. Now we still have to take into account the gate delay. We add this gate delay (see also equation 2.1) to the calculated maximum circuit delay at the inputs using equation 6.2. We also calculate the delay distribution of the total circuit by taking the stochastic maximum over all the
primary outputs of the circuit. The mean and standard deviation of the total delay will be denoted by $\mu_{T_{\text{max}}}$ and $\sigma_{T_{\text{max}}}$ respectively.

The total gate sizing formulation for minimal delay is given in equation 6.15. Note that also different objective functions are possible. In equation 6.15 $\max_{\mu}$ and $\max_{\sigma}$ denote functions calculating the mean and standard deviation of the maximum of a number of normal distributed stochastic variables. Note that the constraints in equation 6.15 are either all equality constraints or simple constraints on the range of individual variables, and that, while some are linear, others are highly nonlinear. We will solve the constrained nonlinear programming formulation of the form described in equation 6.15 using LANCELOT [Con92].

$$\begin{align*}
\text{minimize} & \quad \mu_{T_{\text{max}}} \\
\text{subject to} & \quad \mu_{T_{\text{max}}} = \max_{\mu}(\mu_{T_{\text{out}1}}, \ldots, \mu_{T_{\text{out}n}}, \sigma_{T_{\text{out}1}}, \ldots, \sigma_{T_{\text{out}n}}) \quad \text{n number of primary outputs} \\
& \quad \sigma_{T_{\text{max}}} = \max_{\sigma}(\mu_{T_{\text{out}1}}, \ldots, \mu_{T_{\text{out}n}}, \sigma_{T_{\text{out}1}}, \ldots, \sigma_{T_{\text{out}n}}) \quad \text{n number of primary outputs} \\
& \quad \text{and for each gate } i: \\
& \quad \mu_{t_{pi}S_i} = t_{\text{int},i}S_i + c \cdot \left[ C_{\text{load},i} + \sum_{j \in \text{suces}(i)} C_{\text{in},j}S_j \right] \\
& \quad \mu_{T_{\text{out}i}} = \mu_{U_{\text{max},i}} + \mu_{t_{pi}} \\
& \quad \sigma_{T_{\text{out}i}}^2 = \sigma_{U_{\text{max},i}}^2 + \sigma_{t_{pi}}^2 \\
& \quad \mu_{U_{\text{max},i}} = \max_{\mu}(\mu_{T_{1,i}}, \sigma_{T_{1,i}}, \ldots, \mu_{T_{m,i}}, \sigma_{T_{m,i}}) \quad \text{m number of inputs of gate } i \\
& \quad \sigma_{U_{\text{max},i}} = \max_{\sigma}(\mu_{T_{1,i}}, \sigma_{T_{1,i}}, \ldots, \mu_{T_{m,i}}, \sigma_{T_{m,i}}) \quad \text{m number of inputs of gate } i \\
& \quad \sigma_{t_{pi}} = f(\mu_{t_{pi}}) \\
& \quad 1 \leq S_i \leq \text{limit}
\end{align*}$$

In order for LANCELOT to solve equation 6.15 we have to calculate the first and second order derivatives of all terms in the problem to every problem variable, because otherwise LANCELOT does not work efficiently. We find it advantageous to have as many linear terms, as opposed to nonlinear terms, as possible in each constraint, because this also increases the efficiency of LANCELOT. This is the reason behind the formulation of equation 6.13 in that form, which contains as few nonlinear terms as possible. We also use only the square of the standard deviations in the model. Therefore we substitute a new variable replacing the square of the standard deviation. All the effort put into expressing the mean and the standard deviation of the maximum–
operator as a function of the means and standard deviations of the operands have a purpose: it enables the calculation of analytical partial derivative expressions. These analytical partial derivative expressions enable us to evaluate the value of a partial derivative quickly and accurately.

We can also choose other objective functions and add additional constraints to the gate sizing formulations. For instance we can choose a weighed sum of sizing factors in the objective function. This can model area, or even power, if we take into account capacitances and switching activity under zero delay model in the weights.

We can also add delay constraints: on the mean circuit propagation delay or on the sum of the mean and multiples of the standard deviation. Adding the standard deviation in the circuit propagation delay constraint ensures that a larger percentage of the circuits will conform to the delay constraint. Assume the delay constraint is defined as \( \mu_{T_{\text{max}}} + k \cdot \sigma_{T_{\text{max}}} \). Then the case of \( k = 0 \) implies 50% of the circuits will conform to the delay constraint. The values for \( k = 1 \) and \( k = 3 \) imply conformance of 84.1% and 99.9% respectively. What such a conformance to a delay constraint actually means depends on the variability modeled.

### 6.7 Example

We will now look at the example of figure 6.8. The gates in this network are denoted by the capital letters A to F. The primary inputs are given by the lowercase letters a to e. We give the corresponding gate sizing formulation for minimal \( \mu_{T_{\text{max}}} + 3\sigma_{T_{\text{max}}} \) using a statistical delay method in equation 6.16.

This means that we calculate the sizing factors for each gate in the circuit such that 99.9% of the circuits have a minimum propagation delay. The maximum over all outputs is taken in 6.16a. Note that we can only calculate the statistical maximum for two operands directly at the same time. For the multiple inputs of gate D, we use the two--operand maximum operation repeatedly (6.16b). This is an approximation. The gate delay is added to the maximum over the inputs in 6.16c. The equations relating gate size and delay are in 6.16d. For the function relating mean and standard deviation of a gate we assume the standard deviation to be a quarter of the mean in this example (6.16e). We assume a maximum speed–up (“limit”) of 3 for each gate (6.16f).
Figure 6.8. A small example for the generation of a gate sizing formulation for minimal delay using a statistical delay model

