TIMING ANALYSIS AND OPTIMIZATION OF SEQUENTIAL CIRCUITS

TIMING ANALYSIS AND OPTIMIZATION OF SEQUENTIAL CIRCUITS

Naresh Maheshwari
Iowa State University

Sachin S. Sapatnekar
University of Minnesota

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

Library of Congress Cataloging-in-Publication Data

Maheshwari, Naresh, 1970-
Timing analysis and optimization of sequential circuits / Naresh Maheshwari, Sachin S. Sapatnekar.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-4613-7579-1 ISBN 978-1-4615-5637-4 (eBook)
DOI 10.1007/978-1-4615-5637-4
1. Integrated circuits--Very large scale integration--Design and construction--Data processing. 2. Computer-aided design. 3. Time-series analysis--Data processing.
I. Sapatnekar, Sachin S., 1967- . II. Title.
TK7874.75.M35 1998
621.39'5--dc21 98-42471
CIP

Copyright © 1999 by Springer Science+Business Media New York
Second printing 2001.
Originally published by Kluwer Academic Publishers in 1999
Softcover reprint of the hardcover 1st edition 1999

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, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Springer Science + Business Media, LLC.

Printed on acid-free paper.

Dedicated to our parents

Clock, *n.*: A machine of great moral value to man, allaying his concern for the future by reminding him what a lot of time remains to him.

Ambrose Bierce in "The Devil's Dictionary"

Contents

List of Figures ix
Preface xiii

1. INTRODUCTION 1
   1.1 Performance Optimization of VLSI Circuits 1
   1.2 Outline of the Book 5

2. TIMING ANALYSIS OF SEQUENTIAL CIRCUITS 7
   2.1 Introduction 7
   2.2 Combinational Delay Modeling 10
   2.3 Clocking Disciplines: Edge-Triggered Circuits 17
   2.4 Resolving Short Path Violations 18
   2.5 Clocking Disciplines: Level-clocked Circuits 20
   2.6 Clock Schedule Optimization for Level-Clocked Circuits 26
   2.7 Timing Analysis of Domino Logic 27
   2.8 Concluding Remarks 31

3. CLOCK SKEW OPTIMIZATION 33
   3.1 The Notion of Deliberate Clock Skew 33
   3.2 Is Clock Skew Optimization Safe? 35
   3.3 Clock Tree Construction 35
   3.4 Clock Skew Optimization 40
   3.5 Clock Skew Optimization with Transistor Sizing 46
   3.6 Wave Pipelining Issues 60
   3.7 Deliberate Skews for Peak Current Reduction 62
   3.8 Conclusion 63

4. THE BASICS OF RETIMING 65
   4.1 Introduction to Retiming 65
   4.2 A Broad Overview of Research on Retiming 69
   4.3 Modeling and Assumptions for Retiming 72
   4.4 Minperiod Optimization of Edge-triggered Circuits 74

4.5 Level-clocked Circuits 88
4.6 Concluding Remarks 97

5. MINAREA RETIMING 99
5.1 The Leiserson-Saxe Approach 100
5.2 The Minaret Algorithm 105
5.3 Minarea Retiming of Level-Clocked Circuits 113

6. RETIMING CONTROL LOGIC 123
6.1 Minperiod Initial State Retiming via the State Transition Graph 124
6.2 Minperiod Initial State Retiming via Reverse Retiming 127
6.3 Minarea Initial State Retiming 129
6.4 Maintaining Initial States With Explicit Reset Circuitry 144

7. MISCELLANEOUS ISSUES IN RETIMING 147
7.1 Retiming and Testing 148
7.2 Verification Issues 154
7.3 Retiming for Low Power 156
7.4 Retiming with Logic Synthesis 157
7.5 Retiming for FPGA's 161
7.6 Practical Issues 162
7.7 Conclusion 168

8. CONCLUSION 169

References 171

List of Figures

1.1	A typical area-delay tradeoff curve in transistor sizing.	2
2.1	A general sequential circuit.	8
2.2	An example of a pipelined system.	8
2.3	Illustration of the clocking parameters for a register.	9
2.4	An example of a composite cell.	11
2.5	An example of an RC tree.	12
2.6	The Critical Path Method (CPM).	13
2.7	Circuit delays obtained from the Critical Path Method.	15
2.8	Example illustrating the advantages of using level-sensitive latches.	20
2.9	Phase i of a k-phase clock with all times relative to the local time zone.	22
2.10	Illustration of the phase shift operator.	23
2.11	A typical domino circuit.	28
3.1	An Example Circuit.	34
3.2	The advantage of nonzero clock skew.	34
3.3	The perils of zero skew.	35
3.4	H-tree layout for clocking.	36
3.5	Zero-skew merge of two subtrees [Tsa93] ©1993 IEEE.	38
3.6	A merging segment between two points, with $x = \frac{5}{13}$.	39
3.7	A combinational block in a sequential circuit.	40
3.8	Double-clocking.	41
3.9	Zero-clocking.	42
3.10	Example illustrating the importance of combining sizing and skew optimization.	46
3.11	An example illustrating symbolic delay propagation algorithm.	50
3.12	A model of a synchronous sequential circuit.	51
3.13	Clock network routing considerations.	54
3.14	Modified CPM.	57
3.15	Illustrating the idea of wave pipelining.	60
3.16	An example illustrating logic signal separation constraints.	61

4.1	Effect of retiming on clock period	66
4.2	Effect of retiming on number of registers	67
4.3	Equivalent initial states in reverse retiming.	68
4.4	A correlator circuit.	72
4.5	Graph model of the correlator example.	72
4.6	Retimed graph of the correlator.	76
4.7	An example of a circuit with long path violation.	78
4.8	Using clock skew to correct the long path violation.	78
4.9	Retiming for clock period reduction.	78
4.10	Relationship between retiming and skew.	79
4.11	An example circuit.	85
4.12	ASAP retiming of the example circuit, with the ASAP constraint graph.	85
4.13	ALAP retiming of the example circuit, with the ALAP constraint graph.	86
4.14	Effective skews at FF's after ASAP retiming across a gate.	87
4.15	Example illustrating the change in the critical path with the clock period [LE94] ©1994 IEEE.	89
4.16	The latch shift operator.	92
4.17	The ability of a latch to absorb some skew.	93
4.18	Worst-case situation for remaining skew.	96
5.1	Unconditional register sharing at multiple fanouts.	101
5.2	Model for maximum register sharing at multiple fanouts.	102
5.3	Possible FF locations after retiming.	106
5.4	Example illustrating the approach.	109
5.5	An example circuit.	115
5.6	Retiming for clock period optimization.	115
5.7	Alternate retiming for clock period optimization.	116
6.1	(a) Original circuit. (b) Retimed circuit.	123
6.2	Forward retiming of a combinational logic node [TB93] ©1993 IEEE.	125
6.3	Example of an impossible retiming [TB93] ©1993 IEEE.	126
6.4	Adding minimal reset logic to make the retiming possible [TB93] ©1993 IEEE.	126
6.5	Retiming of the modified circuit [TB93] ©1993 IEEE.	127
6.6	Determining the retimed circuit that preserves the initial state [TB93] ©1993 IEEE.	127
6.7	Conventional minarea retiming.	130
6.8	Example of variation in the number of FF's with the initial state.	130
6.9	Another example of variation in the number of FF's with the initial state.	130
6.10	An example circuit where lower bound Γ is achievable.	131
6.11	A minarea retiming without an equivalent initial state.	131
6.12	A minarea retiming with an equivalent initial state.	131

6.13 Effect of justification choices on the number of FF's. 134
6.14 An example of a pruning technique. 136
6.15 Conditional register sharing at multiple fanouts. 137
6.16 An example of FF sharing. 138
6.17 An example of positive retiming. 141
6.18 An example of negative retiming. 141
6.19 The reset mapping [SMB96] ©1996 IEEE. 144
6.20 Explicit reset circuitry. 144
6.21 Retiming using an explicit reset [SMB96] ©1996 IEEE. 145
7.1 Effect of retiming on sequentially redundant faults: (a) the initial circuit, and (b) the retimed circuit [DC94] ©1994 IEEE. 151
7.2 Example illustrating the effect of forward retiming on replaceability. 154
7.3 Retiming structures with gated clocks [Ish93] ©1993 IEEE. 164
7.4 Retiming precharged logic structures [Ish93] ©1993 IEEE. 164
7.5 Effect of FF sharing on delay. 166

Preface

While a decade ago, most books related to integrated circuits began with the phrase, "With the increasing complexity of VLSI circuits...," the new and trendy catch-phrase today is "deep submicron effects." Both of these phrases have been the subject of numerous jests, but it is undeniably true that with the increasing complexity of VLSI circuits and the onset of deep submicron effects, circuit design is becoming more complex. In such an environment, a paradigm shift is being seen, leading to the need for a more synergistic interaction between designers and CAD tools.

[Our excellent spies tell us that the preceding words will not be applauded by the engineers for whom this book is written, and therefore, we feel obligated to switch from the language of Dilbert cliches to plain English.]

On the performance front, there is a vital need for techniques to design fast, low power chips with the minimum area for increasingly complex systems, while on the economic side, there is the vastly increased pressure of time-to-market. This book considers the problem of analyzing and optimizing the timing behavior of sequential circuits with special reference to performance parameters such as power and area. A unified approach to performance analysis and optimization of sequential circuits is presented. The state of the art in analysis and optimization techniques are described for circuits using edge-triggered or level-sensitive memory elements.

The introductory chapter overviews techniques used for performance optimization of combinational and sequential circuits. Chapter 2 describes the edge-triggered and level-triggered clocking disciplines, and compares the two in terms of their relative advantages and disadvantages. Edge triggered circuits are relatively easier to design since they effectively decouple individual combinational blocks and reduce the complexity of the design, and this chapter begins with an overview of some efficient timing analysis techniques for edge-triggered circuits. The design of level-clocked circuits is more complicated than edge-triggered circuits, but offers significant performance enhancements in terms of reductions in area and the ability to operate at higher clock frequencies. The

CAD techniques for clock period verification are correspondingly more computationally expensive. In the next part of the chapter, the state of the art in methods for formulation clocking constraints and verifying the correctness of a clocking discipline is described, followed by a discussion of clock schedule optimization and timing analysis methods for domino circuits.

Chapter 3 describes a technique proposed for improving the circuit behavior, where the clocking network is designed to permit nonzero skews at the flip-flops/latches. A good amount of research has been carried out in this area in the recent past, and it has been found to give large improvements in the circuit performance. After motivating the section with an outline of the potential benefits of deliberate skew, a linear program formulation for optimal skew calculation is described. Next, the process of combining skew optimization with gate sizing, a circuit optimization technique, is detailed. Other applications of deliberate skews are also outlined, followed by a discussion on the design of clocking networks to provide deliberate nonzero skews.

Chapters 4 and 5 deal with retiming, a technique used to improve the performance of the circuit by relocating memory elements. While the concept and a problem solution approach were first outlined in 1983, there has been a flurry of work in the last few years and this method has been transformed from a theoretically interesting, yet computationally expensive technique to one that can handle practical circuit sizes of tens of thousands of gates in minutes.

Chapter 6 addresses the issue of retiming in finite state machines where it is important to retain the initial state of a circuit, or where the power-on states should remain the same. Several new algorithms for each of these problems have been proposed in the recent past, and these are described here. Finally, in Chapter 7, the application of retiming to aid other optimizations is described.

The authors would like to acknowledge a number of sources. Thanks are due to all of the graduate students in our research group and collaborators who have worked on problems related to those discussed in this book. In particular, the work of Weitong Chuang, Rahul Deokar and Harsha Sathyamurthy has contributed greatly to our progress in this area. We would like to acknowledge our collaborators, Professor Ibrahim Hajj at the University of Illinois, and Jack Fishburn at Bell Labs. Jack has been a major source of inspiration to the second author in this work, and a special vote of thanks is due to him. We are grateful to Liang-fang Chao, Madhav Desai, Tony Drumm, Narendra Shenoy and Bapi Vinnakota for several discussions. Professor Sudhakar Reddy introduced us to test issues in retiming, and we are grateful to him for this. We would also like to thank Mahesh Ketkar for his efforts; his careful proofreading of the book has saved us from several major embarrassments.

The work on this book was carried out at Iowa State University and at the University of Minnesota. We owe a debt of gratitude to the faculty, students and staff of both institutions for their support. We would also like to acknowledge the financial support of our funding agencies, specifically the National Science Foundation (contracts MIP-9502556 and MIP-9796305), the Semiconductor Research Corporation (contract 98-DJ-609), Intel Corporation, Digital Equipment

Corporation, Cadence Design Systems, and a Lucent Technologies/DAC Graduate Scholarship. The first author thanks IBM for the opportunity to work with them over two summers. The credits for the cover are due to the excellent graphic designers at the University of Minnesota Printing Services. The help of Carl Harris and the staff at Kluwer Academic Publishers is also acknowledged.

Last, but not the least, we would like to thank our family and friends for their support.

Although we have attempted to provide a reasonably thorough overview of sequential timing optimization methods, we are acutely aware that no such effort can be entirely complete. We would like to apologize, in advance, for any omissions or errors and attribute them to the human factor.

In closing, we reproduce an appropriate quotation from Jean de la Bruyère:

> "Making a book is a craft, as is making a clock."

We hope that our craftsmanship is of adequate quality.

NARESH MAHESHWARI

SACHIN SAPATNEKAR

1 INTRODUCTION

1.1 PERFORMANCE OPTIMIZATION OF VLSI CIRCUITS

CMOS technology has made massive strides over the last two decades and is now the dominant choice for large VLSI designs. Recent projections on changes in technology, such as those reported in [Sem97], have shown that great changes are required in design methodologies of the future to keep pace with Moore's law. Specifically, due to increases in design complexity, it will be essential for designers to place an even greater reliance on computer-aided design (CAD) tools to ensure the required increase in productivity. With timing optimization becoming one of the all-important factors as clock frequencies are forced upwards, CAD strategies for timing optimization are becoming essential. Traditional techniques that consider one combinational block at a time in a sequential circuit can no longer suffice, and more aggressive sequential timing optimization will be required. The objective of this book is to provide an overview of algorithms for this purpose.

In the succeeding sections of this chapter, we begin with a brief overview of some of the methods that may be used for timing optimization. In particular, we will touch upon the use of transistor sizing, clock skew optimization, retiming, local resynthesis, etc.

N. Maheshwari et al., *Timing Analysis and Optimization of Sequential Circuits*
© Kluwer Academic Publishers 1999

1.1.1 Transistor Sizing

During the optimization of a circuit, it must be ensured that the worst-case delay of the circuit meets a specified performance goal. For example, in the case of edge-triggered circuits, it is necessary to ensure that each combinational block in the circuit satisfies the clock period requirements from every input flip-flop to every output flip-flop. For a given circuit topology, the delay can be controlled by varying the sizes of transistors in the circuit, where the size of a transistor is measured in terms of its channel width, since area and power considerations dictate that the channel lengths must be kept at the minimum allowable size. Improvements in the timing behavior of a circuit can be achieved by increasing the sizes of some transistors in the circuit from the minimum allowable size, and these improvements are made at the expense of additional chip area and often, additional power dissipation. Hence, an optimization problem must be solved to arrive at an optimal set of transistor sizes that gives an acceptable tradeoff. A typical area-delay tradeoff in transistor sizing is of the type shown in Figure 1.1. As the delay specification is tightened, the area of the sized circuit increases, but only up to a point; beyond this point, further delay reduction by sizing is not possible. It is also easy to see that the area beyond the knee of the curve corresponds to prohibitive overheads for delay reduction, and this typically does not constitute a good engineering solution.

Figure 1.1. A typical area-delay tradeoff curve in transistor sizing.

Various formulations of the sizing problem may be considered, with one of area, delay or power constituting the objective function, and with constraints on the other two. One practical formulation recognizes that a designer's objective is to achieve the best performance at a given clock period and may be stated as

$$\begin{array}{rc} \text{minimize} & Power(\mathbf{w}) \\ \text{subject to} & Delay(\mathbf{w}) \leq T_{spec} \\ & Area \leq A_{spec} \\ \textit{and} & \text{Each gate size} \geq \text{Minsize} \end{array} \tag{1.1}$$

where both *Delay* and *Power* are functions of the gate sizes, $\mathbf{w} \in \mathbf{R}^n$ (where n is the number of gate sizes), T_{spec} and A_{spec} are, respectively, the constraints on the circuit delay and area, and Minsize is the minimum gate size allowed by the technology.

1.1.2 Interconnect Design

Interconnect is rapidly becoming one of the most important factors in influencing circuit behavior in integrated circuits, and it is widely accepted that interconnect will play a dominant role in next-generation chips. For appropriate delay reduction, it is not enough to optimize the transistor sizes; careful design of the interconnect is essential.