\[
\text{minimize } \mu_{T_{\text{max}}} + 3\sigma_{T_{\text{max}}} 
\]
\[
\mu_{T_{\text{max}}} = \max_{\mu}(\mu_{T_A}, \sigma_{T_A}, \mu_{T_F}, \sigma_{T_F}) 
\]
\[
\sigma_{T_{\text{max}}} = \max_{\sigma}(\mu_{T_C}, \sigma_{T_C}, \mu_{T_F}, \sigma_{T_F}) 
\]
\[
\mu_{U_C} = \max_{\mu}(\mu_{T_A}, \sigma_{T_A}, \max_{\mu}(\mu_{T_B}, \sigma_{T_B}, \mu_{T_D}, \sigma_{T_D}), \max_{\sigma}(\mu_{T_B}, \sigma_{T_B}, \mu_{T_D}, \sigma_{T_D})) 
\]
\[
\sigma_{U_C} = \max_{\sigma}(\mu_{T_A}, \sigma_{T_A}, \max_{\mu}(\mu_{T_B}, \sigma_{T_B}, \mu_{T_D}, \sigma_{T_D}), \max_{\sigma}(\mu_{T_B}, \sigma_{T_B}, \mu_{T_D}, \sigma_{T_D})) 
\]
\[
\mu_{T_F} = \max_{\mu}(\mu_{T_D}, \sigma_{T_D}, \mu_{T_E}, \sigma_{T_E}) 
\]
\[
\sigma_{T_F} = \max_{\sigma}(\mu_{T_D}, \sigma_{T_D}, \mu_{T_E}, \sigma_{T_E}) 
\]
\[
\mu_{T_C} = \mu_{U_C} + \mu_{t_c} 
\]
\[
\sigma_{T_C}^2 = \sigma_{U_C}^2 + \sigma_{t_c}^2 
\]
\[
\mu_{T_F} = \mu_{U_F} + \mu_{t_f} 
\]
\[
\sigma_{T_F}^2 = \sigma_{U_F}^2 + \sigma_{t_f}^2 
\]
\[
\mu_{t_A} S_A = S_A t_{\text{int}_A} + c \cdot (C_{\text{load}_A} + C_{\text{in}S_C}) 
\]
\[
\mu_{t_B} S_B = S_B t_{\text{int}_B} + c \cdot (C_{\text{load}_B} + C_{\text{in}S_C}) 
\]
\[ \mu_{t_c} S_C = S_C t_{\text{int}_c} + c \cdot \text{C}_{\text{load}_c} \]
\[ \mu_{t_d} S_D = S_D t_{\text{int}_d} + c \cdot \left( \text{C}_{\text{load}_d} + \text{C}_{\text{in}S_C} + \text{C}_{\text{in}S_F} \right) \]
\[ \mu_{t_x} S_E = S_E t_{\text{int}_x} + c \cdot \left( \text{C}_{\text{load}_x} + \text{C}_{\text{in}S_F} \right) \]
\[ \mu_{t_y} S_F = S_F t_{\text{int}_y} + c \cdot \text{C}_{\text{load}_y} \]
\[ \sigma_{t_A} = 0.25 \mu_{t_A} \]
\[ \sigma_{t_B} = 0.25 \mu_{t_B} \]
\[ \sigma_{t_C} = 0.25 \mu_{t_C} \]
\[ \sigma_{t_D} = 0.25 \mu_{t_D} \]
\[ \sigma_{t_E} = 0.25 \mu_{t_E} \]
\[ \sigma_{t_F} = 0.25 \mu_{t_F} \]
\[ 1 \leq S_A \leq 3 \]
\[ 1 \leq S_B \leq 3 \]
\[ 1 \leq S_C \leq 3 \]
\[ 1 \leq S_D \leq 3 \]
\[ 1 \leq S_E \leq 3 \]
\[ 1 \leq S_F \leq 3 \]

### 6.8 Experimental results

We have performed two sets of experiments. The first is intended to show both the applicability of the statistical delay gate sizing method to circuits of up to a few thousand gates, as well as to demonstrate the additional objectives and constraints we can formulate using our statistical gate sizing method. As can be seen in table 6.4 the method is able to handle circuits of up to a few thousand gates. We have performed several experiments with the three circuits. The first two entries (rows) for each circuit give the range in which the mean propagation delay and the area (we use the sum of speed factors as a measure of area in our experiments) can vary. The next two entries (rows) for each circuit show the results of minimizing the mean propagation delay plus one time and plus three times the standard deviation. The last three entries in the table for each of the three circuits minimize the area subject to constraints on the mean propagation delay, the mean propagation delay plus one time and plus three times the standard deviation.
Table 6.4 Results of statistical sizing for some large benchmark circuits

<table>
<thead>
<tr>
<th>name</th>
<th>#gates</th>
<th>minimize constraint</th>
<th>μTmax</th>
<th>σTmax</th>
<th>ΣS_i</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>apex1</td>
<td>982</td>
<td>ΣS_i</td>
<td>173.72</td>
<td>5.867</td>
<td>982</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax</td>
<td>73.21</td>
<td>2.099</td>
<td>1989</td>
<td>41 m 13.5 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + σTmax</td>
<td>73.26</td>
<td>1.972</td>
<td>1949</td>
<td>41 m 10.8 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + 3σTmax</td>
<td>73.57</td>
<td>1.701</td>
<td>1843</td>
<td>67 m 54.8 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax ≤ 120</td>
<td>120.00</td>
<td>2.950</td>
<td>998</td>
<td>67 m 49.9 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + σTmax ≤ 120</td>
<td>117.16</td>
<td>2.842</td>
<td>1001</td>
<td>103 m 19.4 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + 3σTmax ≤ 120</td>
<td>112.07</td>
<td>2.645</td>
<td>1007</td>
<td>85 m 43.1 s</td>
</tr>
<tr>
<td>apex2</td>
<td>117</td>
<td>ΣS_i</td>
<td>31.50</td>
<td>1.784</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax</td>
<td>23.45</td>
<td>1.419</td>
<td>304</td>
<td>18.5 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + σTmax</td>
<td>23.48</td>
<td>1.373</td>
<td>294</td>
<td>10 m 16.5 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + 3σTmax</td>
<td>23.79</td>
<td>1.202</td>
<td>279</td>
<td>52.2 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax ≤ 29</td>
<td>29.00</td>
<td>1.488</td>
<td>123</td>
<td>42.1 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + σTmax ≤ 29</td>
<td>27.64</td>
<td>1.365</td>
<td>131</td>
<td>7.0 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + 3σTmax ≤ 29</td>
<td>25.47</td>
<td>1.176</td>
<td>154</td>
<td>38.3 s</td>
</tr>
<tr>
<td>k2</td>
<td>1692</td>
<td>ΣS_i</td>
<td>183.98</td>
<td>3.281</td>
<td>1692</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax</td>
<td>75.00</td>
<td>1.239</td>
<td>3750</td>
<td>54 m 26.1 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + σTmax</td>
<td>75.02</td>
<td>1.228</td>
<td>3690</td>
<td>50 m 45.7 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + 3σTmax</td>
<td>75.23</td>
<td>1.120</td>
<td>3596</td>
<td>83 m 20.7 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax ≤ 120</td>
<td>120.00</td>
<td>1.829</td>
<td>1794</td>
<td>221 m 32.0 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + σTmax ≤ 120</td>
<td>118.27</td>
<td>1.744</td>
<td>1801</td>
<td>214 m 38.4 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μTmax + 3σTmax ≤ 120</td>
<td>115.10</td>
<td>1.637</td>
<td>1814</td>
<td>157 m 50.6 s</td>
</tr>
</tbody>
</table>