A major issue in the design of VLSI circuits is the construction of clock networks to meet specifications on skew, edge rates, jitter, etc. Traditional methods for clock network design relied on the idea of zero skew to ensure that the clock signal arrived at each memory element at precisely the same time, thereby decoupling the design of clock networks from the design of the logic. However, with advancements in technology, it was found that building a zero-skew clock network was not a trivial task. While earlier techniques in much older technologies could ignore the problem or achieve zero skew by equalizing wire lengths, this is no longer true in modern technologies, where careful delay balancing must be carried out. Moreover, recent work has shown that there are significant performance advantages to be gained by the use of clock networks with deliberately applied nonzero skews. The price that is paid for this improved performance is that the design of the logic cannot be decoupled from the clock network design; however, given the immense benefits of this approach, this method is increasingly gaining acceptance. Further details related to such an approach are presented in Chapter 3.

The design of signal networks is another issue that is changing with evolving technology. In older technologies, the effects of interconnect were, for the most part, easily handled with lumped capacitance models. In recent years, with shrinking feature sizes, this has ceased to be the case, and the resistive and inductive effects are likely to be dominant in the future. The relative contribution of the circuit delay that is attributable to interconnect, as compared to that attributable to gate delays, has been steadily increasing, to the extent that interconnect is foreseen as the chief bottleneck in future designs. Additionally, inductive effects and crosstalk have caused concern with regard to signal integrity. While interconnect issues are not the main subject of this book, it is important for the reader to be aware of them since any discussion on timing is incomplete without some mention of interconnect effects.

1.1.3 Skew Optimization and Retiming

As mentioned above, the conventional approach to circuit design creates a clock distribution network with zero clock skew, i.e., all memory elements obtain the clock signals at precisely the same time. An alternative approach views clock

skews as a manageable resource, rather than a liability. It manipulates clock skews to advantage by intentionally introducing skews to improve the performance of the circuit. The procedure of clock skew optimization, or clock skew scheduling, finds the skews for a circuit for optimized performance. The essence of the idea is to introduce skews in the clocking network to enable time borrowing between adjacent combinational block to achieve better performances or cost-delay tradeoffs.

An alternative method for time borrowing is the procedure of retiming, which moves storage devices across memoryless computational elements to improve the circuit performance without changing the input-output latency. The essential idea of retiming is to relocate latch boundaries to balance the delay of each combinational stage in a sequential circuit. The procedures of retiming and skew optimization are intimately related, and one of the themes of this book is to explore this relationship and to present high-performance algorithms for these two methods.

1.1.4 Resynthesis

Combinational synthesis and technology mapping algorithms have been used for many years to improve the delays of combinational circuits. Sophisticated algorithms have been developed and many are widely used in commercial tools today. The idea of these methods is to build an optimized multilevel (or two-level) combinational circuit that minimizes an objective, such as the area or power dissipation. These methods have been applied in various forms: to optimize a given multioutput Boolean network at early stages of the design process, or to perform local resynthesis moves that have a low layout impact at later stages of the design process.

While these techniques have met with great success, the problem of optimizing a sequential circuit is more demanding. If, for example, two adjacent combinational stages have vastly different numbers of levels of combinational logic, it is likely that movement of the latches may permit a better combinational synthesis. Techniques for this purpose, such as peripheral retiming, or methods for local resynthesis have been used, and constitute a prime weapon to be placed in the armory of the sequential circuit designer.

1.1.5 Additional Optimization Techniques

The list of circuit optimizations is very large, and it is not our purpose to enumerate these options in this chapter. It suffices to mention that methods such as transistor reordering, remapping of complex gates, choice of circuit families and the choice of layout design style are among the alternatives that are available to the circuit designer in attempting to meet the timing specifications. The simultaneous application of all of these methods is likely to be extremely difficult from an algorithmic point of view in that the optimality of the final circuit is hard to guarantee when an algorithm tries to achieve too much at once. However, given this toolbox and a good CAD tool and an experienced

designer manipulating the parameters of the tool, it is possible to get excellent, near-optimal designs from the successive applications of these methods.

1.2 OUTLINE OF THE BOOK

The objective of this book is to provide the reader with a detailed understanding of methods in timing optimization of sequential circuits. Chapter 2 provides an overview of timing analysis techniques used for sequential circuits, and an introduction to a few timing optimization methods. In the remainder of the book, we focus on two methods: clock skew optimization, described in Chapter 3, and retiming, described in detail in Chapters 4 through 7; the thread of skew optimization continues in the latter chapters in that it is used to develop algorithms that find an optimal retiming. The use of other optimizations, listed in the previous section, combined with retiming and skew, is described throughout the book.

2 TIMING ANALYSIS OF SEQUENTIAL CIRCUITS

2.1 INTRODUCTION

A general sequential circuit is a network of computational nodes (gates) and memory elements (registers). The computational nodes may be conceptualized as being clustered together in an acyclic network of gates that forms a combinational logic circuit. A cyclic path in the direction of signal propagation is permitted in the sequential circuit only if it contains at least one register. In general, it is possible to represent any sequential circuit in terms of the schematic shown in Figure 2.1, which has I inputs, O outputs and M registers. The registers outputs feed into the combinational logic which, in turn, feeds the register inputs. Thus, the combinational logic has $I+M$ inputs and $O+M$ outputs.

Pipelined systems, such as the one shown in Figure 2.2, are a special subset of the class of general sequential circuits, and are primarily used in datapaths. A pipelined system uses registers to capture the output of each logic stage at the end of each clock period. Data proceeds through each combinational block, or pipeline stage, until it reaches the end of the pipeline, with the registers serving to isolate individual stages so that they may parallelly process data corresponding to different data sets. The use of this parallelism causes an increase in the data throughput rate of the system. While many pipelines have no feedback, the definition of a pipeline does not preclude the use feedback, and algorithms for scheduling data into a pipeline with feedback have been

N. Maheshwari et al., *Timing Analysis and Optimization of Sequential Circuits*
© Kluwer Academic Publishers 1999

Figure 2.1. A general sequential circuit.

devised [HB84]. Acyclic pipelines form an important subclass of circuits and are popularly used in data-oriented designs as they are much simpler to design, analyze and optimize than general sequential circuits.

Figure 2.2. An example of a pipelined system.

The functioning of registers is key to the behavior of any sequential circuit, and there are several choices of register type that are available to a designer. The behavior of each register is controlled by the clock signal, and depending on the state of the clock, the data at the register input is either effectively isolated from its output, or transmitted to the output. Several types of registers may be used in a synchronous system, and are differentiated by the manner in which they use the clock to transmit data from the input to the output.

Level-clocked latches These are commonly referred to as latches, and permit data to be transmitted from the input to the output whenever the clock is high.

Edge-triggered flip-flops These are commonly called flip-flops (FF's), and use the clock edge to determine when the data is transmitted to the output. In a positive [negative] edge-triggered FF, data is transmitted from the input to the output when the clock makes a transition from 0 to 1 [1 to 0]. FF's can typically be constructed by connecting two level-clocked latches in a master-slave fashion; for details, see [WE93].

A circuit in which all memory elements are level-clocked latches is commonly referred to as a level-clocked circuit, and a circuit composed of edge-triggered FF's is called an edge-triggered circuit.

Other types of storage elements include the RS latch, the JK flip-flop, the T flip-flop, etc. We will not address these in our discussion and will primarily restrict ourselves to level-clocked and edge-triggered registers described above. A description of the transistor-level details of these registers is beyond the scope of this book, but we point out that various choices are available, and they influence the number of transistors used, the number of clock signals to be routed, and the area of and power dissipated by the chip.

Figure 2.3. Illustration of the clocking parameters for a register.

The data input of the register, commonly referred to as the D input, must receive incoming data at a time that is at least T_{setup} units before the onset of the latching edge of the clock. The data will then be available at the output node, Q, after the latching edge. The input, D, must be kept stable for a time of T_{hold} units so that the data is allowed to be stored correctly. The quantities, T_{setup} and T_{hold}, are, unsurprisingly, referred to as the setup and hold times of the latch, respectively. Additionally, each latch has a delay between the time the data and clock are both available at the input, and the time when it is latched; this is referred to as the clock-to-Q delay, T_q.

For a negative edge-triggered register, the setup and hold time requirements are illustrated in Figure 2.3. The clock is a periodic waveform that repeats after every P units of time, where P is called the clock period or the cycle time.

The organization of this chapter is as follows. We will first survey, in brief, techniques for combinational delay modeling. Next, we will consider issues related to correct clocking, first for edge-triggered circuits and then for level-clocked circuits. During this discussion, we will describe techniques for setting up a description of the timing constraints for a circuit, and while this chapter is aimed mainly at timing analysis, we will periodically touch briefly on issues in timing optimization as well.

2.2 COMBINATIONAL DELAY MODELING

Finding the delay of a digital circuit accurately is an important part of the design and verification process. Various levels of simulation can be used, depending on the accuracy desired and the amount of CPU time that is affordable.

Of all techniques currently used to simulate a circuit, circuit-level simulation [CL75] provides the highest degree of accuracy. This method attempts to solve the circuit equations, i.e., the Kirchoff's current and voltage law equations, in conjunction with the device characteristics to find the voltage at every node and the current through every branch of the circuit. The process involves solving systems of nonlinear differential equations. The chief drawbacks of this method are the high computational cost and extensive memory requirements. In addition, due to various problems associated with nonlinear algebraic and differential equations, the convergence of the solution process is not always guaranteed.

To circumvent these disadvantages, techniques such as timing simulation, logic simulation and mixed-mode simulation may be used to compute waveforms of interest using various approximations, to achieve improved computational times at the expense of, it is hoped, small, sacrifices in accuracy.

2.2.1 Gate Delay Models

Often, a rough estimate of the circuit delay can suffice for solving macroscopic design issues, and one need not pay attention to precise voltage or current waveforms. In this context, it is useful to define an important parameter of a voltage waveform, namely, the delay associated with a transition. The transition delay is often defined as the amount of time required by the output waveform to cross the 50% threshold, after the input waveform has crossed its 50% threshold. Several other definitions are possible, but this is arguably the most commonly used one.

Since CMOS logic offers a good deal of modularity in that the analysis of the waveform of a gate can frequently be decoupled from the analysis of the waveform of most other gates in the circuit, modeling at the gate level is an important piece of the combinational timing analysis puzzle. Several approaches have been proposed for delay estimation in a computationally cheap manner. These approaches usually fall in one of the following classes:

- Modeling using RC equivalents: The gate is replaced by an equivalent RC circuit, and the delay of the RC circuit is computed. The advantage of this approach is that the RC circuit delay is typically much easier to estimate than the delay of the gate, and this leads to a significant computational advantage, with some loss in accuracy.

- Macromodeling of template gate types, such as inverters, NANDs, NORs, XORs and complex gates such as AOI and OAI gates, or certain composite cells, such as that in Figure 2.4, by deriving formulæ for the transition time.

Figure 2.4. An example of a composite cell.

The use of resistors and capacitors for transistor level modeling has several advantages over more detailed simulation procedures, although it involves a loss in accuracy. In addition to the major reduction in the computational complexity, the RC representation often offers a simple closed-form formula for the delay of a circuit. Such simple formulæ are particularly useful for delay optimization, since it is far easier, in terms of tractability and computational cost, to perform mathematical operations on this relatively simple delay expression. Another benefit provided by this technique is that the effects of interconnect delay can easily be incorporated by augmenting the RC network that describes the MOS transistor netlist, with additional resistors and capacitors that represent interconnect lines.

The Elmore delay is a good approximation to the dominant time constant of the step response [Wya87]. This is adequate in situations where one time constant dominates; however, for ultra-deep submicron circuits, it is likely that more complex techniques may be required. Reduced order modeling techniques, such as asymptotic waveform evaluation (AWE) [PR90], Padé via Lanczos (PVL) [FF95] and PRIMA [OP97], extend the Elmore idea to nondominant, but important, time constants of the circuit, and may be used.

To properly consider the effects of driver nonlinearities, it is appropriate to consider timing models that derive explicit closed-form solutions for the situation where a driver drives a load that may be either purely capacitive, or an RC load modeled directly or as an effective capacitance [QPP94], or an RLC line. The basic unit in this approach is the channel-connected component (henceforth referred to simply as a component). Each component corresponds to a set of transistors that are connected by drain and source nodes, and the delay model captures the effect of the component driving an interconnect line. A detailed explanation of these approaches is beyond the scope of this book, but the reader is referred to, for example, [SK93, CN94, DMQP94].

Elmore Delay Calculations. The Elmore delay is a useful way of estimating the dominant time constant of an RC circuit. For a fairly complete discussion of Elmore delays, the reader is referred to [SK93]; we will overview the basics of Elmore delay calculations for simple structures in this section.

A commonly-encountered structure is the RC tree, which is a network of resistors and capacitors such that there are no resistor loops (i.e., all of the resistors form a topological tree that does not include the ground node[1]), and all of the capacitors in an RC tree are connected to ground. One of the nonground nodes of the tree is assumed to be the input, and one or more nodes the outputs. Due to the tree structure, there is a unique path through the resistive path of the network from any nonground node to the input node. An example of an RC tree is shown in Figure 2.5. The input node of the network is the one marked *"IN"*.

Figure 2.5. An example of an RC tree.

For an RC tree, the Elmore delay can be calculated by inspection as follows:

- Let P_i be the unique path from the input node $\mathbf{n}_0$ to node $\mathbf{n}_i$.
- Let $P_{ij} = P_i \cap P_j$ denote the portion of the path between $\mathbf{n}_0$ and $\mathbf{n}_i$ that is common to the path between $\mathbf{n}_0$ and $\mathbf{n}_j$.
- Let R_{ij} denote the sum of the resistances in P_{ij}; if $P_{ij} = \emptyset$, then $R_{ij} = 0$.

The Elmore delay to node i in the RC tree is then given by the expression

$$T_{Di} = \sum_{j=0}^{n} R_{ij} C_j. \tag{2.1}$$

Example : Consider the circuit shown in Figure 2.5. The node $\mathbf{n}_0$ is the input node. The procedure described above may be used to compute the Elmore delay to various nodes in the circuit. For example,

$$\begin{aligned} T_{D\mathbf{n}_7} &= R_1C_1 + R_1C_2 + R_1C_3 + R_1C_4 + R_1C_5 + (R_1 + R_6)C_6 \\ &\quad +(R_1 + R_6 + R_7)C_7 + (R_1 + R_6 + R_7)C_8. \end{aligned} \tag{2.2}$$

$$\begin{aligned} T_{D\mathbf{n}_5} &= R_1C_1 + (R_1 + R_2)C_2 + (R_1 + R_2)C_3 \\ &\quad +(R_1 + R_2 + R_4)C_4 + (R_1 + R_2 + R_4 + R_5)C_5 \\ &\quad +R_1C_6 + R_1C_7 + R_1C_8. \end{aligned} \tag{2.3}$$

This calculation is performed by conducting a postorder tree traversal [CLR90] to calculate the downstream capacitance for each node, followed by a preorder traversal to calculate the Elmore delay to each node.

2.2.2 *Delay Calculation at the Circuit Level*

At the circuit level, the delay calculation deals primarily with purely combinational logic circuits. An edge-triggered circuit can be decomposed into combinational logic blocks that lie between latches. Since each of these logic blocks has to satisfy a particular timing requirement dictated by system clock signals, it is the delay of a combinational logic block that has practical significance; the precise forms of these constraints are presented in Section 2.3.

The method that is commonly referred to as PERT (Program Evaluation and Review Technique) and is popularly used in timing analysis is actually a misnomer; it refers to the CPM (Critical Path Method) in project management. In reality, PERT is a technique that is used to propagate probabilistic delay information through a network, while CPM propagates static delay information. The confusion probably arose because one of the earliest works on timing analysis by Kirkpatrick and Clark [KC66] used PERT for statistical timing analysis[2]. These principles were extended by later work for scalar delay values, without necessarily changing the nomenclature. For an overview of the genesis of timing analysis methods, the reader is referred to an early paper by Hitchcock [Hit82]. The CPM procedure is now an integral part of most fast algorithms for circuit delay calculation.

The procedure is best illustrated by means of a simple example. Consider the circuit in Figure 2.6. Each box represents a component. The number within the box represents the delay associated with it. We assume that the worst-case arrival time for a transition at any primary input, i.e., at the inputs to boxes A, B, C, and D, is 0.

Figure 2.6. The Critical Path Method (CPM).

A component is said to be *ready for processing* when the signal arrival time information is available for all of its inputs. Initially, since signal arrival times are known only at the primary inputs, only those components that are fed solely by primary inputs are ready for processing. In the example, these components would be A, B, C, and D. These are placed in a queue and are processed in the order in which they appear in the queue.

In the iterative process, the component at the head of the queue is scheduled for processing. Each processing step consists of

- Finding the latest arriving input to the component, which triggers the output transition. This involves finding the maximum of all worst-case arrival times of inputs to the component.
- Adding the delay of the component to the latest arriving input time, to obtain the worst-case transition time at the output.
- Checking all of the components that the current component fans out to, to find out whether they are ready for processing. If so, the component is added to the tail of the queue.