Gate sizing combined with the statistical delay calculation method thus enables us to size circuits, to obtain a certain confidence in the circuit realizing its timing constraints, or to optimize the number of circuits that will operate at the required clock frequency given the uncertainties in the propagation delay. All experiments are performed on a Hewlett Packard K260. The CPU–times reported refer to the performance LANCELOT (see section 2.7.1) on this processor. The considerable differences in CPU–times show that solving these nonlinear optimization problems is a challenge. This challenge lies both in numerical issues as well as in the highly nonlinear problem descriptions.
The second set of experiments is on the tree–circuit of figure 6.9, which contains seven NAND–gates. These experiments are to show how different constraints and objective functions affect the speed factors for this simple circuit. The first two entries (rows) of table 6.5 denote the range in which the area and mean propagation delay of the circuit can vary. We have selected three values of the mean propagation delay in this range. One is chosen in the middle and the other two nearer the extremes of the range. Table 6.5 shows that there is a margin to change the standard deviation given a fixed mean propagation delay, and that the interval is largest for the middle choice in the range of mean propagation delays. It is also clear from the table 6.5 that minimal standard deviation given a fixed mean propagation delay leads to a higher area usage than just minimizing area given that mean propagation delay.
**Table 6.5** Results for the tree–circuit

<table>
<thead>
<tr>
<th>objective</th>
<th>constraint</th>
<th>( \mu_{T_{\text{max}}} )</th>
<th>( \sigma_{T_{\text{max}}} )</th>
<th>( \Sigma S_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \min \Sigma S_i )</td>
<td></td>
<td>7.4</td>
<td>0.811</td>
<td>7.00</td>
</tr>
<tr>
<td>( \min \mu_{T_{\text{max}}} )</td>
<td></td>
<td>5.4</td>
<td>0.592</td>
<td>21.00</td>
</tr>
<tr>
<td>( \min \Sigma S_i )</td>
<td>( \mu_{T_{\text{max}}}=5.8 )</td>
<td>5.8</td>
<td>0.631</td>
<td>14.73</td>
</tr>
<tr>
<td>( \min \sigma_{T_{\text{max}}} )</td>
<td>( \mu_{T_{\text{max}}}=5.8 )</td>
<td>5.8</td>
<td>0.622</td>
<td>15.66</td>
</tr>
<tr>
<td>( \max \sigma_{T_{\text{max}}} )</td>
<td>( \mu_{T_{\text{max}}}=5.8 )</td>
<td>5.8</td>
<td>0.667</td>
<td>19.22</td>
</tr>
<tr>
<td>( \min \Sigma S_i )</td>
<td>( \mu_{T_{\text{max}}}=6.5 )</td>
<td>6.5</td>
<td>0.704</td>
<td>9.54</td>
</tr>
<tr>
<td>( \min \sigma_{T_{\text{max}}} )</td>
<td>( \mu_{T_{\text{max}}}=6.5 )</td>
<td>6.5</td>
<td>0.689</td>
<td>10.20</td>
</tr>
<tr>
<td>( \max \sigma_{T_{\text{max}}} )</td>
<td>( \mu_{T_{\text{max}}}=6.5 )</td>
<td>6.5</td>
<td>0.831</td>
<td>15.51</td>
</tr>
<tr>
<td>( \min \Sigma S_i )</td>
<td>( \mu_{T_{\text{max}}}=7.2 )</td>
<td>7.2</td>
<td>0.786</td>
<td>7.21</td>
</tr>
<tr>
<td>( \min \sigma_{T_{\text{max}}} )</td>
<td>( \mu_{T_{\text{max}}}=7.2 )</td>
<td>7.2</td>
<td>0.689</td>
<td>7.25</td>
</tr>
<tr>
<td>( \max \sigma_{T_{\text{max}}} )</td>
<td>( \mu_{T_{\text{max}}}=7.2 )</td>
<td>7.2</td>
<td>0.817</td>
<td>9.08</td>
</tr>
</tbody>
</table>

We now look at the speed factors in table 6.6 corresponding to the sizing experiments for the tree–circuit for minimal area, and for minimal and maximal standard deviation. Table 6.6 shows that both sizing for minimal area (sum of speed factors) and for minimal standard deviation treat similar gates (first group: \( S_A, S_B, S_D \) and \( S_E \); second group: \( S_C \) and \( S_F \)) similarly, and gates towards the output of the circuit get larger speed factors. This behavior is more extreme in case of sizing for minimal standard deviation. The standard deviation for gates nearer the input does not need to be as small as for gates nearer the output, because for a balanced mean delay and similar gates the maximum operator results in a slightly higher mean but considerably smaller standard deviation. Sizing for maximal standard deviation clearly differentiates delays on different paths to maximize the standard deviation, as is to be expected. The last gate \( S_G \) is then appropriately sized to achieve the required mean propagation delay.