The iterations end when the queue is empty.

In the example, the algorithm is executed as follows:

Step 1. In the initial step, A, B, C, and D are placed in the queue.

Step 2. A is scheduled. The latest input transition for A is 0, the delay of A is 1; hence, the latest output transition for A is at time (0+1) = 1. No additional elements can be added to the queue.

Step 3. B is scheduled. The latest output transition is found to be at time (0+3) = 3. No additional elements can be placed in the queue.

Step 4. C is scheduled, and the worst-case output transition occurs at time (0+1) = 1. At this point, all input information for component E is available, and it is placed at the tail of the queue.

Step 5. D is scheduled; the output transition time is found to be (0+2) = 2.

Step 6. E is scheduled; the latest arriving input comes in at time 3; hence, the worst-case output transition occurs at time (3+1) = 4. At this point, F and G are added to the tail of the queue.

Step 7. F is scheduled; its latest output transition is at time (4+1) = 5.

Step 8. G is scheduled; its latest output transition occurs at time (4+2) = 6. H is added to the tail of the queue.

Step 9. H is scheduled; its worst-case output transition is at time (6+3) = 9. The queue is now empty and the algorithm terminates.

The worst-case delays at the output of each component are shown in Figure 2.7.

The *critical path*, defined as the path between an input and an output with the maximum delay, can now easily be found by using a traceback method. We begin with the component whose output is the primary output with the latest transition time. This is the last component on the critical path. Next, the latest arriving input to this component is identified. The component that causes this transition is the preceding component on the critical path. The process is repeated recursively until a primary input is reached.

In the example, we begin with component H at the output. The latest arriving input to this component is caused by component G, which thus precedes H on the critical path. Similarly, the transition at G is caused by the latest arriving input, which comes in from component E, and so on. By continuing this process, the critical path from the input to the output is identified as B-E-G-H.

Figure 2.7. Circuit delays obtained from the Critical Path Method.

The method described above can easily be adapted to find the *minimum*, instead of the maximum delay from any input to any output. The only changes in the procedure involve the manner in which an individual component is processed: the earliest arrival time over all inputs is now added to the delay of the component to find the earliest arrival time at its output.

In the case of CMOS circuits, the rise and fall delay transitions are calculated separately. For inverting CMOS gates, the latest arriving input rise (fall) transition triggers off a fall (rise) transition at the output. This can easily be incorporated into CPM described above, by maintaining two numbers, t_r and t_f, for each gate, corresponding to the worst-case rise (high transition) and fall (low transition) delays from a primary input. To obtain the value of t_f at an output, the largest value of t_r at an input node is added to the worst-case fall transition time of the component; the computation of t_r at an output is analogous. For noninverting gates, t_r and t_f are obtained by adding the rise (fall) transition time to the worst-case input rise (fall) transition time.

The CPM procedure does not incorporate the role of false paths, i.e., paths that cannot be excited by any set of input signals. For a detailed treatment of false paths, the reader is referred to [MB91, YH97].

2.2.3 *Calculating the Worst-case Delays from a Specific Input to All Outputs*

The CPM procedure described above finds the worst-case delay from any input to any output in a combinational block. A commonly encountered problem (for example, see Section 3.4) is that of finding the maximum delay, $\overline{d}(i,j)$, from *one* specified input to all outputs. The work in [Fis90] suggested a procedure for computing $\overline{d}(i,j)$ for all j. The arrival time at input i is set to zero, and that at all other inputs to $-\infty$. The resulting signal arrival time at each output j, found using CPM, is the value of $\overline{d}(i,j)$. However, if this procedure were to be performed directly, it would lead to large computation times. This would be accentuated if all input-to-output delays were required, which would mean that

this procedure would have to be executed for each input to the combinational block.

For practical circuits, it was observed during the symbolic propagation of constraints in [CSH95] (described in Section 3.5.1) that in most cases, the input to a combinational block exercises only a small fraction of all of the paths between the inputs of the combinational block and the outputs. Based on this observation, an efficient procedure for calculating the values of $\overline{d}(i,j)$ was developed in [Sap96]. It was found that the use of this procedure yielded run-time improvements of several orders of magnitude over the direct multiple applications of CPM described in the previous paragraph.

The level number, level(k), of each gate k in the circuit is first computed by a single CPM run; the level number is defined as the largest number of gates from a primary input to the gate, inclusive of the gate. In other words, the level number of a gate is found by a topological ordering algorithm. To find $\overline{d}(i,o)$, the largest delay from primary input i to all primary outputs o, an event-driven CPM-like exercise is conducted, starting at flip-flop i, as described in the following piece of pseudocode. During the process, a set of queues, known as level queues, is maintained. The level queue indexed by k contains all gates at level k that have had an input processed. The pseudocode for the procedure is as follows:

```
Algorithm ALL_OUTPUT_DELAYS_FROM_SINGLE_INPUT
Initialize all level queues to be empty;
currentlevel = 0;
currentgate = input i;
while (currentgate ≠ nil) {
   if (currentgate ≠ i)
      for (all fanins j of currentgate)
         d̄(i,currentgate) = max(d̄(i,j)) + delay(currentgate);
   else
         d̄(i,i) = 0;
   for (all fanouts k of currentgate)
      Append gate k to the level queue indexed by level(k),
         if it does not already lie on the queue;
   if (all level queues are not empty) {
      currentlevel = min.  level number with nonempty level queue;
      currentgate = head of the level queue for currentlevel;
      delete currentgate from its level queue;
   }
   else
      currentgate = nil;
}
```

At each step, an element from the lowest unprocessed level is plucked from its level queue, and the worst-case delay from flip-flop i to its output is computed. All of its fanouts are then placed on their corresponding level queues, unless

they have already been placed on these queues. Note that by construction, no gate is processed until the delay to all of its inputs that are affected by flip-flop i have been computed, since such inputs must necessarily have a lower level number.

2.3 CLOCKING DISCIPLINES: EDGE-TRIGGERED CIRCUITS

So far in this chapter, we have focused primarily on timing analysis techniques for combinational circuits. While this is an important problem, the real utility of these procedures is in considering a combinational circuit in the context of how it is embedded in a sequential circuit, as shown in Figure 2.1, and determining whether the sequential circuit meets the requirements imposed by the clock signal or not. We will first perform this analysis for edge-triggered circuits.

In the edge-triggered scenario, let us consider two FF's, i and j, connected only by purely combinational paths. Given all such paths $i \rightsquigarrow j$, let the upper bound on the delay $d(i,j)$ from FF i to FF j be $\overline{d}(i,j)$, and the lower bound be $\underline{d}(i,j)$, i.e.,

$$\underline{d}(i,j) \leq d(i,j) \leq \overline{d}(i,j)$$

We will denote the setup time, hold time and clock-to-Q delay of any arbitrary FF k as T_{s_k}, T_{h_k}, and T_{q_k} respectively, and the clock period as P.

The data is available at the launching FF, i, at time T_{q_i}, and will arrive at the latching FF, j, at a time no later than $T_{q_i} + \overline{d}(i,j)$. For correct clocking, the data is required arrive one setup time before the latching edge of the clock at FF j, i.e., at a time no later than $P - T_{s_j}$. This leads to the following constraint:

$$\begin{aligned} T_{q_i} + \overline{d}(i,j) &\leq P - T_{s_j} \\ \text{i.e., } \overline{d}(i,j) &\leq P - T_{s_j} - T_{q_i} \end{aligned} \tag{2.4}$$

For obvious reasons, this constraint is often referred to as the *setup time constraint.* Since this requirement places an upper bound on the delay of a combinational path, it is also called the *long path constraint.* A third name attributable to this is the *zero clocking constraint*, so christened because the data will not arrive in time to be latched at the next clock period if the combinational delay does not satisfy this constraint.

The data must be stable for an interval that is at least as long as the hold time after the clock edge, if it is to be correctly captured by the FF. Hence, it is essential to ensure that the new data does not arrive at FF j before time T_{h_j}. Since the earliest time that the incoming data can arrive is $T_{q_i} + \underline{d}(i,j)$, this gives us the following *hold time constraint*:

$$\begin{aligned} T_{q_i} + \underline{d}(i,j) &\geq T_{h_j} \\ \text{i.e., } \underline{d}(i,j) &\geq T_{h_j} - T_{q_i} \end{aligned} \tag{2.5}$$

Since this constraint places a lower bound on the combinational delay on a path, it is referred to as a *short path constraint.* If the constraint is violated, then the data in the current clock cycle is corrupted by the data from the next clock cycle; as a result, data is latched twice instead of once in a clock cycle. Hence, this constraint is also called the *double clocking constraint.* Notice that if the clock-to-Q delay of FF i is greater than the hold time of FF j, i.e., $T_{q_i} \geq T_{h_j}$ (this condition is normally not true in practice), then the right hand side of the constraint is negative. In this case, since $\underline{d}(i,j) \geq 0$, the short path constraint is always satisfied.

2.4 RESOLVING SHORT PATH VIOLATIONS

Correct circuit operation requires that all data paths have a delay that lies between an upper bound and a lower bound to satisfy long path and short path constraints. Traditional approaches in delay optimization for combinational circuits have dealt with methods to decrease the delay of the longest path, frequently neglecting to consider the problem of short path violations that may already exist, or may be introduced as a result of the optimizations.

In [SBS93], the issue of satisfying the lower bound delay constraints for edge-triggered circuits is addressed. The short path constraints are handled as a post-processing step after traditional delay optimization techniques by padding delays into the circuit to ensure that it meets all short path constraints. The problem of inserting a minimum number of such delays is referred to as the minimum padding problem. Two methods for solving the problem are described in [SBS93]:

(1) The first simple algorithm finds an edge on a short path that does not lie on any critical long path. A delay is inserted on the edge until one of the following two cases becomes true:

   (a) either some path containing the edge becomes a critical long path, in which case no further delays are added on this edge, and the process is continued for the edge that is identified next, or

   (b) the short path containing the edge meets the lower bound and thus success is achieved in padding.

   This procedure is carried for all the short paths until the lower bound delay constraints are met. This procedure is purely a heuristic and has no guarantees of optimality.

(2) The second approach formulates the problem as a linear program. The combinational circuit is represented as a directed acyclic graph with every gate output, primary input or primary output in the circuit represented by a vertex in the graph. Two type of edges are defined, namely, internal and external edges, denoted by E^I and E^E. An internal edge is directed from an input pin of a gate to its output pin. The external edges represent the interconnections or wiring in the circuit. Hence, if an edge $e_{i,j} \in E^I$, then

the weight $w_{i,j}$ of the edge is the delay of the gate from the input represented by i to the output of the gate. If the edge $e_{i,j} \in E^E$, then the weight $w_{i,j}$ of the edge is a variable whose value is to be determined; the physical meaning of the variable $w_{i,j}$ here is the number of delays padded on the edge $e_{i,j}$. Delays may be inserted only on the external edges. There are two arrival times associated with each vertex i: an early arrival time a_i and a late arrival time A_i. If i is a primary input (PI), then a_i and A_i are specified by the user or the algorithm at the higher level, say, as $a_i = \lambda_i, A_i = \Lambda_i$. If i is an input/output pin of a gate, then

$$\begin{aligned} a_i &= \min_{j \in FI(i)} (a_j + w_{j,i}) \\ A_i &= \max_{j \in FI(i)} (A_j + w_{j,i}) \end{aligned} \tag{2.6}$$

where $FI(i)$ denotes the set of fanins of i. A path is a sequence of vertices, and the delay of a path p is denoted by $d(p)$. At every primary output (PO) i, the data is required to be available no earlier than r_i and no later than R_i; namely, $r_i \leq a_i \leq A_i \leq R_i$. A path p fails to meet the short path constraint if $a_i + d(p) \leq r_i$. In that case, delays $w_{i,j}$ are added along the edge (i,j) on the short path to remove this violation. The area overhead of this addition is proportional to the number of delays added, and is approximated as $\sum_{e_{i,j} \in E^E} w_{i,j}$. The problem of finding the minimum-cost padding that satisfies the timing constraints can now be expressed as a linear program, with constraints obtained by relaxing equation (2.6), and an objective function that corresponds to the number of buffers added.

$$\begin{aligned} \text{minimize} \quad & \sum_{e_{i,j} \in E^E} w_{i,j} \\ \text{subject to} \quad & A_j \geq A_i + w_{i,j} \quad \forall i \in FI(j) \\ & a_j \leq a_i + w_{i,j} \quad \forall i \in FI(j) \\ & A_i \leq R_i \quad \forall i \in PO \\ & a_i \geq r_i \quad \forall i \in PO \\ & A_i = \Lambda_i \quad \forall i \in PI \\ & a_i = \lambda_i \quad \forall i \in PI \end{aligned} \tag{2.7}$$

It was shown that in any circuit, if for every pair $i_1 \in PI$ and $i_n \in PO$ such that $\exists$ a combinational path $p : i_1 \rightsquigarrow i_n$, we are given that $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}$, then the padding problem can be solved. An intuitive explanation for this condition is as follows; if we interpret the term, $A_{i_1} - a_{i_1}$, as the uncertainty interval in the arrival time of the signal at the primary input, i_1, and the term, $R_{i_n} - r_{i_n}$, as the uncertainty in the required time at the output, then since the circuit is causal, it cannot make the uncertainty interval at the output any narrower than the uncertainty interval at its input.

2.5 CLOCKING DISCIPLINES: LEVEL-CLOCKED CIRCUITS

2.5.1 Introduction

As mentioned in Section 2.1, a latch, unlike an FF, is transparent during the active period of the clock. This makes the analysis and design of level-clocked circuits more complex than edge-triggered circuits, since combinational blocks are not insulated from each other by the memory elements, and multicycle paths are possible when the data is latched during the transparent phase of the clock. Even though this transparent nature introduces an additional level of complexity to the design phase, level-clocked circuits are widely used for high-performance designs since they offer more flexibility than edge-triggered circuits, both in terms of the minimum clock period achievable and the minimum number of memory elements required.

As an illustration of this notion, consider the simple circuit in Figure 2.8 with unit delay gates and a single-phase clocking scheme with a 50% duty cycle. Let us assume that the data signals are available at the primary inputs at the falling edge of the clock, and must arrive at the primary outputs before the falling edge. At the level-triggered latch L1, the data may depart at any time while the clock is high. A data signal in this circuit is allowed precisely two clock periods to reach the primary output from the primary input.

Consider the operation of the circuit under a clock period of 2 units; notice that the path delay between latch L1 and the output is more than the clock period. However, the circuit works correctly due to the transparent nature of the latches. As shown in the figure, the data departs from the IN node at time 0, arrives at and departs from the latch L1 at time 1, and is latched at the output at time 4, which corresponds to the onset of the second clock edge. In contrast, if L1 were an edge-triggered FF, then a clock period of 2 units would have been untenable, and the minimum possible clock period would have been 3 units. This example demonstrates the advantages of using level clocked circuits. This practice of using the active period of the clock to allow combinational paths to have a delay of more than the clock period is referred to variously as either cycle borrowing, cycle stealing, slack stealing, or time borrowing.

Figure 2.8. Example illustrating the advantages of using level-sensitive latches.

For edge-triggered circuits, the insulating nature of the memory elements leads to the requirement that the delays through all combinational logic paths must be less than the clock period, with an allowance for setup and hold time constraints. Therefore, timing constraints need only be enforced between FF's connected by a purely combinational path. For level-clocked circuits, the delay through a combinational logic path may be longer than one clock cycle, as long as it is compensated by shorter path delays in subsequent cycles.

To ensure that the extra delay is compensated, we must enforce timing constraints between a latch and every other latch reachable from it via a path that traverses combinational logic, and possibly multiple latches.

Example. Consider a linear N stage pipeline with $N+1$ memory elements $(m_0, m_1 \ldots m_N)$. If these memory elements were edge-triggered FF's, then we would need only N timing constraints $(m_i \rightsquigarrow m_{i+1},\ \ 0 \leq i \leq N)$. However, if these memory elements were to be level sensitive latches, then we would need $N \cdot (N+1)/2$ timing constraints $(m_i \rightsquigarrow m_j\ \forall j > i$ and $0 \leq i \leq N)$ to check the correctness of multicycle paths. In the presence of feedback paths, the timing analysis of level-clocked circuits would become even more complex.

As will soon be shown, some of this complexity can be reduced by the introduction of appropriate intermediate variables.

2.5.2 Clock Model

Single-phase and Multiphase Clocking. The most conventional form of clocking uses a single-phase clock, where every rising [falling] clock edge is perfectly aligned with every other rising [falling] clock edge, assuming that no skews are introduced by the clock distribution network. High-performance circuits may, instead, use multiphase clocking; in a k-phase clock, the k phases typically have the same clock period, but are staggered from each other by a fixed time delay. Each phase consists of two intervals: an active interval during which the latches are enabled, and a passive interval when they are disabled; these typically correspond to the interval when the clock signal is high and low, respectively.

Depending on the way the clock scheme is designed, the phases may be overlapping or nonoverlapping, i.e., their active times may or may not simultaneously intersect in time with any other phase. Clocking schemes may be designed to be symmetric, which implies that all phases have an equal active time, and the time difference between the rising clock edges for successive phases i and j are all equal; a clock scheme that does not satisfy this property is said to be asymmetric.

Although it is not essential for correct operation, if one were to traverse any path in the circuit in the direction of signal propagation, one would typically find that a latch of phase i would be followed by a latch of phase $(i+1) \bmod k$; if so, the circuit is referred to as a well-formed circuit.

The SMO Clock Model. The clock model used by Sakallah, Mudge and Olukotun [SMO92], so named after the initials of its authors, presents a complete description of a multiphase clock signal; this model will be used later in this chapter to describe the set of timing constraints for a level-clocked circuit, as derived by their work.

A k-phase clock is a set of k periodic signals, $\Phi = \{\phi_1 \ldots \phi_k\}$, where $\phi_i, 1 \leq i \leq k$ is referred to as phase i of the clock Φ. All of the ϕ_i's have the same clock period, P. Each phase, i, has an active interval of duration T_{ϕ_i}, and a passive interval of duration $(P - T_{\phi_i})$. Let Ψ denote the set of all latches. Each latch, $j \in \Psi$, is clocked by exactly one phase of the clock Φ, which is denoted by $p(j)$. All of the latches controlled by a clock phase are enabled during the active interval and disabled during the passive interval. Without loss of generality, it will be assumed in this discussion that the active interval corresponds to the clock being at logic value 1, while in the passive interval, the clock is at logic 0. The term "clocking scheme" is used to indicate the relative locations of the clock transition edges, and the duty cycles of the individual phases. Thus, a clocking scheme, together with a clock period, P, defines a clock schedule, Φ.

Associated with each phase, i, is a *local time zone*, shown in Figure 2.9, such that the passive interval starts at time 0, the enabling edge occurs at time $(P - T_{\phi_i})$, and the latching edge occurs at time P. Phases are ordered so that $e_1 \leq e_2 \ldots \leq e_{k-1} \leq e_k = P$, and are numbered modulo-k, i.e., $\phi_{k+1} = \phi_1$ and $\phi_{1-1} = \phi_k$. A global time reference is maintained as a general reference, and e_i denotes the time when the phase ϕ_i ends, relative to this global reference.

Figure 2.9. Phase i of a k-phase clock with all times relative to the local time zone.

The phase shift operator, $E_{i,j}$, illustrated with the help of Figure 2.10, is defined as follows:

$$E_{i,j} = \begin{cases} (e_j - e_i) & \text{for } i < j \\ (P + e_j - e_i) & \text{for } i \geq j \end{cases} \tag{2.8}$$

Note that $E_{i,j}$ always takes on positive values, and acts as a readjustment factor in going from one time zone to another. Specifically, when $E_{i,j}$ is subtracted from a time point in the current time zone of ϕ_i, it changes the frame of reference to the local time zone of ϕ_j.

The clock is required to satisfy the following constraints:

$$T_{\phi_i} \leq P \tag{2.9}$$

$$0 \leq e_i \leq P \tag{2.10}$$

$$e_i \leq e_{i+1} \tag{2.11}$$

Figure 2.10. Illustration of the phase shift operator.

$$P \geq 0 \tag{2.12}$$

$$T_{\phi_i} \geq 0 \tag{2.13}$$

$$E_{i,j} \geq T_{\phi_j} \tag{2.14}$$

The constraints (2.9), (2.10), (2.12) and (2.13) follow from the definition of the clock. The constraint (2.11) maintains an ordering of the phases, and (2.14) ensures that the phases are nonoverlapping. The last of these may be relaxed in case of an overlapping clock.

2.5.3 Timing Constraints for Level-Clocked Circuits

We now enumerate the set of timing constraints that dictate the correct operation of a level-clocked circuit. As before, the parameters $T_{s,i}$ and $T_{h,i}$ represent the setup and hold times, respectively, of the ith latch, and we assume δ_i and Δ_i to be the minimum and maximum delays through each latch. In consistency with previously introduced notation, for each pair of latches, i and j, connected by a combinational path $i \rightsquigarrow j$, the maximum and minimum path delays are denoted by $\overline{d}(i,j)$ and $\underline{d}(i,j)$, respectively.

The data input to each latch, i, has an associated arrival time that lies in the range between the earliest time, a_i, and the latest time, A_i. Similarly, the earliest and latest departure time of the data are denoted by d_i and D_i, respectively. All of these times are with respect to the local time zone of the latch.

Due to the transparent nature of the latches, a signal can depart from latch i at any time during the active interval of the phase $p(i)$, i.e.,

$$P - T_{\phi_{p(i)}} \leq d_i \leq D_i \leq P$$

However, a signal cannot depart from a latch before it has arrived at that latch, i.e.,

$$A_i \leq D_i$$
$$a_i \leq d_i$$

We can combine these constraints to obtain

$$D_i = \max\{A_i, P - T_{\phi_{p(i)}}\}$$
$$d_i = \max\{a_i, P - T_{\phi_{p(i)}}\}$$

The arrival time at a latch, j, of a signal departing from another latch, i, connected by one or more purely combinational paths, $i \rightsquigarrow j$, must satisfy the following relations

$$\begin{aligned} D_i + \Delta_i + \overline{d}(i,j) - E_{p(i),p(j)} &\leq A_j \\ d_i + \delta_i + \underline{d}(i,j) - E_{p(i),p(j)} &\geq a_j \end{aligned}$$

In addition, the latest arrival time must be at least one setup time before the falling edge of the clock, and the earliest arrival time must occur no sooner than a hold time after the rising edge of the clock.

Combining all of these relations, we can obtain the timing constraints that must be imposed for a level clocked circuit to be correctly clocked as

$$A_j = \max_{i \rightsquigarrow j; i,j \in \Psi} \{D_i + \Delta_i + \overline{d}(i,j) - E_{p(i),p(j)}\} \tag{2.15}$$

$$D_i = \max(A_i, P - T_{p(i)}) \; \forall \, i \in \Psi \tag{2.16}$$

$$A_i \leq P - T_{s,i} \; \forall \, i \in \Psi \tag{2.17}$$

$$a_j = \min_{i \rightsquigarrow j; i,j \in \Psi} \{d_i + \delta_i + \underline{d}(i,j) - E_{p(i),p(j)}\} \tag{2.18}$$

$$d_i = \max(a_i, P - T_{p(i)}) \; \forall \, i \in \Psi \tag{2.19}$$

$$a_i \geq T_{h,i} \; \forall \, i \in \Psi \tag{2.20}$$

A critical path of a circuit corresponds to the path corresponding to a sequence of constraints associated with the gates and latches on the path, such that each of these constraints is an active or violated constraint. A critical path in a level-clocked circuit would typically traverse several combinational and sequential circuit elements. The term "critical path of a circuit" is often used colloquially to refer to the path with the largest violation.

Critical paths may be classified into three categories:

Critical long paths are associated with long path constraints, or setup time constraints. Any increase in the delay along the path will worsen the slack or violation along that path.

Critical short paths are those that are associated with short path or hold time constraints. These paths are sensitive to a reduction in the delay along the path; any such reduction is liable to worsen the slack or violation along the path.

Critical cycles correspond to cyclic critical paths in the graph along which the constraints are active or violated.

For correct operation of a circuit, timing violations along all critical paths must be resolved.

Timing Verification. Several procedures for timing analysis have been proposed, based on the SMO constraints, including [SMO92, SS92, Szy92, LTW94, BSM95]. In this section, we provide a brief synopsis of this work; the reader

is referred to these papers for further details. The procedure *checkTc* [SMO90] proposed an iterative procedure for resolving the timing constraints. It was shown in [SS92] that the system of equations can be solved by relaxation in polynomial time in the number of latches, until a fixed point is obtained. It was also shown that the original system of equations in the SMO formulation (not the relaxed versions!) have a unique solution unless the circuit operates at the minimum clock period. The following algorithm, TIMING_VERIFY, shows an outline of the procedure:

```
Algorithm TIMING_VERIFY
/* initialize the latch arrival times */
for all latches i
   A[i] = a[i] = -∞;
/* Iterate the evaluation of the departure and arrival time */
/* equations until convergence or a maximum of |L| iterations, */
/* where |L| is the number of latches.  */
iter = 0;
repeat
   iter++;
   for i = 1 to |L| {
       Aprev[i] = A[i]; aprev[i] = a[i];
   }   /* Update the latch departure time based on the latch */
   /* arrival times computed in the previous iteration */
   for i = 1 to |L| {
       D[i] = max(APrev[i],P - T_p(i));
       d[i] = max(aPrev[i],P - T_p(i));
   }
   /* Update the latch arrival time based on the just-computed */
   /* latch departure times */
   for j = 1 to |L| {
       A[j] = max_i (D[i] + Δ[i] + d̄(i,j) - E(p(i),p(j));
       a[j] = min_i (d[i] + δ[i] + d̲(i,j) - E(p(i),p(j));
   }
until ((A[i] == APrev[i]) && (a[i] == aPrev[i])) || (iter > |L| - 1);

/* Check and record setup and hold time violations */
for i = 1 to |L| {
    SetupVio[i] = (A[i] > P - T_s,i);
    HoldVio[i] = (a[i] < T_h,i);
};
```

The procedure is essentially a variant of the Bellman-Ford algorithm [CLR90], which will be described in detail in Section 3.4.4. In case the constraint set has some slacks, as is almost always the case, a general algorithm that finds values of each a_i, A_i, d_i and D_i that satisfies the above constraints is not obliged to report times that are physically realizable. In contrast, this procedure finds the

earliest arrival and departure times for the signals, corresponding to a physically meaningful solution.

Several variants of the procedure may be used to speed up the computation. For instance, by clipping off the departure times to physically realizable values (for example, setting $D_i = \min(\max(A_i, P - T_{p(i)}), P)$, the information detected is more relevant. If the signal does not arrive in time for the closing edge of the clock, it is signaled as a timing violation. This procedure serves to localize the effect of timing violations, and such a diagnostic provides useful input to procedures that are used to remedy such violations.

An alternative approach to this method utilizes the concept of back-edges [LTW94]. For a directed acyclic graph processed in topological order, the Bellman-Ford procedure (or the algorithm `TIMING_VERIFY` above) concludes in one iteration. It can be shown that if the graph has b back edges, b iterations are necessary. While the problem of identifying a minimal number of back-edges (i.e., the minimum feedback vertex set problem) is NP-hard, in real circuits, it is found that the number of back edges is small enough that almost any set is good enough to provide a good improvement over a naive implementation of the relaxation procedure above.

2.6 CLOCK SCHEDULE OPTIMIZATION FOR LEVEL-CLOCKED CIRCUITS

The clock schedule optimization problem involves finding the minimum clock period, and the corresponding locations of the rising and falling edges of each phase, so that the clocking scheme meets all timing constraints. Due to the freedom allowed in the arrival times for level-clocked circuits, this optimization can make a substantial difference to the behavior of the circuit.

The *minTc* algorithm [SMO90] formulates the clock schedule optimization problem as a linear program. The problem is stated as follows:

$$\begin{array}{ll} \text{minimize} & P \\ \text{subject to} & \text{Clock constraints in relations (2.9) through (2.14)} \\ & \text{Latch constraints in relations (2.15) through (2.17)} \end{array} \tag{2.21}$$

After relaxing all max operators in relations (2.15) through (2.17) by replacing them by "$\geq$" inequalities, the problem maps on to a linear program whose minimum objective function value can be proven to be the same as that of the original problem. Notice that this formulation considers the long path constraints only. In case short path constraints are to be considered, it is not possible to relax them in the same way as long path constraints [BSM95].

The work in [Szy92] developed techniques to reduce the number of long path constraints in the linear program. A principal idea was the concept of *relevant paths*. Before defining this concept, we observe that if $r(p)$ denotes the number of latches on a path $p : e_i \rightsquigarrow e_j$, and $w(p)$ is the delay along the path (with allowances for setup times), then it must be true that

$$w(p) \leq r(p) \cdot P + E_{p(i),p(j)} \tag{2.22}$$

A path, p, is said to be relevant for period P if for any other path, p', with the same endpoints as p, $[w(p') - r(p') \cdot P] < [w(p) - r(p) \cdot P]$ if $r(p') < r(p)$, and $w(p') \leq w(p)$ if $r(p') = r(p)$. It was shown that in order to check the feasibility of a clock schedule, it suffices to check if the constraints on the set of relevant paths are satisfied; the principle behind this is that satisfying the relevant constraint will ensure that other constraints that were not relevant will be forced to satisfy relation (2.22).

The work in [Szy92] uses this pruning to reduce the constraint set before feeding the resulting optimization problem to an LP solver. Subsequently, graph based algorithms for solving the k-phase clock scheduling problem were presented in [SBS92]; graph-based algorithms for clock schedule optimization for two-phase circuitry have also been presented in [ILP92]. For a fixed clock period, the timing constraints can be represented as a set of difference constraints of the type $x_i - x_j \leq c$, where the x_i's are the variables in the problem and c is a constant. This set of constraints can be represented by a constraint graph, and the longest path in the constraint graph yields a solution to the set of difference constraints, if one exists. If the clock period is infeasible, no such solution exists, and this is determined by a procedure such as the Bellman-Ford algorithm. Therefore, a binary search over the clock period is used to find the minimum value of the clock period at which the constraint graph has a solution; this procedure works since the search space is proven to be convex in P^3. The complexity of this algorithm is approximately $O((|\Psi|k^2\log_2|\Psi|)$, where $|\Psi|$ is the cardinality of the set, Ψ, of latches in the circuit, and k is the number of phases of the clock [SBS92].

A vital contribution of [Szy92], subsequently used in many other papers such as [SR94, SD96, MS98b], is the development of the predecessor heuristic to update the value of the search boundary on the detection of a positive cycle in the constraint graph; this is described in more detail in Section 3.4.5.

2.7 TIMING ANALYSIS OF DOMINO LOGIC

Domino logic is a popular choice of design style for implementing high speed logic designs since it has the advantage of small area, fast operation and low power. In this section, we will discuss techniques used for the timing analysis of domino logic.

The basics of domino gate operation may be explained with the help of a representative domino gate configuration is shown in Figure 2.11. When the clock input is low, the gate precharges, charging the dynamic node d to logic 1. In the next half-cycle of the clock when it goes high, the domino gate evaluates, i.e., the dynamic node either discharges or retains the precharged state, depending on the values of the input signals. The two-step mode of operation with a precharge and an evaluate phase causes the timing relationships in domino logic to be more complex than those for static logic.

The clock input to the domino gate is shown in Figure 2.11. The precharge phase begins at $T_{c,f}$ and continues until $T_{c,r}$, and the evaluate phase begins at that time and ends at time $T_{c,f} + P$ where P is the period of the clock signal

feeding the domino gate. The reference time $t = 0$ is set with respect to the clock signal at the primary inputs. If more than one clock signal is used, any one of them may be used as a reference, and the transition times of the other clocks may be expressed according to the reference.

Figure 2.11. A typical domino circuit.

2.7.1 *Timing Constraints*

The node timing constraints for domino logic may be expressed in terms of the signal arrival times and the clock arrival time. Note that in case of multiple clocks for the domino logic, the clock signal c should be set to be the clock signal that feeds the gate that is currently under consideration. The set of constraints were first presented in [VCL+96] and subsequently modified in [VMS96, ZS98]; the following presentation is the most complete of all and is taken from [ZS98].
(i) The first set of constraints addresses the arrival time of a falling transition at any data input of a domino gate. Any such falling event should meet the setup-time requirement to the rising edge of the evaluate clock to ensure that the dynamic node is not inadvertently discharged by a late arriving signal. If $T_f(in)$ refers to the falling event time of the input node, then it is required that

$$T_f(in) \leq T_{c,r} - T_{setup} \tag{2.23}$$

where the setup time T_{setup} is a constant that acts as a safety margin.
(ii) The second set of constraints is related to the arrival time of a rising transition at the output of the domino gate, and is presented here in terms of the arrival time of rising transitions at each of the input nodes. The constraint may be stated as follows: the rising event of the output node of the domino gate must be completed before the falling edge of evaluate clock. If $T_r(out)$ refers to the rising event time at the output node, then the circuit operates correctly only if

$$T_r(out) \leq T_{c,f} + P \tag{2.24}$$

In other words, before the beginning of the precharge for next cycle, the correct evaluation result must have traveled to the output node.