**Table 6.6** Speed factors for the tree–circuit for \( \mu_{T_{\text{max}}}=6.5 \)

<table>
<thead>
<tr>
<th>objective</th>
<th>( S_A )</th>
<th>( S_B )</th>
<th>( S_C )</th>
<th>( S_D )</th>
<th>( S_E )</th>
<th>( S_F )</th>
<th>( S_G )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \min \Sigma S_i )</td>
<td>1.22</td>
<td>1.22</td>
<td>1.45</td>
<td>1.22</td>
<td>1.22</td>
<td>1.45</td>
<td>1.74</td>
</tr>
<tr>
<td>( \min \sigma_{T_{\text{max}}} )</td>
<td>1.00</td>
<td>1.00</td>
<td>2.01</td>
<td>1.00</td>
<td>1.00</td>
<td>2.01</td>
<td>3.00</td>
</tr>
<tr>
<td>( \max \sigma_{T_{\text{max}}} )</td>
<td>3.00</td>
<td>1.00</td>
<td>1.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>1.51</td>
</tr>
</tbody>
</table>

**6.9 Discussion**

In this chapter we have shown a static timing analysis approach using a statistical delay model based upon a normal distribution. We have also
presented an analytical formulation for the mean and standard deviation of the result of a maximum-operation on two normally distributed stochastic variables as a function of their mean and standard deviation. We have shown that the assumption of a normal distribution for the gate propagation delays and the approximation to a normal distributed function of the result of a maximum-operation is reasonable.

We have presented a gate sizing method under a statistical delay model, which we expressed as a nonlinear programming problem. As an essential step in the modeling of the statistical gate sizing method, we have expressed the mean and standard deviation of the result of the maximum-operator as a function of the means and standard deviations of the operandi. We have solved the gate sizing formulation under a statistical delay problem exactly, for problems up to a few thousand gates using the large scale nonlinear programming package LANCELOT. We have presented several improvements to the gate sizing formulation and discussed implementation details crucial to solving the gate sizing problem efficiently. We also presented experiments demonstrating the effect of different objective functions and the result of those objective functions on the speed factors.

Future work will look into dealing with correlations between stochastic variables in the circuit, as a result of reconverging paths, which is currently not included in our delay model and is a cause of error. It might also be possible to eliminate some intermediate approximations of distribution function to normal distributions, by looking into the direct calculation of the mean and standard deviation of the maximum operation of multiple (more than two) inputs.
Chapter

7 Concluding Remarks

7.1 Research goals and results

In this thesis we have directed our attention to the problems regarding power dissipation and timing in the field of logic synthesis for CMOS circuits. We motivated this choice for the logic synthesis domain, by the fact that though larger savings in power dissipation can be achieved at higher abstraction levels than the logic domain, many problems regarding power dissipation in logic synthesis still exist. The main idea behind all this is the still existing lack of predictability of power dissipation at this abstraction level. Lack of predictability means that synthesis at higher abstraction levels is hampered, because the result of design decisions cannot be evaluated in a sufficiently fast and accurate way. The necessity of fast and accurate evaluation is the main concern in all gate sizing synthesis approaches presented. This objective is combined with increasing the predictability of the results of synthesis. The necessity of accuracy of the model, while still being able to evaluate it for practically sized circuits, is the main concern for the upper and lower bound power and current estimation methods we have presented.

Chapter 2 has dealt with the modeling of logic gates and their timing. We have introduced a timing model that incorporates the internal resistances and capacitances, as well as external capacitances in the model for the computation of the gate delay. We have also investigated the propagation of glitches through logic gates. We have both looked at inverters and buffer gates as well as more complex gates, with multiple inputs. A method has been derived that calculates the output pulse width as a function of the input pulse width, both for an input pulse on one input, and as a result of transitions on multiple inputs.

We have also given a brief discussion and description of mathematical programming solvers. We have looked at linear programming solvers, those using the simplex method and those using interior point methods. We have also seen nonlinear programming solvers. These encompass general nonlinear solvers like LANCELOT, which are based on an augmented Lagrangian method, as well quadratic and geometric solvers based on interior point methods. These solvers are the means by which we have
attacked both the estimation and synthesis problems described in the other chapters of this thesis.

In chapter 3 we have taken a thorough look into gate sizing problems. We have restricted ourselves to gate sizing based on mathematically described delay models. We have modelled both dynamic power dissipation and short-circuit power dissipation in the context of gate sizing problems. We have seen and solved gate sizing problems for minimal power, minimal power under a propagation delay constraint, minimal circuit propagation delay or a weighted combination of those. We have solved the gate sizing problem using a linear program, after PWL-approximation of the SGPDs. Geometric programming and general nonlinear programming using LANCELOT have also been applied directly to the nonlinear optimization problem. Linear programming seems most suitable, in case interior point methods are used, as it is by far the fastest. Using linear programming however introduces an error because of the PWL-approximation of essentially nonlinear constraints. The interior point method is better than the simplex method as it is able to deal with larger problems and does not run into numerical difficulties. In case the problem has to be solved without PWL-approximation LANCELOT works best. LANCELOT is able to solve larger problems faster than geometric programming, even though a geometric program is a special case of nonlinear programming, with a specialized solver.

We have enhanced the gate sizing method to be able to deal with glitches. One of the problems of gate sizing for power is that gate sizing not only changes the capacitances and resistances of the gate which directly influences power dissipation, it also changes the propagation delay. A change in propagation delay influences the transition density of, in theory, possibly all signals in the circuit. The transition density, a measure of the number of transitions, is also an important aspect of power dissipation. We have dealt with this problem by assuming a zero delay model, without glitches. This is a valid assumption, because we balance the propagation delays of the circuit during the gate sizing process. We also need to introduce buffers to even out excessive differences in propagation delays. The resulting gate sizing problem contains additional constraints, which forbid PWL-approximation of SGPDs. We have therefore solved the problem using LANCELOT.