For example, in Figure 2.11, the rising event of the output node of domino gate with output o must satisfy relation (2.24). Therefore,

$$T_r(o) = \max((T_r(x) + D_f(x,d), T_r(y) + D_f(y,d), T_r(z) + D_f(z,d), \\ T_{c,r} + D_f(c,d))) + D_r(d,o) \quad (2.25)$$

where

$T_r(x)$, $T_r(y)$, $T_r(z)$ are the rising event times at inputs x, y and z, respectively
$D_f(i,d)$ represents the delay of a falling transition at the dynamic node d due to a rising transition at input $i \in \{x,y,z\}$
$D_r(d,o)$ represents the rise delay of the inverter feeding the gate output node o
$D_f(c,d)$ is the delay from the clock node c to the dynamic node d

Therefore, for $i \in \{x,y,z\}$,

$$D_f(i,d) + D_r(d,o) - P \leq T_{c,f} - T_r(i) \quad (2.26)$$
$$D_f(i,d) + D_r(d,o) - P \leq T_{c,f} - T_{c,r} \quad (2.27)$$

The relation (2.26) corresponds to the requirement that the rising edge of each input should appear in time for the falling edge of the evaluate clock so as to allow sufficient time for the output to be discharged.

The above formulation considers the delay from each node to the output node separately since these would typically be different. To illustrate this, it can be seen in Figure 2.11 that if the latest arriving input is y, then the fall transition does not need to discharge node n_1. On the other hand, if the latest arriving input is x, then node n_1 may have to be discharged, in addition to the output node o.

The relation (2.27) ensures that the pulse width of the evaluate clock is sufficient for pulling down the output node when the last transistor to switch is the lowermost one, connected to the clock node.

(iii) The third set of constraints addresses the timing requirements on rise transitions at the dynamic node d. The rising event d of the domino gate must be completed before the rising edge of the evaluation clock, i.e.,

$$T_r(d) \leq T_{c,r} \quad (2.28)$$

If the rise time of the dynamic node through the p-transistor fed by the clock is denoted by $D_r(c,d)$, then the rising event time can be expressed as:

$$T_r(d) = T_{c,f} + D_r(c,d) \quad (2.29)$$

This leads to the constraint given by

$$D_r(c,d) \leq T_{c,r} - T_{c,f} \quad (2.30)$$

This implies that the pulse width of precharge must be capable of pulling up the output node.

Note that unlike (ii) above, the delay to only node d is considered here, and not to the output node o. Note that the constraint to node o is very loose and is satisfied except in the most pathological cases. There are two possibilities:

- if the evaluate phase causes the dynamic node to go low, then the completion of the fall transition at o during precharge is irrelevant.
- if the evaluate phase causes the dynamic node to remain high, then it will remain so until the end of the precharge phase in the next clock cycle, and the node o correspondingly has a long time (equal to $P + T_{c,r}$) to discharge, and therefore, the constraint is very loose.

In most cases, the overriding timing constraint will be due to some other constraint, such as that corresponding to a gate at which the signal o is being utilized.

In addition to the constraints in (i) through (iii) above, there is another pair of timing constraints for domino logic.
(iv) The first of these states that in the previous clock cycle, a falling input data line may not go down to logic 0 but must be held at the logic 1 level until the output transition in that cycle has been completed. In other words, it is required that

$$T_f(in) \geq T_{c,r} - P + \max_{\forall in \in I_{domino}} (D_f(in, d) + D_r(d, o)) \tag{2.31}$$

(v) For any rising input data line, the hold constraint states that a transition should occur only after $T_{c,f}$, i.e.,

$$T_r(in) \geq T_{c,f} \tag{2.32}$$

These constraints are similar to the short path constraint in static timing analysis.

The presentations in [VCL+96, VMS96] differ from the above descripton in that they break up constraint (ii) into two separate constraints, and do not describe constraint (iv).

2.7.2 Timing Analysis

The timing analysis procedure described here is based on the CPM procedure. In mixed static-domino circuits, the static elements can easily be represented by nodes in the CPM graph. Dynamic nodes can also be represented as nodes in the CPM graph by representing output rise and fall times by expressions similar to equation (2.25), and then using the relaxed version of the constraint, replacing each "max" by an inequality. This may be intuitively seen by observing that the relations in Section 2.7.1 can be written in terms of difference constraints [CLR90].

The determination of rising and falling event arrival times at the output node of a domino gate is similar to the CPM computation for static gates. A major difference is that the rising event at the dynamic node is related only

to the falling edge of the domino clock and is independent of the other input nodes. This effect is captured by setting the value of D_r from each input node of the domino gate to the output node as $-\infty$. The domino clock input node is treated in the same way as any primary input node, and the rising or the falling edge of the clock provide the corresponding event times for the clock node. In case the clock input to the domino gate is the output of another gate, its event time is given by the maximum of all gate input event arrival times, plus the transition delay of the gate.

2.8 CONCLUDING REMARKS

In this chapter, we have covered the basics of timing analysis for combinational circuits, edge-triggered sequential circuits, and level-clocked circuits, and for domino circuits. We have also made small forays into the domain of timing optimization by discussing methods for minimum padding to resolve short path constraints, and for clock schedule optimization. This presentation forms the basis for our subsequent detailed treatment of two timing optimization methods, clock skew optimization and retiming, in the remainder of the book.

Notes

1. This is in contrast with an RC mesh, which is like an RC tree, except that resistive loops are permitted.

2. The vintage of the article is apparent from the fact that it illustrated the use of PERT on an example of a ferrite core system design.

3. This basic procedure is widely used in many other algorithms for clock skew optimization and retiming, presented in Chapters 3 and 4, respectively, and the reader is invited to read those sections for a fuller understanding.

3 CLOCK SKEW OPTIMIZATION

3.1 THE NOTION OF DELIBERATE CLOCK SKEW

Conventional synchronous circuit design is predicated on the assumption that each clock signal of the same phase arrives at each memory element at exactly the same time. In a sequential VLSI circuit, due to differences in interconnect delays on the clock distribution network, this simultaneity is difficult to achieve and clock signals do not arrive at all of the registers at the same time. This is referred to as a skew in the clock. In a single-phase edge-triggered circuit, in the case where there is no clock skew, the designer must ensure that for correct operation, each input-output path of a combinational subcircuit has a delay that is less than the clock period. In the presence of skew, however, the relation grows more complex and the task of designing the combinational subcircuits becomes more involved.

The conventional approach designs the clock distribution network so as to ensure zero clock skew. An alternative approach views clock skews as a manageable resource rather than a liability. It manipulates clock skews to advantage by intentionally introducing skews to improve the performance of the circuit. To illustrate how this may be done, consider the edge-triggered circuit shown in Figure 3.1, and assume each of the inverters to have a unit delay. This circuit cannot be properly clocked to function at a period of 2 time units, because as shown in the figure, the required arrival time of the signal at register F1 is 2 units, while the data arrives after 3 time units. It is readily verifiable that

N. Maheshwari et al., *Timing Analysis and Optimization of Sequential Circuits*
© Kluwer Academic Publishers 1999

the fastest allowable clock for this circuit has a period of 3 units. However, if a skew of +1 unit is applied to the clock line to register F1, the circuit can operate under a clock period of 2 units. This is possible because as shown in Figure 3.2, the application of a skew of +1 unit delays the clock arrival at register F1 by one unit, thus changing the required data arrival time to the new arrival time of the first clock tick, which is 3 units (i.e., the period of 2 unit delayed by +1 unit). Under these circumstances, the actual data arrival time of 3 units does not cause a timing violation, and the circuit is correctly clocked. A formal method for determining the minimum clock period and the optimal skews was first presented in the work by Fishburn [Fis90], where the clock skew optimization problem was formulated as a linear program that was solved to find the optimal clock period.

Figure 3.1. An Example Circuit.

Figure 3.2. The advantage of nonzero clock skew.

The organization of material in this chapter is as follows. The method of clock skew optimization began as a controversial idea when it was first proposed, but is beginning to be accepted much more widely now; we first address this issue. Next, we briefly describe techniques that may be used to build zero-skew and fixed-skew clock trees. This is followed by clock skew optimization algorithms for period optimization, keeping the structure of the remainder of the circuit unchanged. Finally, the use of clock skew optimization in conjunction with transistor sizing, where the parameters of the circuit are changed at the same time as the optimal skews, and its use for ground bounce reduction, are presented.

3.2 IS CLOCK SKEW OPTIMIZATION SAFE?

A common misconception about changing clock skews has been that it is believed to be an "unsafe" optimization, in that a small change in the gate/interconnect delays may cause a circuit with precariously small tolerances to malfunction. In fact, this is not so; one can build in safety margins, as shown in Section 3.4 that ensure that skewing errors do not disrupt circuit functionality. These safety margins ensure that the circuit will operate in the presence of unintentional process-dependent skew variations. Introducing deliberate delays within the clocking network has been a tactic that has long been used by designers [Wag88]. From the design automation point of view, zero-skew clock tree construction techniques may be adapted to build fixed-skew clock networks. A proof of the practicality of this concept was demonstrated in a pipelined data buffer chip using the concept of skewed clocks, which was designed and fabricated in [HW96].

Figure 3.3. The perils of zero skew.

It is, in fact, a misconception to believe that zero skew is entirely safe. To see this, consider a shift register consisting of register A whose output is connected to register B with *no* combinational logic between the two [SSF95, SSF98], as shown in Figure 3.3. Even for a circuit designed for zero skew, a small unintentional positive skew at register B will cause double-clocking, i.e., a short path constraint violation. Such problems may be avoided by the use of the safety margins mentioned in the previous paragraph and the introduction of deliberate nonzero skew: a small amount of deliberate positive skew at A provides an effective safety margin against short path violations.

3.3 CLOCK TREE CONSTRUCTION

3.3.1 Early Attempts

The clock distribution network is required to propagate the clock signal to every register in a synchronous circuit. Consequently, it must be designed carefully to ensure the optimal use of resources and to achieve performance requirements on the clock signal at the sink nodes. These performance requirements include requirements on the signal arrival time and on the sharpness of the clock edge. The skew constraint on clock networks is a hard constraint, and to ensure the tractability of the problem, many clock networks are constructed as trees. In this section, we will present the essentials of clock tree construction methods targeted at achieving zero skew and fixed skew. Although they will not be discussed here, it is worth pointing out that meshes have been proposed for

clock distribution, for example, in [XK95, DCJ96]. This section is only intended to be an abbreviated reference on clock tree construction; for details, the reader is referred to [Fri95, KR95, CHKM96].

Early methods for clock tree design attempted to achieve zero-skew by balancing the wire length to each sink. H-trees [FK82], illustrated in Figure 3.4 and so named because of their structure, were recognized for years as a technique to help reduce the skew in synchronous systems. The H-tree ensured, by construction, that all sinks of the clock tree are equidistant from the clock source. For regular structures such as systolic arrays, the H-tree works well to reduce skew, but in the general case, asymmetric distributions of clock pins are common and the H-tree may not be effective for clock routing. To address this issue, the method of recursive geometric matching [KCR91] was proposed, and is an extension of the H-tree idea to structures with a smaller degree of regularity.

Figure 3.4. H-tree layout for clocking.

In contrast with the bottom-up approach used in both of these methods, the method of means and medians [JSK90] works in a top-down manner, recursively bipartitioning the routing area into subregions. The centers of these regions are connected appropriately, with the aim of generating at a balanced tree in which the distance from the clock source to the sinks is minimized.

3.3.2 Exact Zero-skew Clock Routing

Each of the methods described so far simplifies the problem of equalizing the clock delays into the geometric problem of equalizing the path lengths on a Manhatan grid. This assumption is valid for a perfectly symmetric clock network in which the load capacitances at all the sinks is precisely the same; however, this

is an invalid assumption for real circuits. The work in [Tsa91, Tsa93] recognized that due to uneven loading and buffering effects, path-length equalization methods do not achieve the effect of balancing clock delays. An approach that incorporates this effect and uses the Elmore delay model to automatically build zero-skew clock trees was suggested. We will now describe this method in detail, and explain how it may be extended to build clock trees with specified, and possibly nonzero, skews at the sink nodes.

The Elmore delay model [RPH83] is used to calculate the signal propagation delay from a clock source to each clock sink. A modified hierarchical method is proposed for computing these delays in a bottom-up fashion. This algorithm takes advantage of the structure of the Elmore delay calculations to construct a recursive bottom-up procedure for interconnecting two zero-skew subtrees to form a new tree with zero skew.

We now describe one recursive step of the bottom-up procedure. In each such step, two subtrees are combined into one subtree; by construction, it is ensured that each such subtree has zero skew from its current root to all of its sinks. In other words, the signal delay from the root to the leaf nodes of the subtree are forced to be equal. In the first step, each sink node constitutes a subtree, and the skew within each such subtree is trivially seen to be equal to zero. In subsequent steps, partially constructed zero-skew subtrees are combined.

In one step of the recursion, two zero-skew subtrees are connected with a wire and a new root is defined for this combined structure, ensuring zero skew from the root to all sinks in the merged tree. This is illustrated in Figure 3.5, and this new root is referred to as the *tapping point.* If we denote the total wire length of this connecting wire segment as l, and the wire length from the tapping point to the root of subtree1 be $x \times l, 0 \leq x \leq 1$, then the wire length from the tapping point to the root of subtree2 is $(1-x) \times l$.

A closed-form expression for the value of x that yields zero skew will now be determined. Consider each subtree in Figure 3.5. Since subtree1 [subtree2] has zero skew, the delay from its root to each leaf node is equal, say t_1 [t_2]. Let the corresponding total capacitances in the subtrees be C_1 [C_2], respectively. The wires of length $x \times l$ and $(1-x) \times l$ are modeled using the RC π model shown in the figure.

The delay from the tapping point to each leaf node in subtree1 is equated to the delay to each node in subtree2 using the Elmore delay formula [SK93] to provide the following equation

$$r_1(\frac{c_1}{2} + C_1) + t_1 = r_2(\frac{c_2}{2} + C_2) + t_2 \tag{3.1}$$

Substituting $r_1 = \alpha x l$, $r_2 = \alpha(1-x)l$, $c_1 = \beta x l$, and $c_2 = \beta(1-x)l$, where α and β are, respectively, the per unit resistance and capacitance of the interconnecting wire, the tapping point is found to correspond to

$$x = \frac{(t_2 - t_1) + \alpha l \left(C_2 + \frac{\beta l}{2}\right)}{\alpha l(\beta l + C_1 + C_2)} \tag{3.2}$$

Figure 3.5. Zero-skew merge of two subtrees [Tsa93] ©1993 IEEE.

If the solution of equation (3.2) results in a value x that satisfies $0 \leq x \leq 1$, then the tapping point lies along the segment interconnecting the two subtrees and is legal. In case x lies outside this range, it implies that the loads on the two subtrees are too disparate. If so, the tapping point is set to lie at the root of the subtree with the larger loading, and the wire connected to the other segment is elongated in order to ensure that its additional delay balances the delay of the first subtree. This procedure of elongating the wire is referred to as *snaking*. If $x < 0$, then the tapping point is kept at $x = 0$ and the connecting wire is elongated by an additional length l'. The value of l' is found by matching the delays in each subtree, and is given by

$$l' = \frac{\sqrt{(\alpha C_1)^2 + 2\alpha\beta(t_2 - t_1)} - \alpha C_1}{\alpha\beta} \tag{3.3}$$

The formula for l' for the $x > 1$ scenario can be derived in a similar manner.

Thus, in this approach the delay time to each node is derived from its immediate predecessor, the branch resistance, and the subtree capacitance. Buffering effects are accounted for in this procedure by storing C_1 and C_2, as the total left and right subtree capacitances, respectively, up to their closest downstream buffer locations; buffers are added to ensure that the clock edge remains sharp.

3.3.3 Further Refinements

The procedure described by the algorithm described above does not attempt to reduce the length of the zero-skew clock tree. The deferred-merge embedding

(DME) algorithm [Eda91, BK92, CHH92] substantially reduces the tree cost while satisfying the skew specifications. In each step of the zero-skew merging procedure described earlier, the two subtrees are connected by a wire of length l. In a Manhattan geometry, there are several routes between two points that have the exact same length. Therefore, the precise route taken by the wire connecting the two zero-skew subtrees is important in that it affects the optimality (but not the correctness under Elmore delays) of future merge operations, where the optimality is defined in terms of minimizing the wire length of the clock tree.

Figure 3.6. A merging segment between two points, with $x = \frac{5}{13}$.

The DME procedure works by deferring the precise route, or embedding, corresponding to each node for as long as possible, and operates in two phases. In the first bottom-up phase, a tree of "merging segments" is constructed, where each merging segment is the locus of possible placements of a specific tapping point in the clock tree, under a Manhattan geometry. An example of a merging segment for two points, S1 and S2, is shown in Figure 3.6, with the value of x in equation (3.2) set to $\frac{5}{13}$. It should be pointed out that the locus of points that form a merging segment for a pair of straight lines segments is another straight line segment; this implies that as the method goes up the recursive merging process, the locus of merging points is always given by a straight line. In the second top-down phase, the exact location of these internal nodes are resolved with the purpose of minimizing the total wire length of the clock tree.

Apart from reducing the delay by minimizing the wire lengths and inserting buffers, one may also use wire width optimization for sharpening the clock edge. Techniques for this purpose have been presented in, for example [Eda93, PMP93, PMOP93]. The problem of optimally inserting buffers in clock trees has been addressed in [CS93b, VMS95, TS97].

The zero-skew merge procedure is extended to RLC clocking networks in [LS96, LS98], using higher-order moment matching methods that are a generalization of the Elmore delay modeling approach. The procedure ensures adequate signal damping and satisfies constraints on the clock signal slew rate.

3.3.4 Fixed Skew Clock Network Construction

The procedures described above can be extended [Tsa93] to build a clock tree that is required to deliver the clock with a predetermined skew at each

sink. This problem is termed the fixed-skew clock network construction, or the prescribed-skew routing problem. As described earlier in this chapter, the major utility of nonzero skews is to enable time-borrowing between combinational circuit segments that are connected by a flip-flop.

The technique of [Tsa91, Tsa93] (and its extensions) can be adapted to solve this problem by adding a fictitious delay element on each clock sink. For a prescribed skew of $+D_i$ at the ith element, the algorithm begins with each sink node corresponding to a (trivial) tree with a delay of $-D_i$, instead of zero as before. Note that according to the notation, a positive skew $+s$ implies that the clock signal is delayed by time s.

Therefore, in principle, in terms of the design automation effort, it is no harder to design fixed-skew clock trees than it is to build zero-skew trees. Several other papers on fixed-skew clock network construction have been published, including [NF96, NF97, HLFC97]. The work in [NF96, NF97] develops a technique that first determines the optimal skew schedule for a circuit, and then constructs a fixed skew network by using the delays of wires and by inserting buffers for additional delays, whle minimizing sensitivity to process and environmental delay variations. The approach in [HLFC97] is distinguished by the fact that it performs a detailed circuit-level design and proves the viability of the process through a fabricated chip.

3.4 CLOCK SKEW OPTIMIZATION

It has been demonstrated at the beginning of this chapter that the use of intentional nonzero clock skews can speed up the clock period for a sequential circuit since it facilitates cycle-borrowing. The work of Fishburn [Fis90] formalizes an approach to find the optimal skews at each memory element in an edge-triggered circuit, where the optimality is defined in terms of minimizing the clock period. The problem is posed as a linear program, and this formulation is described in this section.

Figure 3.7. A combinational block in a sequential circuit.

3.4.1 Timing Constraints

Consider a combinational block in a sequential circuit, as shown in Figure 3.7. Let FF_i and FF_j be a pair of flip-flops at the input and output, respectively, of the combinational block, with skews of x_i and x_j, respectively.

Denote the delay of the combinational block between them as $d(i,j)$, with $\underline{d}(i,j)$ being the minimum delay and $\overline{d}(i,j)$ being the maximum delay). If T_{hold} is the flip-flop hold time, T_{setup} the flip-flop setup time, and P the clock period, then in the presence of skews the double-clocking and zero-clocking constraints may be written as described below, in a manner similar to the zero-skew case discussed in Section 2.3.

Double-clocking/Short path constraint To avoid double-clocking, the data from the next clock cycle should arrive at FF_j no earlier than a time T_{hold} after the current clock. The data leaves FF_i at time x_i and, therefore, the earliest time it can reach FF_j is $x_i + \underline{d}(i,j)$. Thus, the double-clocking constraint illustrated in Figure 3.8 is

$$x_i + \underline{d}(i,j) \geq x_j + T_{hold} \tag{3.4}$$

Figure 3.8. Double-clocking.

Zero-Clocking/Long path constraint To avoid zero-clocking, the data from the current clock cycle should arrive at FF_j no later than a time T_{setup} before the next clock. Since the data leaves FF_i at time x_i, the latest time by which it will reach FF_j is $x_i + \overline{d}(i,j)$. The zero-clocking constraint, shown in Figure 3.9, may be expressed as

$$x_i + \overline{d}(i,j) \leq x_j + P - T_{setup} \tag{3.5}$$

Figure 3.9. Zero-clocking.

3.4.2 Clock Period Minimization

The problem of minimizing the clock period P by controlling the clock skew at each flip-flop, subject to correct timing, can now be formulated as the following linear program

$$\begin{array}{lll} \text{minimize} & P & \\ \text{such that} & x_i + \underline{d}(i,j) & \geq x_j + T_{hold} \\ & x_i + \overline{d}(i,j) & \leq x_j + P - T_{setup} \end{array} \tag{3.6}$$

The linear program assumes that the delays of the combinational segments are constant. These delays can be calculated efficiently by using the method of Section 2.2.2.

The above formulation may be altered to model uncertainties in the clock skews. If it can be guaranteed that in the manufactured circuit, the skew at flip-flop k will be within the range $[x_k - \delta/2, x_k + \delta/2]$, where x_k is the designed value of the skew, then the difference between any skews, $(x_i - x_j)$ in the manufactured circuit, must be within δ of the designed value of $(x_i - x_j)$. A linear program that is guaranteed to produce a functional manufactured circuit is formulated below:

$$\begin{array}{lll} \text{minimize} & P & \\ \text{such that} & x_i + \underline{d}(i,j) & \geq x_j + T_{hold} + \delta \\ & x_i + \overline{d}(i,j) & \leq x_j + P - T_{setup} - \delta \end{array} \tag{3.7}$$

An alternative formulation to maximize the tolerance of the solution to unpredictable changes, for a given clock period P, is as follows. This may be achieved by maximizing the minimum slack over all the constraints, converting the problem into a minmax problem. A new variable M is introduced, and is added to each of the main constraint inequalities, so that maximizing M is tantamount to finding the skew values that maxime the minimum slack over all the inequalities. The precise formulation is as shown below:

$$\begin{array}{lll} \textit{maximize} & M & \\ \textit{s.t.} & x_i + \underline{d}(i,j) - M & \geq x_j + T_{hold}, \\ & x_i + \overline{d}(i,j) + T_{setup} + M & \leq x_j + P \end{array} \tag{3.8}$$

3.4.3 Efficient Solution of Skew Optimization Problem

The constraints of the linear program (3.7) are rewritten as

$$\begin{array}{lll} x_i - x_j & \geq & T_{hold} - \underline{d}(i,j) + \delta \\ x_j - x_i & \geq & T_{setup} + \overline{d}(i,j) - P + \delta, \end{array} \tag{3.9}$$

Note that the skews at the primary inputs and the primary outputs may be set to zero under the assumption that they cannot be controlled or adjusted.

For a constant value of P, the constraint matrix for this problem reduces to a system of difference constraints [CLR90]. For such a system, it is possible to construct a constraint graph. The vertices of the constraint graph correspond to the x_i variables, and each constraint $x_a - x_b \geq c$ corresponds to a directed edge from node b to node a with a weight of c. If P is achievable, then a set of skews that satisfies P can be found by solving the longest path problem on this directed graph. The clock period P is feasible, provided the corresponding constraint graph contains no positive cycles.

This observation was utilized in [DS94] for efficient solution of the skew optimization problem. The optimal clock period is obtained by performing a binary search. At each step in the binary search (for clock period P) the Bellman-Ford algorithm, to be described shortly, is applied to the corresponding constraint graph to check for positive cycles. The procedure continues until the smallest feasible clock period is found. The following skeletal pseudocode is used to perform the binary search for the optimal clock period.

```
Algorithm MINPERIOD_WITH_OPTIMAL_SKEWS
      Construct the constraint graph;
      P_min = P_low;          ...(Lemma 3.3 below)
      P_max = P_high;         ...(Lemma 3.4 below)
      while (P_max - P_min) > ε {
         P = (P_max + P_min)/2;
         if constraint graph has a positive cycle
            P_min = P;
         else
            P_max = P;
      }
```

The following results are used to show the correctness of the binary search; for proofs, the reader is referred to [DS94, SD96].

Lemma 3.1 : If $P = P_1$ does not permit a feasible solution to the linear program (3.6), then nor does $P = P_2 < P_1$.

Corollary 3.2 : If $P = P_1$ permits a feasible solution to the linear program (3.6), then so does $P = P_2 > P_1$.

Next, two lemmas that provide bounds P_{low} and P_{high} on the optimal clock period, are presented.

Lemma 3.3 : Given a pair of inequalities in relation (3.9), define

$$P_{ij} = \overline{d}(i,j) - \underline{d}(i,j) + T_{setup} + T_{hold} + 2\delta \tag{3.10}$$

A lower bound, P_{low}, on the value of the optimal clock period is given by

$$P_{low} = \max_{all(i,j)} P_{i,j} \geq 0. \tag{3.11}$$

where the max operates on all pairs (i, j) of FF's that are connected by a combinational path.

Lemma 3.4 : If the original circuit satisfies all short path constraints, then an upper bound, P_{high} on the value of the optimal clock period is given by

$$P_{high} = \max_{all(i,j)} (\bar{d}(i,j) + T_{setup} + \delta) < \infty \tag{3.12}$$

As before, the max operates on all pairs (i, j) of FF's that are connected by a combinational path.

3.4.4 Applying the Bellman-Ford Algorithm

As stated earlier, each step of the binary search applies the Bellman-Ford algorithm to solve the longest path problem on the constraint graph to determine whether P is feasible or not, and if so, to find the values of the corresponding skews. A brief description of the procedure follows.

The Bellman-Ford algorithm may be applied to solve a single-source longest path problem on a directed graph. Unlike Dijskstra-like algorithms, it can handle edge weights of either sign, but has a higher computational complexity. The pseudocode for the application of the algorithm to a graph $G = (V, E, \mathbf{w})$, where V and E are the set of vertices and edges, respectively, of the graph, and $\mathbf{w}$ is the vector of edge weights, is as follows:

```
Algorithm BELLMAN_FORD
      for each vertex v ∈ V
          x_v ← 0;
          predecessor(v) ← ∅;
      for i ← 1 to |V|-1
          if x_v < x_u + w(u,v) then
             x_v ← x_u + w(u,v);
             predecessor(v) ← u;
      for each edge (u,v) ∈ E
          if (x_v < x_u + w(u,v)) then
             /* A positive cycle exists */
             SATISFIABLE ← FALSE;
          else
             SATISFIABLE ← TRUE;
```

If the skews are initialized to 0, the Bellman-Ford solution achieves the objective of minimizing $|x_{i,max} - x_{i,min}|$ [CLR90]. On a graph with V vertices and E edges, the computational complexity of this algorithm is $O(V \cdot E)$.

In the solution found above, all skews must necessarily be positive, since the weight of each node in the Bellman-Ford algorithm was initialized to zero. Also, in general, the skew at the host node (corresponding to primary inputs and outputs) could be nonzero at the end of these iterations. Since the objective is to ensure a zero skew at the primary input and output nodes, the solution

is modified to achieve this. Note that if $[x_1, \cdots x_n]$ is a solution to a system of difference constraints in $\mathbf{x}$, then so is $\mathbf{x}' = [(x_1 + k), (x_2 + k), \cdots, (x_n + k)]$. Therefore, by selecting k to be the negative of the skew at the host node, a solution $\mathbf{x}'$ with a zero skew at the host is found.

3.4.5 *Complexity Analysis and Efficiency-enhancing Methods*

The number of iterations of the binary search, in the worst case, is $(P_{high} - P_{low})/\epsilon$. The time required to form the constraint graph may be as large as $|f| \cdot |G|$, where $|f|$ is the maximum number of inputs to any combinational stage. In practice, though, it is seen that this upper bound is seldom achieved. Therefore, the iterative procedure above, when carried to convergence, provides the solution to the linear program (3.7) with a worst-case time complexity of

$$O\left(F \cdot E \cdot log_2\left[(P_{high} - P_{low})/\epsilon\right] + |f| \cdot |G|\right) \tag{3.13}$$

where F is the number of flip-flops in the circuit, E is the number of pairs of flip-flops connected by a combinational path, P_{high} and P_{low} are as defined in Lemmas 3.3 and 3.4, and ϵ, defined in the pseudocode above, corresponds to the degree of accuracy required.

We caution the reader that the complexity shown above is not a genuine indication of the complexity for real circuits. Firstly, in practical cases, it is seen that $E = O(F)$. Secondly, if the implementation is cleverly carried out, using back-pointers during the Bellman-Ford process, to be described in the remainder of this section, and the procedure in Section 2.2.3 for $\overline{d}(i, j)$ calculations, the computational cost can be substantially reduced.

Instead of dividing the interval into half during each step of the binary search, it is possible to use circuit information to reduce the computation and update the search space more efficiently [Szy92]. Suppose that for some value of P, the constraint graph was found to be infeasible. This implies that a positive cycle exists, and each edge on that positive cycle has either a constant weight, if it corresponds to a short-path constraint, or a weight of the form (constant - P), if it is a long-path constraint edge. Given any positive cycle, it is possible to sum up these weights on the edges and update the bound on P so that any value of P that is less than this bound would force this cycle to be infeasible. This reduces the binary search space.

The detection of a positive cycle proceeds as follows. During the Bellman-Ford procedure, a back-pointer `predecessor(v)` is maintained from each vertex v to the predecessor vertex that updated its value; in other words, the edge between these vertices corresponds to an active constraint. After the Bellman-Ford procedure is completed, if the constraint graph is infeasible, a reverse trace is carried out from the vertex v for which it was found that $x_v > x_u + w(u, v)$. This node v must necessarily lie on a positive cycle. The procedure steps back to the predecessor of v, and further back, until v is reached again; at this time, a positive cycle in the graph has been detected. The sum of the weights of all edges on this cycle is calculated as a function of P. As stated in the previous paragraph, this sum is a linear function of P of the form $k_1 - k_2 P$; the lower

bound on the clock period is updated to k_1/k_2 to ensure that this cycle will not have a positive weight. This procedure can be repeated on every cycle in the graph of predecessor edges; note that the number of edges in such a graph is $O(|V|)$, where $|V|$ is the number of vertices, which means that this can be performed inexpensively. Similarly, if the clock period is feasible, the graph of predecessor edges may be used to update the upper bound on the clock period by exploring the amount by which the clock period could be lowered before the predecessor graph would be altered.

Note that the solution to the problem of clock period minimization corresponds to the maximum P over all positive cycles; it is, in general, not possible to find this maximum in polynomial time since the number of cycles could be exponential in the number of vertices. However, it has been seen that from a practical standpoint, the back-pointer procedure works well in reducing the amount of computation.

3.5 CLOCK SKEW OPTIMIZATION WITH TRANSISTOR SIZING

In the discussion so far, it has been assumed that the delay of each combinational path is fixed. However, the drive strength of gates can be altered by transistor sizing [SK93], thereby changing the combinational path delays. Treating clock skew optimization and transistor sizing as separate optimizations is liable to lead to a suboptimal solution, and a synergistic blend of the two procedures can greatly improve the cost-performance tradeoff for the circuit.

To illustrate this, consider a pair of combinational stages separated by latches, as shown in Figure 3.10(a). We will consider area as the cost function here, but the same idea is valid for other correlated cost functions such as power. If only the sizing transformation were to be applied, the nature of the cost-delay tradeoff curve for each stage would be as shown in Figure 3.10(b). If the timing specification on Stg_1 were stringent, then the optimal sizing solution would lie beyond the knee of the curve, at a point such as point A. Similarly, if the specifications on Stg_2 were loose, then the optimal solution might lie at a point such as C. The cost of the sizing solution would then be [Area(A) + Area(C)][1].

Now consider a situation where a small skew, S, is applied to the registers between the two stages, allowing Stg_1 to borrow time from Stg_2. This would lead to a looser specification on Stg_1, bringing the solution point from A to B, and a tighter specification for Stg_2, moving its solution from C to D. The cost of the overall sizing solution, [Area(B) + Area(D)], can be seen to be much smaller than the cost with zero skew, even for a small amount of cycle borrowing, since the cost at the solution point A was well above the knee of the curve[2]. Therefore, the judicious use of sizing in conjunction with deliberate skew can deliver solutions of significantly better quality than sizing alone. It is important to perform the two optimizations concurrently, rather than sequentially, to fully incorporate the mutual interactions between skew and sizing. This section presents three techniques for simultaneous optimization of skews and transistor sizes.

Figure 3.10. Example illustrating the importance of combining sizing and skew optimization.

3.5.1 *A Linear Programming Formulation*

Problem Formulation. The set of long path constraints in the presence of clock skews can be seen from inequation (3.5) to be linear in the skew variables and the gate delays. If the gate delays could also be represented in terms of linear functions of the transistor sizes, then the set of timing constraints on a circuit would be linear. This idea forms the basis of the linear programming formulation [CSH93, CSH95] discussed in this section.