We have presented an upper bound maximum power estimation method in chapter 4, which models the estimation problem as a linear optimization problem. The approach is able to deal with relatively large circuits of several thousand gates, provided the logic depth of the circuit is limited to 10 per pipeline stage. We have also discussed the extension of this method to the problem of estimating an upper bound on the maximum current. This is possible for a unit delay model, for which the number of windows is limited.
to the logic depth of the circuit (after limiting the logic depth to 10 per stage). The number of windows is so small that the repeated calculation of the upper bound estimation approach for each window can be performed in a reasonable amount of time. For a general delay model there can exist several hundred windows. The repeated calculation of the linear program for each window can in this case take considerable time.

In chapter 5 we have presented a method for the estimation of a lower bound on the maximum power of CMOS circuits. We have presented this method in order to be able to obtain an idea of the range of the actual maximum power, together with the previously mentioned upper bound method. The lower bound method also enables the calculation of the corresponding input vector pair, which realizes the lower bound estimate. Unfortunately the lower bound estimation method can only handle circuits that are much smaller compared to those handled by the upper bound method. This is not surprising as the lower bound method relies on general nonlinear optimization, which is known to be more difficult and less robust than the linear programming used for the upper bound maximum power estimation method.

Chapter 6 has introduced a statistical static delay model. We have modeled the statistical delay model such that all operations that have to be performed while calculating the static propagation delay of a circuit are modeled as analytical functions. One of the main contributions of this chapter is the modeling of the maximum of two statistically normally distributed delays on the input of a gate as an analytical function. We have shown that an approximation of this maximum is possible by again a normal distribution with limited error. The mean and standard deviation of the normal distribution of this maximum can be given as an analytical function of the means and standard deviations of the inputs. These analytical functions are essential to the evaluation speed of the model and enable the calculation of analytical derivatives. This again makes the model applicable in a synthesis process. We have incorporated this statistical delay model in a gate sizing problem. We have shown the applicability of the statistical delay model for the gate sizing process on several benchmark circuits for different objective functions. We can now not only minimize for circuit delay, but also give an indication of the reliability of the circuit actually conforming to the calculated delay.

7.2 Suggestions for future work

We have discussed the timing modeling of logic gates in chapter 2. We have taken into account propagation delays as a result of internal capacitances and resistances as well as external capacitances. We have also discussed the
influence of glitches on the propagation of signals and their influence on the timing of the gate. However, there is one aspect of the timing modeling of logic gates that is not yet taken into account: the current model does not include the influence of the slopes of the input signals. This could deliver an even more accurate delay model at the logic level. We should however guard ourselves against a too complex model, since this will take too much time to evaluate. An excessive evaluation time would disqualify the model for use in the synthesis process.

With regard to gate sizing for power minimization there are still possibilities for improvements. Though a fairly exhaustive look has been taken into the gate sizing process itself, there are still improvements to the preprocessing step. For instance it may be possible to perform delay equalization with less buffers. This can save both area and power as buffers also consume power. It may also be possible to perform other logic synthesis steps in such a way that they reduce switching activity and simultaneously equalize the propagation delays along different paths. While the minimizing of switching activity has been the subject of several papers before [Lin93], the problem of delay equalization, and more demandingly the combination of these objectives has hardly been discussed. One option of an approach may use a wavefront approach [Stok99] for the technology mapping step to bound the choices.

The nonlinear formulation for gate sizing may also be used in combination with clock skew optimization. It is possible to formulate the problem described in [Sath95], in one nonlinear formulation. This can possibly create optimal, instead of suboptimal results for this problem. It may also realize these solutions in less time.

The last suggestions regarding gate sizing concern the discrete nature of most libraries. On one hand given a desired drive strength it is possible to generate cells automatically. There have already been some papers solving subproblems related with such an approach. One of these is [Eijn94]. Much work however still can be done in for example characterizing these generated gates during the automatic generation, calculating both propagation delays as well as power dissipation. On the other hand it is also possible to cast the discrete gate sizing problem into a nonlinear problem. The nonlinear nature enables the use of nonlinear equality constraints. One can model a variable $x$ to be either 0 or 1 using the nonlinear equality constraint $x^2 - x = 0$. It may be possible to solve a gate sizing formulation also containing these kind of constraints using LANCELOT.

As has already been noted in section 4.7, the approach for estimating an upper bound on the maximum power can also be extended to upper bound maximum current estimation. A more careful look, however, still has to be
taken into the way maximum current estimation fits into the total design flow in order to be able to improve the performance of the algorithm. Maximum current estimation may be useful for a limited part of the circuit at a time, for example in a row of gates which share the same power and ground lines. Such an approach will limit the number of windows that have to be considered, which is the main bottleneck of the approach. Also a lower bound on the maximum current can be interesting as a lower bound comes with a set of input vectors which realize that current estimate unlike with the upper bound estimation. An input vector pair is useful, because this input vector pair can be used to accurately simulate power and timing and to obtain other useful information applying circuit simulation for this input vector pair.

The last suggestions of course concern the statistical delay model. As has already been noted the remaining inaccuracies of the method are due to the repeated approximation of arbitrary distribution functions by normal distributions. Calculating the maximum of multiple stochastic variables directly instead of repeatedly taking the maximum of two may already eliminate inaccuracies. Another source of inaccuracies is the assumption of statistical independence between the two operand variables when taking their maximum. This assumption is not valid for reconverging paths in the circuit. This problem unfortunately still remains completely open.

To use the statistical delay model we have to obtain more accurate statistical models of gates. Also more insight is required about the statistical correlations of gate model parameters. For this the correspondence between fabrication process parameters and gate model parameters has to be investigated. This requires significantly more process information than is currently available. A method which uses a statistical approach based on fabrication process information may however improve yield considerably.
Appendix

A Stochastic maximum

Here, we will derive the mean and standard deviation of a stochastic variable C which is the maximum of two normal distributed statistically independent stochastic variables A and B. In order to derive this mean and standard deviation we will change the bases \((x, y)\) of the double integration:

\[
\int_{-\infty}^{\infty} x f_A(x) \int_{-\infty}^{\infty} f_B(y) dy dx
\]

(A.1)

which is taken from the calculation of \(\mu_C\) in equation 6.8 into \((u, v)\). We will change the bases in order to formulate a double integral in which the bounds of the second integral do not contain the integration variable of the first integral. Therefore we need a rotation. We will also choose the variables for our new bases such that the expression is as simple to integrate as possible. We therefore need a translation to account for \(\mu_A\) and \(\mu_B\) and a scaling to account for \(\sigma_A\) and \(\sigma_B\). We choose our new bases as follows:

\[
\frac{x - \mu_A}{\sigma_A} = \frac{u \sigma_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} - \frac{v \sigma_A}{\sqrt{\sigma_A^2 + \sigma_B^2}}
\]

(A.2)

which gives:

\[
x = \frac{u \sigma_A \sigma_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} - \frac{v \sigma_A^2}{\sqrt{\sigma_A^2 + \sigma_B^2}} + \mu_A
\]

(A.3)

and:

\[
\frac{y - \mu_B}{\sigma_B} = \frac{u \sigma_A}{\sqrt{\sigma_A^2 + \sigma_B^2}} + \frac{v \sigma_B}{\sqrt{\sigma_A^2 + \sigma_B^2}}
\]

(A.4)

which gives:

\[
y = \frac{u \sigma_A \sigma_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} + \frac{v \sigma_B^2}{\sqrt{\sigma_A^2 + \sigma_B^2}} + \mu_B
\]

(A.5)

For this change of bases we calculate:
\[
\frac{\delta(x,y)}{\delta(v,u)} = \begin{vmatrix}
\frac{\sigma_A \sigma_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} - \frac{\sigma_A^2}{\sqrt{\sigma_A^2 + \sigma_B^2}} & \frac{\sigma_A^3}{\sqrt{\sigma_A^2 + \sigma_B^2}} \\
\frac{\sigma_A \sigma_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} & \frac{\sigma_B^2}{\sqrt{\sigma_A^2 + \sigma_B^2}} 
\end{vmatrix} = \frac{\sigma_A \sigma_B^3 + \sigma_A^3 \sigma_B}{\sqrt{\sigma_A^2 + \sigma_B^2}^2} = \sigma_A \sigma_B
\]  
(A.6)

---

**Figure A.1.** From old bases (x,y) to new bases (u,v) (scaling not incorporated for simplicity).

To integrate over the same area as before we also have to incorporate the area between the u-axis and the line y=x. This is accomplished by integrating \(v\) until \(\frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}}\) instead of until 0. This is the distance between the u-axis and the line y=x. The mean of the stochastic variable \(C\) then becomes:

\[
\mu_C = \int_{-\infty}^{\infty} x f_C(x) \, dx = \int_{-\infty}^{\infty} x f_A(x) F_B(x) \, dx + \int_{-\infty}^{\infty} x F_A(x) f_B(x) \, dx
\]

\[
= \frac{1}{\sigma_A \sigma_B \sqrt{2\pi} \sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \left\{ \frac{u \sigma_A \sigma_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} - \frac{v \sigma_A^3}{\sqrt{\sigma_A^2 + \sigma_B^2}} + \mu_A \right\} e^{-\frac{1}{2}(u^2 + v^2) \sigma_A \sigma_B} \, dv \, du + ...
\]  
(A.7)
\[
\begin{align*}
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \left( \frac{\nu A \nu B}{\sqrt{\sigma_A^2 + \sigma_B^2}} - \frac{\nu^2 A}{\sqrt{\sigma_A^2 + \sigma_B^2}} + \mu_A \right) e^{-\frac{1}{2}(u^2 + v^2)} dv du + \ldots \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left( -\frac{\nu A^2}{\sqrt{\sigma_A^2 + \sigma_B^2}} + \mu_A \right) e^{-\frac{1}{2}(v^2)} dv + \ldots \\
&= \left[ \frac{1}{2\pi} \frac{\sigma_A^2}{\sqrt{\sigma_A^2 + \sigma_B^2}} e^{-\frac{1}{2}v^2} \right]_{-\infty}^{\infty} + \frac{\mu_A}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(v^2)} dv + \ldots \\
&= \left[ \frac{1}{2\pi} \frac{\sigma_B^2}{\sqrt{\sigma_A^2 + \sigma_B^2}} e^{-\frac{1}{2}v^2} \right]_{-\infty}^{\infty} + \frac{\mu_B}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(v^2)} dv \\
&= \frac{\sqrt{\sigma_A^2 + \sigma_B^2}}{\sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{\nu A^2}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)^2} + \mu_A \Phi \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right) + \mu_B \Phi \left( \frac{\mu_B - \mu_A}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)
\end{align*}
\]

in which \( \phi(x) \) is given by:

\[
\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}u^2} du \quad (A.8)
\]

Note that in some lines of equation A.7 we have only given one half of the equation explicitly. The other half is depicted by triple dots, and is similar to the first half of the equation. We will now calculate the standard deviation of stochastic variable C in two steps. The first step is the calculation of \( E[ x_C^2 ] \):

\[
E[ x_C^2 ] = \int_{-\infty}^{\infty} x^2 f_C(x) dx \quad (A.9)
\]