The delay of a gate is represented as a convex piecewise linear function of its own size and that of its fanout gates. The delay $D(w_k, w_{k,1}, \cdots, w_{k,fo(k)})$ of a gate with size w_k, and with fanout gates of sizes $w_{k,1} \cdots w_{k,fo(k)}$, can be represented using a convex piecewise linear function with q regions as:

$$D(w, w_{k,1}, \cdots, w_{k,fo(k)}) = \begin{cases} a_1 \cdot w + b_{1,1} \cdot w_{k,1} + \cdots + b_{1,f} w_{k,fo(k)} + c_1 \\ \qquad (w, w_{k,1} \cdots w_{k,fo(k)}) \in \text{Region } \mathbf{R}_1 \\ a_2 \cdot w + b_{2,1} \cdot w_{k,1} + \cdots + b_{2,f} w_{k,fo(k)} + c_2 \\ \qquad (w, w_{k,1} \cdots w_{k,fo(k)}) \in \text{Region } \mathbf{R}_2 \\ \vdots \\ a_q \cdot w + b_{q,1} \cdot w_{k,1} + \cdots + b_{q,f} w_{k,fo(k)} + c_q \\ \qquad (w, w_{k,1} \cdots w_{k,fo(k)}) \in \text{Region } \mathbf{R}_q \end{cases} \tag{3.14}$$

$$= \max_{1 \le j \le q} (a_j \cdot w + b_{j,1} \cdot w_{k,1} + \cdots b_{j,f} w_{k,fo(k)} + c_j)$$
$$\forall\, (w_k, w_{k,1} \cdots w_{k,fo(k)}) \in \bigcup_i \mathbf{R}_i. \tag{3.15}$$

The second equality follows from the first since $D(w_k, w_{k,1}, \cdots, w_{k,fo(k)})$ is convex.

The variables m_k^i (p_k^i) are introduced to represent the longest (shortest) delay from each flip-flop i to gate k. The introduction of the m and p intermediate variables ensure that the number of constraints depends on the number of gates rather than the number of combinational paths; the number of constraints in the latter case would have been prohibitively large. As before, the skew at flip-flop i is denoted as x_i, while w_k and d_k are the width and delay of gate k respectively. The value of the skew is set to a constant (typically, 0) for primary inputs and outputs. The skew optimization and sizing problem for a general synchronous sequential circuit can now be formulated as follows.

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{k=1}^{\mathcal{N}} \gamma_k \cdot w_k & \\
\text{subject to} & d_k \geq D(w_k, w_{k,1}, \ldots w_{k,fo(k)}), & 1 \leq k \leq \mathcal{N} \\
& w_k \geq \text{Minsize}(k), & 1 \leq k \leq \mathcal{N} \\
& w_k \leq \text{Maxsize}(k), & 1 \leq k \leq \mathcal{N} \\
& \text{For all flip-flops } i, & 1 \leq i \leq \mathcal{L} \\
& \quad x_i + p_k^i \geq x_j + T_{hold} & 1 \leq j \leq \mathcal{L}, \;\; k = Fanin(FF_j) \\
& \quad x_i + T_{setup} + m_k^i \leq x_j + P_{spec} & 1 \leq j \leq \mathcal{L}, \;\; k = Fanin(FF_j) \\
& \quad \text{For all gates } \; k = 1, \cdots, \mathcal{N} & \\
& \qquad m_l^i + d_k \leq m_k^i, & \forall\, l \in Fanin(k) \\
& \qquad p_l^i + d_k \geq p_k^i, & \forall\, l \in Fanin(k)
\end{array}
\tag{3.16}
$$

Here, γ_k is the *area coefficient*, a constant associated with gate k, so that if gate k has size w_k, then its area is $(\gamma_k \cdot w_k)$. In the notation used above, $\mathcal{N}$ is the number of gates, $\mathcal{L}$ is the number of flip-flops in the circuit, and $Fanin(i)$ is the set of gates or flip-flops at the fanin of a gate or flip-flop i.

The problem formulation above is a linear program in the variables d_i, m_i, p_i, s_i and w_i. The entries in the constraint matrix are very sparse, which makes the problem amenable to fast solution by sparse linear programming approaches.

Reducing the Number of Constraints. For large circuits the number of constraints in the above linear program can be tremendous. A symbolic propagation method is used to prune the number of constraints by a judicious choice of the intermediate variables, m and p, without sacrificing accuracy. Although the actual reduction is dependent on the structure of the circuit, experimental results showed that this algorithm can reduce the number of constraints to less than 10% of the original number for most tested circuits in [CSH95].

We begin by counting the number of linear constraints in (3.16), ignoring the constraints on the maximum and minimum sizes of each gate since these are handled separately by the simplex method. The d_k inequalities impose q constraints for each of the gates in the circuit to the linear programming formulation, where q is given in (3.15). Let $\mathcal{F} = \sum_{i=1}^{\mathcal{N}} Fanin(i)$. Then for each flip-flop i, there are $O(\mathcal{F} + \mathcal{L})$ constraints, where $\mathcal{L}$ is the total number of flip-flops in the circuit. Therefore, the total number of constraints could be as large as $O(\mathcal{N} \cdot q + \mathcal{L} \cdot (\mathcal{F} + \mathcal{L}))$. Assume that the average number of fanins to a gate is 2.5 and $q = 5$. Then $\mathcal{F} = 2.5\mathcal{N}$, and $\mathcal{L} \cdot \mathcal{F}$ is the dominant term in the expression above. For real circuits, $\mathcal{L}$ is large, and hence, the number of constraints could be tremendous. The symbolic constraint propagation method presented here prunes the number of constraints by attempting to minimize the intermediate variables m and p, without sacrificing the exactness of the original formulation. For any primary input or flip-flop, the m and p variables are introduced only for those gates that lie in that primary inputs fanout cone. Moreover, constraints on chains of gates are collapsed wherever possible by introducing a new m or p variable only when essential. The pseudocode for the procedure is shown below:

```
Algorithm SYMBOLIC_PROPAGATION()
for i = 1 to L {
   ∀ gates j, w_j ← 0; mstring(j) ← ""; pstring(j) ← "";
   for j = 1 to max_level {
     for each gate k at level j {
       if (w_l = 0 ∀ l ∈ fanin(k)) continue; /* do nothing */
       if (exactly one w(l) = 1 for l ∈ fanin(k)) {
         mstring(k) ← mstring(l) + "d_k";
         pstring(k) ← pstring(l) + "d_k";
         w_k ← 1;
       }
       else {
         mstring(k) ← "m_k^i";
         pstring(k) ← "p_k^i";
         w_k ← 1;
         for all w_l = 1, l ∈ fanin(k) {
           write out the constraints
            mstring(l) + d_k ≤ m_k^i;
            pstring(l) + d_k ≤ p_k^i;
         }
       }
     }
   }
}
```

The sequential circuit is first levelized. For this purpose, the inputs of flip-flops are considered as pseudo-primary outputs and the outputs of flip-flops

Figure 3.11. An example illustrating symbolic delay propagation algorithm.

are considered as pseudo-primary inputs. Two string variables, $mstring(i)$ and $pstring(i)$, are used to store the long-path delay and short-path delay constraints associated with gate i, respectively. For each gate and each primary input (including pseudo-primary inputs), an integer variable $w_i \in \{0, 1\}$ is introduced to indicate its status. w_i has the value 1 whenever $mstring(i)$ and $pstring(i)$ are non-empty, i.e., when the constraints stored in $mstring(i)$ and $pstring(i)$ must be propagated; otherwise, $w_i = 0$.

In the following discussion of the algorithm, we elaborate on the formation of *mstring*; the formation of *pstring* proceeds analogously. For each gate j, w_j and $mstring(j)$ are initialized by setting $w_j = 0$, and $mstring(j)$ to the null string. If $w_l = 0$ for all $l \in fanin(k)$, i.e., if all of gate k's input gates have a null *mstring*, then no constraints need to be generated or propagated. If exactly one of all of gate k's input gates, say gate l, has a non-empty *mstring*, and all others are have null *mstring*'s, then the constraint is propagated, rather than introducing a new m or p variable. This is implemented by concatenating $mstring(l)$ and "d_k", and storing the resulting string in $mstring(k)$. Also, w_k is set to 1 to indicate that further propagation is required beyond this gate. Finally, if more than one of gate k's input gates have a non-empty *mstring*, then a new intermediate variable, m_k^i is added, and the string "m_k^i" is stored at $mstring(k)$. For each input gate whose *mstring* is non-empty ($w_l = 1$), a delay constraint is needed.

Example: Figure 3.11 gives an example that illustrates the symbolic delay constraint propagation algorithm for calculating *mstring* constraints only. Assume that $mstring(11)$ = "m_{11}^1", and $mstring(12) = mstring(13)$ = "", the null string. Since gate 14 has only one input with a w value of 1, this implies that $mstring(14)$ = "$m_{11}^1 + d_{14}$" and $w_{14} = 1$. Propagating this further, it is found that similarly, $mstring(15)$ = "$m_{11}^1 + d_{14} + d_{15}$", and $w_{15} = 1$. Finally, for gate 16, since more than one input has a w value of 1, a new variable, m_{16}^1, must be introduced and w_{16} is set to 1. The following constraints may now be written and appended to the set of constraints for the linear program:

$$m_{11}^1 + d_{14} + d_{15} + d_{16} \leq m_{16}^1$$

Figure 3.12. A model of a synchronous sequential circuit.

$$m_{11}^1 + d_{14} + d_{16} \leq m_{16}^1$$

Partitioning Large Sequential Circuits. The number of constraints in the linear programming formulation is proportional to the product of the number of gates and the number of flip-flops in the circuit in the worst case. Ideally for a given sequential circuit, all variables and constraints should be considered together to obtain an optimal solution. However, for large sequential circuits, the size of the linear program could become prohibitive, even after applying the pruning methods embedded in the symbolic constraint propagation algorithm. Therefore, it is desirable to partition large sequential circuits into smaller, more tractable subcircuits, so that the algorithms can be applied to each subcircuit. While this would entail some loss of optimality, an efficient partitioning scheme would minimize that loss; moreover the reduction of execution time would be very rewarding.

It is well-known that multiple-way network partitioning problems are NP-hard. Therefore, typical approaches to solving such problems find heuristics that will yield approximate solutions in polynomial time [San89, YC91, She95]. Traditional partitioning problems usually have explicit objective functions; for example, in physical layout it is desirable to have minimal interface signals resulting from partitioning the circuit, and hence, the objective function to be minimized there is the number of nets connecting more than two blocks. The sequential circuit partitioning problem, however, is made harder by the absence of a well-defined objective function; since the ultimate goal is to minimize total area of the circuit, there is no direct physical measure that could serve as an objective function for partitioning. In the remainder of this section, we present a heuristic measure that will be shown to be an effective objective function for the partitioning problem.

To help in the description of the partitioning algorithm, the following terminology is introduced. For a synchronous sequential circuit, such as one shown in Figure 3.12,

An **internal latch** is a latch whose fanin and fanout gates belong to the same combinational block.

A **sequential block** consists of a combinational subcircuit and its associated internal latches.

Boundary latches are latches that act as either a pseudo-primary input or a pseudo-primary output (but not both) to a combinational block, i.e., latches whose fanin and fanout gates belong to different combinational blocks.

A partition of a synchronous sequential circuit $\mathbf{N}$ is the division of the sequential blocks of $\mathbf{N}$ into disjoint groups. A b-way partitioning of the network is described by the b-tuple $(\mathbf{G}_1, \mathbf{G}_2, \ldots \mathbf{G}_b)$ where the $\mathbf{G}_i's$ are disjoint sets of sequential blocks whose union is the entire set of blocks in the network. Each $\mathbf{G}_i$ is said to be a *group* of the partition.

For a given sequential block $\mathbf{B}$, let $L_{\mathbf{B}}$ denote the set of boundary latches incident on $\mathbf{B}$, and for a given boundary latch L, $\mathbf{B}_L$ denotes the set of sequential blocks on the latch L. For each boundary latch L, the *input tightness* τ_{in}, *output tightness* τ_{out}, and the *tightness ratio* τ are defined as

$$\begin{aligned}
\tau_{in}(L) &= \text{maximum combinational delay from any boundary latch to } L \\
&\quad \text{in the unsized circuit,} \\
\tau_{out}(L) &= \text{maximum combinational delay from } L \text{ to any boundary latch} \\
&\quad \text{in the unsized circuit,} \\
\tau(L) &= \begin{cases} \tau_{in}/\tau_{out} & \text{if } \tau_{in} \geq \tau_{out} \\ \tau_{out}/\tau_{in} & \text{if } \tau_{in} < \tau_{out} \end{cases}
\end{aligned} \tag{3.17}$$

where the adjective "unsized" implies that all gates in the subcircuit are at the minimum size. The tightness ratio $\tau(L)$ provides a measure of how advantageous it would be to provide a skew at L.

For each pair of blocks $(\mathbf{B}_i, \mathbf{B}_j)$, define merit μ_{ij} as

$$\mu_{ij} = \sum_{\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j} \tau(L_k) \tag{3.18}$$

where $\mathbf{B}_i \overset{L_k}{\leftrightarrow} \mathbf{B}_j$ means that latch L_k lies between $\mathbf{B}_i$ and $\mathbf{B}_j$. μ_{ij} is defined to be 0 if $\mathbf{B}_i$ and $\mathbf{B}_j$ are disjoint. Physically, μ_{ij} is used to measure the figure of merit if $\mathbf{B}_i$ and $\mathbf{B}_j$ are in the same group.

The cost associated with each block, $\mathbf{B}_i$, is c_i, the number of linear programming constraints required for solving $\mathbf{B}_i$. This number can be calculated very efficiently. Assume that group $\mathbf{G}_k$ consists of blocks $\mathbf{B}_{ki}, i = 1, \ldots |\mathbf{G}_k|$. Then the cost of $\mathbf{G}_k$ is defined as $C(\mathbf{G}_k) = \sum_{i=1}^{|\mathbf{G}_k|} c_{ki}$, and the merit of $\mathbf{G}_k$, $M(\mathbf{G}_k) = \sum_{i=1}^{|\mathbf{G}_k|} \sum_{j=i+1}^{|\mathbf{G}_k|} \mu_{ij}$. The problem is now stated as

$$\begin{aligned}
\text{maximize} \quad & \sum_{k=1}^{N} M(\mathbf{G}_k) \\
\text{subject to} \quad & C(\mathbf{G}_k) < MaxConstraints.
\end{aligned} \tag{3.19}$$

where N is the number of groups, and $MaxConstraints$ is the maximum number of constraints that one wishes to feed to the linear program. During the partitioning procedure, which proceeds by a sequence of moves between groups, the capacity of each group is allowed to exceed $MaxConstraints$ by a multiplicative factor $\alpha \geq 1$, which is a technique that is used to permit some amount of hill-climbing in finding the optimal solution by permitting the cost of a group to exceed $MaxConstraints$ temporarily.

Now that the partitioning problem has been explicitly defined, a multiple-way synchronous sequential circuit partitioning algorithm based on the algorithm proposed by Sanchis [San89] is applied to generate the subcircuits.

For each group $\mathbf{G}_k$, and each boundary latch L, define the connection number, $\mathbf{\Phi}$, as:

$$\mathbf{\Phi}_{G_k}(L) = |\{\mathbf{B}|\mathbf{B} \in \mathbf{G}_k \text{ and } \mathbf{B} \in \mathbf{B_L}\}|. \tag{3.20}$$

Since each boundary latch connects exactly two blocks, $\mathbf{\Phi}_{G_k}(L) \in \{0,1,2\}$. In other words, if $\mathbf{B}_i \overset{L}{\leftrightarrow} \mathbf{B}_j$, then (a) if $\mathbf{B}_i \notin \mathbf{G}_k$ and $\mathbf{B}_j \notin \mathbf{G}_k, \mathbf{\Phi}_{G_k}(L) = 0$, (b) if $\mathbf{B}_i \notin \mathbf{G}_k$ and $\mathbf{B}_j \in \mathbf{G}_k$, or vice versa, $\mathbf{\Phi}_{G_k}(L) = 1$, and (c) if $\mathbf{B}_i \in \mathbf{G}_k$ and $\mathbf{B}_j \in \mathbf{G}_k, \mathbf{\Phi}_{G_k}(L) = 2$.

The *gain* associated with moving $\mathbf{B}$ from $\mathbf{G}_i$ to $\mathbf{G}_j$ is defined as

$$\begin{aligned} \mathbf{\Gamma}_{ij}(\mathbf{B}) &= \sum_l (\tau(L_l)|L_l \in L_{\mathbf{B}} \; and \; \mathbf{\Phi}_{G_j}(L_l) = 1) \\ &\quad - \sum_n (\tau(L_n)|L_n \in L_{\mathbf{B}} \; and \; \mathbf{\Phi}_{G_i}(L_n) = 2) \end{aligned} \tag{3.21}$$

The first term of (3.21) measures the benefit of moving $\mathbf{B}$ to $\mathbf{G}_j$, while the second measures the penalty of moving $\mathbf{B}$ out of $\mathbf{G}_i$.