\[
= \int_{-\infty}^{\infty} x^2 f_A(x) f_B(x) dx + \int_{-\infty}^{\infty} x^2 f_A(x) f_B(x) dx
\]
\[\begin{align*}
    &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \left( \frac{u\sigma^2_A + \sigma^2_B}{\sqrt{\sigma^2_A + \sigma^2_B}} - \frac{v\sigma^4_A + \mu^2_A}{\sqrt{\sigma^2_A + \sigma^2_B}} + \mu_A \right)^2 e^{-\frac{1}{2}(u^2 + v^2)} du dv + \ldots \\
    &\quad - \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \left( \frac{u\sigma^2_A + \sigma^2_B}{\sqrt{\sigma^2_A + \sigma^2_B}} - \frac{v\sigma^4_A + \mu^2_A}{\sqrt{\sigma^2_A + \sigma^2_B}} + \mu_A \right) e^{-\frac{1}{2}(u^2 + v^2)} du dv + \ldots \\
    &\quad + \left[ \frac{2\mu_A \sigma^2_A}{\sqrt{\sigma^2_A + \sigma^2_B}} e^{-\frac{1}{2}\sigma^2_A} \right]_{-\infty}^{\infty} + \left[ \frac{2\mu_A \sigma^2_A}{\sqrt{\sigma^2_A + \sigma^2_B}} e^{-\frac{1}{2}\sigma^2_A} \right]_{-\infty}^{\infty} \\
    &\quad + \left[ \frac{\sigma^4_A + \sigma^2_B}{\sigma^2_A + \sigma^2_B} e^{-\frac{1}{2}\sigma^2_A} \right]_{-\infty}^{\infty} + \ldots
\end{align*}\]
\[\begin{align*}
&= \left( \sigma_A^2 + \mu_A^2 \right) \phi \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right) + \\
&\quad \frac{e^{-\frac{1}{2} \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)^2}}{\sqrt{2\pi} \sqrt{\sigma_A^2 + \sigma_B^2}} \left( 2\mu_A \sigma_A^2 - \frac{\sigma_A^4 (\mu_A - \mu_B)}{\sigma_A^2 + \sigma_B^2} \right) + \ldots \\
&= \left( \sigma_A^2 + \mu_A^2 \right) \phi \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right) + \\
&\quad \frac{e^{-\frac{1}{2} \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)^2}}{\sqrt{2\pi} \sqrt{\sigma_A^2 + \sigma_B^2}} \left( \mu_A \sigma_A^4 + 2\mu_A \sigma_A^2 \sigma_B^2 + \mu_B \sigma_A^4 \right) + \ldots \\
&= \left( \sigma_A^2 + \mu_A^2 \right) \phi \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right) + \left( \sigma_B^2 + \mu_B^2 \right) \phi \left( \frac{\mu_B - \mu_A}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right) + \\
&\quad (\mu_A + \mu_B) \sqrt{\frac{\sigma_A^2 + \sigma_B^2}{\sqrt{2\pi}}} e^{-\frac{1}{2} \left( \frac{\mu_A - \mu_B}{\sqrt{\sigma_A^2 + \sigma_B^2}} \right)^2}
\end{align*}\]

Note that in equation A.9 we also have given only one half of the equation explicitly, with the other half which is similar to the first, depicted by triple dots. We can now calculate the standard deviation of stochastic variable \(C\) with the following equation:

\[\sigma_C^2 = E[x_C^2] - \mu_C^2 \quad \text{(A.10)}\]

We have now expressed \(\mu_C\) and \(\sigma_C\) as functions of \(\mu_A, \mu_B, \sigma_A\) and \(\sigma_B\).
References


<table>
<thead>
<tr>
<th>Ref</th>
<th>Author(s)</th>
<th>Title</th>
<th>Publication Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eijn94</td>
<td>J.T.J. van Eijndhoven</td>
<td>“CMOS cell generation for Logic Synthesis”</td>
<td>In proceedings of the first international conference on ASIC, Beijing, China, October 18–21, 1994, pp. 75–79.</td>
</tr>
</tbody>
</table>


Index

accuracy, 3
addition, 102
AOI–gate, 95
approximation, 102
architectural synthesis, 5
array–based design, 5
ASIC, 5
ATPG. See Automated Test Pattern Generation
augmented Lagrange method, 32
Automated Test Pattern Generation, 88
average power dissipation, 6
barrier function, 27
    control variable, 28
    basic variables, 26
BDD. See Binary Decision Diagram
behavioral description, 5
Binary Decision Diagram, 23
binding, 5
Boolean difference, 23
Boolean domain, 88, 90, 95
Boolean function, 90
Boolean level, 75
Boolean network, 99
Boolean relation, 78, 91
Boolean value, 90
Boolean variable, 23
    complement, 23
branch and bound, 31
buffer, 91
buffer insertion, 37, 65

capacitance, 20, 71
gate oxide, 20, 40
internal, 20, 40
wire, 20, 40

Central Limit Theorem, 105
charge sharing, 11
Cholesky factorization, 29
dense columns, 29
fill–in, 29
circuit reliability, 69
clock skew, 39, 126
    optimization, 65
clock skew optimization, 39
cloning, 37
CMOS–gate, 39
cofactor, 23
complement, 23
complementary variable, 95
concave, 25
continuous domain, 88, 90
continuous variable, 90
control unit, 5
conversion, 95
convex, 25
correlation, 70, 122
cross talk, 99
datapath, 5
delay
    balanced, 63
    best–case, 100
equalization, 126
    inertial, 13
    path propagation, 2
    pin–to–pin, 10
    propagation, 10, 11
    sizable gate propagation, 41
typical, 100
    wire, 10
    worst–case, 100
delay distribution, 101
delay model, 10
general, 24
inertial, 24
statistical, 99, 114
unit, 24
zero, 24
DeMorgan, 78, 91
derivative, 33
  first order, 90, 95
  partial, 33
  second order, 33, 90, 95
derivatives
  first order, 115
  partial, 116
  second order, 115
design abstraction, 3
design flow, 3
design style, 5
  array-based, 5
  full custom, 5
  standard-cell, 5
desirability, 88
deviations, 99
  global, 99
  local, 99
discrete library, 126
distribution function, 102
  shape of, 101, 104
divided differences, 33
domain
  Boolean, 88, 90, 95
  continuous, 88, 90
doping level, 114
drive strength, 37
dynamic power dissipation, 20, 43, 71
electromigration, 2
fabrication process, parameters, 127
fall time, 10
falling slope, 12
feasible area, 28
filtering effect, 13, 21, 63
floorplanning, 5
full custom, 5
functionality, 5
gain factor, 21
gate
capacitance. See capacitance
  resistance, 40
  state of, 99
gate sizing, 6, 37, 114
comparison, 62
glitch removal, 64
interior point, 48
LANCELOT, 58, 115
linear programming, 44
LOQO, 50
LP solve, 46
nonlinear programming, 57, 115
PCx, 48
PWLa approximation, 45
quadratic programming, 50
simplex, 46
statistical delay model, 99, 114
transition density, 63
geometric program, 33
glitch generation, 18
glitch propagation, 16, 21
glitch removal, 63
glitches, 2, 7
ground lines, 6, 69

hardware, 5

ILP. See integer linear program
inertial delay, 13
input, primary, 88
input vector, 6, 69, 80, 87
  combinations, 97
  reconstruction, 80
integer linear program, 30
relaxation, 74
integral solution, 80
interior point
  accuracy, 28
  feasible area, 28
  infeasibility, 28
method, 28
solver, 27
starting point, 28
termination, 28
inverter, 91

Lagrange
  augmented method, 32
  method of, 25
  multiplier, 25
LANCELOT, 31, 94, 115
layout, 5
LCP. See linear complementary
  problem
leakage current, 20
library, discrete, 126
linear complementary problem, 38
linear program, 25, 69
  dual, 25
  primal, 25
  primal–dual, 27
logarithmic barrier function, 27
logic depth, 84
logic effect, 12
logic gate, 91
logic level, 11, 65, 70, 72, 89
logic synthesis, 5
  combinational, 5
  sequential, 5
LOQO, 26
lower bound maximum power
  estimation, 87
LP solve, 26

maximum power estimation
  formulation, 94
LANCELOT, 94
lower bound, 87
solving, 94
starting point, 95
upper bound, 69
maximum–operation, 100, 102, 107
MILP. See mixed integer linear
  program
mixed integer linear program, 31
module allocation, 5
Monte–Carlo simulation, 100, 101,
  104
MPS, 28

Newton
  iteration, 28
  method, 28
NMOS, 21
non–basic variables, 26
nonlinear constraint, 90
nonlinear optimization, 98
nonlinear programming, 31
nonlinear term, 114
normal distribution, 101, 104
normal form, 26
  restricted, 26

operandi, 116

partitioning, 5
path, 71, 121
PCx, 26
penalty parameter, 32
physical design, 5
piecewise linear, 38
pin, 5
pin–to–pin delay, 10
pipeline stage, 85
pivot, 29
placement, 5
PLATO, 38
PMOS, 21
positive definite, 36
posynomial, 34, 52
power dissipation
  average, 6
dynamic, 20, 43, 71
maximum, 6, 73
short–circuit, 21, 42
sources of, 20
power lines, 6, 69
power supply voltage, 99
precedence constraint, 74
predictability, 3, 6
capacitance, 3
timing, 3
prima–dual linear program, 27
primary input, 79, 88, 97
probability density function, 103
probability distribution, 100
process technology, 20
propagation delay, 10, 11
  pin-to-pin, 11
  sizable gate, 41
propagation delay distribution, 114
PWL. See piecewise linear
quadratic program, 35
real variable, 87, 95
reconverging paths, 102, 112
register transfer level, 5
relaxation, 74
rise time, 10
rising slope, 12
rounding, 95
routing, 5, 100
RTL. See register transfer level
scenario, 69, 73, 89
schedule time, 101
scheduling, 5
semi–definite, 36
SGPD. See sizable gate
  propagation delay
short–circuit power dissipation, 21, 42
SIF, 31
signal uncertainty waveform, 70
  inaccuracy, 70
simplex
  method, 26
  solver, 26
sizable gate propagation delay, 41
slack variable, 25
slope, 12, 126
software, 5
specification, 5
speed factor, 40
stage, 84, 105
standard–cell design, 5
starting point, 28
state of gate, 99
static delay analysis, 99
static timing analysis, 99
statistical calculation, 100
statistical delay model, 99, 114
statistical independence, 102
statistical timing model, 7
stochastic variable, 100
supply noise, 99
switching activity, 22
  weighted, 88
switching event, 89
symmetric, 36
synthesis
  architectural, 5
  logic, 5
  system level, 5
  system level synthesis, 5
technology mapping, 65
  wavefront, 126
temperature, 99
termination, 28
threshold voltage, 21
TILOS, 38
time, rise and fall, 21, 99
time–entry, 71, 89
time–entry list, 71, 89
time–sequence, 71
timing constraint
glitch removal, 63
two–sided, 39, 65
timing model, statistical, 7
transistor
length, 114
width, 114
transistor sizing, 38, 52
transition, 71
rising and falling, 16, 88
transition density, 20, 23
calculation, 22
changes, 63
transitions, number of, 71, 72, 89

uniform distribution, 104
upper bound maximum current

estimation, 83
upper bound maximum power
estimation, 69

variable
complementary, 95
continuous, 90
real, 87, 95
stochastic, 100

waveform, signal uncertainty, 70
wavefront technology mapping, 126
window, 83
windows, number of, 84
wire
delay, 10
width, 114
wiring load, 114
Biography

Etienne Jacobs was born on September 4, 1971 in Tilburg, the Netherlands. He studied Information Engineering at the Eindhoven University of Technology, from which he graduated on August 24, 1995. From September 1995 he has been working towards a doctorate in the Design Automation Section of the Eindhoven University of Technology. He expects to receive this degree based on the work presented in this thesis on April 9, 2001.

From June to September 1997 he visited IBM T.J. Watson Research Center in Yorktown Heights NY, USA, where he worked on JiffyTune. JiffyTune is a fast simulation based transistor sizing tool. JiffyTune can optimize area, performance, power and also improve some parameters to the benefit of manufacturability.

Currently Etienne Jacobs is a member of scientific staff in the Design Automation Section of the Department of Electrical Engineering of the Eindhoven University of Technology. His research interests include: power estimation and synthesis for low power, timing issues in CMOS, design for manufacturability and high level synthesis.