Before beginning the partitioning procedure, the number of linear programming constraints, c_i, required for each block i is calculated using the symbolic constraints propagation algorithm. If $c_i \geq MaxConstraints$ for some block $\mathbf{B}_i$, then it is placed in a group alone, and will not be processed later. Let $TotalConstraints = \sum_j (c_j | c_j < MaxConstraints)$. Each remaining block is put into one of the N' groups,

$$N' = \left\lceil \frac{TotalConstraints}{MaxConstraints} \right\rceil, \tag{3.22}$$

such that for each group k, $C(\mathbf{G}_k) < MaxConstraints$. This is an integer knapsack problem, and many heuristic algorithms can be used to obtain an initial partition (see, for example, Chapter 2, [SDK83]). In some cases, it may be impossible to put all blocks into N groups without violating the restriction on $C(\mathbf{G}_k)$ above; if so, the number of groups may be larger than that given in (3.22).

Given the initial partition, the algorithm improves it by iteratively moving one block of partition from one group to another in a series of passes. A block is labeled *free* if it has not been moved during that pass. Each pass in turn consists of a series of iterations during each of which the free block with the largest gain

is moved. During each move, that the number of constraints in a block are not permitted to violate the limit given by (3.19). The gain number, $\mathbf{\Gamma}_{ij}(\mathbf{B})$, is updated constantly as blocks are moved from one group to another. At the end of each pass, the partitions generated during that pass are examined and the one with the maximum objective value, as given by (3.19), is chosen as the starting partition for the next pass. Passes are performed until no improvement of the objective value can be obtained.

After the partitioning procedure has been applied to the circuit, the optimization algorithm described earlier is applied to each group.

The experimental results published in [CSH95] showed considerable improvements when sizing and skew were used in conjunction, as against using sizing alone.

3.5.2 A Convex Programming Formulation

In [SSF95, SSF98], a method for combined sizing and skew optimization is presented for acyclic pipelines. As in the case of the linear programming approach just presented, this procedure utilizes the idea of cycle-borrowing using clock skew optimization to relax the stringency of the timing specification on the critical stages of the pipeline. Instead of the convex piecewise linear delay model in the previous approach, this method uses Elmore delays.

The combined problem of sizing and skew optimization is formulated as:

$$\begin{aligned} \text{minimize} \quad & Area \\ \text{subject to } x_i + \overline{d}(i,j) + T_{setup} &\leq x_j + P + \delta && (3.23) \\ x_i + \underline{d}(i,j) &\geq x_j + T_{hold} + \delta && (3.24) \\ -X_{max} &\leq x_i \leq X_{max} && (3.25) \end{aligned}$$

where $\overline{d}(i,j)$ and $\underline{d}(i,j)$ are functions of the gate sizes in the circuit, P is the specified clock period, and X_{max} is the maximum allowable skew magnitude. The $Area$ objective function is approximated as the sum of all transistor sizes.

The area of the clocking network is not explicitly included in the formulation for two reasons. Firstly, it is difficult to derive a relation between the skews and the clocking network area. Secondly, the complexity of routing the clock network is such that it is not possible to predict whether a nonzero skew clock tree will necessarily use up more routing resources than a zero-skew tree.

A simple model such as the sum of the skews would not work; consider, for example, the case where FF_1 and FF_2 are two flip-flops that are physically adjacent in the layout of the circuit and have the same skew. Then one may take advantage of their physical proximity by using the same set of buffers to generate the skew for both, as shown in Figure 3.13(a), rather than wastefully adding extra buffers as shown in Figure 3.13(b).

Even so, it is essential to ensure that the expense of clock routing does not run amuck, and therefore, the constraint (3.25) is introduced; the idea is that since the clock tree is likely to consume routing resources when the skew

Figure 3.13. Clock network routing considerations.

magnitudes are large, the cost of the clock tree is controlled by limiting the maximum skew magnitude.

Properties of the Optimization Problem. Under the Elmore delay model, it can be shown [FD85] that the gate delays are posynomial functions[3] [Eck80] of the gate sizes. A posynomial function in $\mathbf{y}$ can be transformed into a convex function in $\mathbf{z}$ using the mapping $y_i = e^{z_i}$. Based on this fact, it was pointed out in [Fis90] that the above optimization problem is a *signomial* programming problem [Eck80] and does not, in general, correspond to a convex program. This makes it difficult to arrive at a good solution to the problem.

It can be seen that each long path constraint is of the form

$$\overline{d}(i,j) + x_i - x_j \leq K, \tag{3.26}$$

where $\overline{d}(i,j)$ is the maximum delay between flip-flops i and j (which is some posynomial function of the transistor sizes), x_i and x_j are clock delays to the source and destination flip-flops, and K is a constant. The left-hand side of this inequality is not a posynomial because of the negative coefficient of x_j. If the logarithmic substitution, $x = e^z$, were performed for each clock skew variable x and transistor size variable w in this inequality, the result would not be a convex constraint.

However, performing the substitution $w = e^z$ for each transistor width w appearing in $\overline{d}(i,j)$, while leaving x_i and x_j alone, results is a convex constraint: the left-hand side of the inequality is the sum of $\overline{d}(i,j)$, which is convex in the new z variables, and $x_i - x_j$, which is linear (hence convex), in the variables x_i and x_j[4].

Therefore, it can be concluded that for general sequential circuits, the problem of adjusting both clock skews and transistor sizes to meet long path constraints, while minimizing total gate area, corresponds to a convex optimization program.

Based on these observations, the following approach was proposed to solve this problem in two steps:

1. Neglect the short path constraints (which may be nonconvex [Fis90]) and solve the combined sizing and skew optimization under long path constraints only. This is equivalent to a convex optimization problem and is, therefore, a unimodal problem, i.e., any local minimum is also a global minimum.

2. Resolve any short path constraints that are violated at the end of Step 1 by changing the topology of the circuit and adding buffers in an optimal manner using variation of techniques such as [SBS93], described in Section 2.4.

Apart from easing the solution of the optimization problem, the above strategy has a useful side-effect. Since the presence of short path constraints results in a larger value of the optimal clock period over that achieved in the case where they are neglected, the method above will result in better clock periods than are achievable by keeping the circuit topology the same. It is also hoped that in practical circuits, the number of short path violations, and hence, the number of buffers to be added, will be not be too large.

The approach taken to solve Step 1 is an adaptation of the TILOS algorithm [FD85]. Although more exact methods such as those in [SRVK93] may be used to solve the convex programming problem exactly, these are not practical for use in this situation since the number of variables is too large for the exact algorithm to handle efficiently.

The optimization approach is divided into two stages that are repeated iteratively. In the first, violations of the clocking constraints must be detected. We present a modification of CPM for delay estimation, generalized to handle acyclic sequential circuits. Next, a critical path is defined and identified, and the size of the most sensitive gate on this path is bumped up by a small amount. The iterations continue until all long path-constraints are satisfied for the given clock period.

All of the theory developed so far applies to general sequential circuits. However, from this point, the discussion of this algorithm is specific to acyclic sequential circuits. This restriction is brought about by the fact that the fast timing analysis procedure described below can only handle acyclic circuits (though it may be extended to circuits with cycles).

Detection of Long-Path Constraint Violations. The procedure for timing analysis requires the determination of the optimal skews and the arrival times at all gate outputs. Since the constraint graph for an acyclic pipeline is acyclic, and since timing analysis involves the solution of the longest path problem on this constraint graph, the complexity can be reduced substantially by processing the vertices in topological order [CLR90]. Specifically, the timing analysis procedure is equivalent to applying CPM, described in Section 2.2.2, to the graph, with gates being represented in the normal way, and flip-flops being represented by blocks with "delay" values of $T_{setup} - P - \delta$, where T_{setup}

is the setup time for the flip-flop and P is the applied clock period, and δ is the uncertainty in the clock skew[5].

We point out here that the reference point for the arrival time in each combinational block is the zero skew. For example, an arrival time of a at the output of a flip-flop implies that the clock signal at the flip-flop has been skewed by a. If this flip-flop fans out to a single inverter with a unit delay, then the arrival time at the output of this inverter would be $a + 1$. Since skews may be either positive or negative, it can be seen that arrival times may have either sign.

Figure 3.14. Modified CPM.

Consider Figure 3.14(a). We symbolically show the path from FF_i to FF_j as being represented by a single gate with delay $G_{maxdelay}$, which corresponds to the maximum combinational delay between these flip-flops. From (3.23) we have

$$x_i + G_{maxdelay} + T_{setup} \leq x_j + P + \delta \tag{3.27}$$

where x_i is the latest arrival time at the input of G, x_j is the latest arrival time at the input of gate H. Writing $x_i + G_{maxdelay}$ as d_G, the latest arrival time at the input of FF_j, we have the following constraint

$$x_j - d_G \geq T_{setup} - P - \delta \tag{3.28}$$

For a regular combinational gate, as shown in Figure 3.14(b), if d_{G1} and d_{G2} are the latest arrival times at the input of $G2$ and $G3$, then we have the difference constraint

$$d_{G2} - d_{G1} \geq delay(G2) \tag{3.29}$$

From the two difference constraints given above it can be seen that FF_j behaves like a gate with a delay of $T_{setup} - P - \delta$.

It is noteworthy that CPM is valid if the circuit is acyclic; if not, techniques such as those used in [LW83] will have to be applied for analysis. Alternatively, for optimization purposes, a cyclic graph can be forced to be acyclic using the techniques in [BSM94], with additional constraints imposed at points where the cycles are broken.

The method used here for modeling the delay of a gate has been used extensively and successfully in several gate sizing algorithms [SK93]. The delay of a gate can be characterized by its size and the capacitive load that it drives. All transistors of the same type (n or p) within the gate are assumed to have the same size in this algorithm, and the gate delay is calculated by replacing it by an equivalent inverter with parameters W_N and W_P corresponding to the equivalent n and p transistor sizes, respectively.

Each gate is replaced by an RC tree with a driving resistance R_{on} and a capacitive load C_{out}. The Elmore delay of the charging(discharging) network is taken to be the rise(fall) time of the gate. As has been done in previous work on sizing, the on-resistance, R_{on}, is inversely proportional to W_P (W_N) during the charging (discharging) transition. The capacitive load, C_{out}, is composed of two parts: C_{fanout}, the fanout capacitance, and C_{intr}, its intrinsic capacitance. The value of C_{fanout} of gate i is equal to its fanout gate capacitances and is given by

$$C_{fanout}(i) = \beta \cdot \sum_{i=1^f} (W_{i,N} + W_{i,P}) \tag{3.30}$$

where $W_{1,N}, W_{1,P}, \cdots W_{f,N}, W_{f,P}$ are the sizes of the n and p transistors in the equivalent inverters corresponding to the f gates that logic gate i fans out to, and β is a constant of proportionality.

The value of C_{intr} of gate i is proportional to its own size and is given by

$$C_{intr} = \alpha \cdot W_i \tag{3.31}$$

where W_i is the size of gate i, given by the sum of the n and p-transistor channel widths, and α is a constant of proportionality.

We can now write the delay of a gate by the following equation [FD85, ME87, SSFD88, SRVK93]:

$$Delay = R_{on} \cdot (C_{fanout} + C_{intr}) \tag{3.32}$$

Unifying Transistor Sizing and Clock Skew Optimization. Initially, all gate sizes are set to the minimum value. The sensitivity-based algorithm performs the following tasks in each iteration:

1. A timing analysis is carried out on the combinational circuit using CPM to identify paths that fail to satisfy the delay constraint. A violation is said to occur if the rise/fall delay at a primary output is ≥ 0, or if the skew at

a flip-flop exceeds the specified maximum skew. In each case, the delay of the path from the primary inputs to the flip-flop or primary output must be reduced. Starting from the flip-flop or primary output with the maximum violation, a critical path is traced back from the primary output with the largest violation, to a primary input.

2. The sensitivity of each gate along the critical path is computed, and the most sensitive gate is identified, so that its sizes may be bumped up. Depending on whether the gate to be sized is undergoing a rise (fall) transition, the p-channel (n-channel) width is bumped up.

Therefore, in each iteration, it is expected that the delay of the critical path will be reduced by a small amount. However, the change in the size of the most sensitive transistor for the current critical path is liable to increase the delays of other paths in the circuit, and hence, as in [FD85], it is important to use a small bumpsize.

The algorithm continues until all timing requirements are met, or until the sensitivity of the most sensitive gate in the most critical path is ≥ 0. At this stage, any further increase in channel widths results only in the increase in delay, and the specified clock period is unachievable. The process of calculating the optimal skews falls out as a natural consequence of this procedure: if the timing requirements are met, then the arrival time at the output of a flip-flop, as calculated by CPM, is the optimal skew to be applied to that flip-flop.

The experimental results published in [SSF98] showed considerable improvements when sizing and skew were used in conjunction, as against using sizing alone.

3.5.3 A Procedure that Incorporates the Cost of the Clock Tree

Both the linear programming and convex programming technique described here control the cost of the clocking network only by placing bounds on the clock skews. An alternative procedure in [XD97] optimizes the cost of the clock tree directly, but uses more approximate models for gate sizing. The optimization problem is set up as

$$\text{minimize } C(T, \mathbf{x}) = \lambda L(T) + \gamma \cdot \Phi(\mathbf{x}) \tag{3.33}$$

where $C(T, X)$ is a cost function that is dependent on the routing topology T of the clock tree and on the vector of gate sizes $\mathbf{x}$. The functions $L(T)$ and $\Phi(\mathbf{x})$ represent the total wire length of the clock tree, and the power dissipation of the sized gates, respectively, and λ, γ are appropriately chosen weighting parameters. The cost function is optimized using simulated annealing [KGV83].

The algorithm first derives bounds on the relative skew at various sinks. If a pair of latches i and j is connected by combinational logic, then it is possible to derive upper and lower bounds on the value of the difference in skew $x_i - x_j$ using expression (3.9); these are referred to as the positive skew bound, PSB_{ij}, and the negative skew bound, NSB_{ij}. For nonadjacent sinks, no such bound

exists. Note that since the expression (3.9) is dependent on the delays, the bounds are derived over all circuit delays that correspond to allowable gate sizes.

For a complete understanding of this procedure, it is necessary to understand the DME procedure [KR95] fully (see Section 3.3.3). We will outline the essentials of the method to provide a general overall understanding.

Unlike the case of a Euclidean geometry, under a Manhattan geometry, there are several paths between two points that have the same distance. Therefore, when the tapping point (see Section 3.3) is chosen during a bottom-up zero-skew merge, it may lie at a distance $x \times l$ away from one node along *any* of these routes. It can be shown that the locus of possible tapping points forms a line segment, called a *merging segment.*

In the problem at hand, since the skew is not fixed, but bounded, this permits a variation in the location of the tapping point, and therefore, a variation in the location of the merging segment of the DME procedure. A move in simulated annealing corresponds to changing the position of a merging segment in such a way that the skew bounds are satisfied.

The gate sizing results are stored in the form of a look-up table. The minimum power of each combinational block for various sets of skews is determined and stored in this table. When the clock network is perturbed, the skews are mapped on to the closest set of skew values listed in the table, and the table is accessed to determine the corresponding power dissipation. The cost of this configuration is evaluated and fed into a simulated annealing loop that finds the optimal solution to the problem.

The tradeoff in this approach is that it provides accuracy in modeling the cost of the clock tree network, and an approximate and coarse modeling of the sizing procedure; the reverse was true of the previous two methods.

3.6 WAVE PIPELINING ISSUES

The treatment in [GLC94] provides an excellent illustration of the idea of wave pipelining, and is summarized here. A conventional combinational circuit and its associated timing are shown in Figure 3.15(a). The data flows out of the input registers on the onset of the clock, and propagates through the logic, reaching the output registers in time for the onset of the next clock.

In contrast, in a wave pipelined system, the propagation time for the data through the logic may be larger than the clock period. As an example, in Figure 3.15(b), the clocks at the input and output register are skewed. The data that leaves the input registers reaches the output registers after more than one clock period. During this time, the propagation of the second set of data from the input registers has commenced at time T_{clk}. Therefore, there is a period of time when two different signals corresponding to two different clock periods are flowing through the combinational logic, corresponding to two "waves." To ensure correct operation of the circuit, it is imperative to ensure that the two waves do not collide. The advantage of wave pipelining over conventional pipelining is that it may not require the use of as many registers,

Figure 3.15. Illustrating the idea of wave pipelining.

leading to potential savings in hardware and in the overhead of setup and hold times at each register. Moreover, the overhead of clock distribution is also reduced when the number of registers is reduced.

The use of skews usually implies that more than one wave moves through at least one of the combinational segments during a clock period. As an example, in Figure 3.2, when the first data set reaches the output of gate G2 after two time units (i.e., one clock period), the second data set is sent into the circuit at the input to gate G1. Since there is only one combinational path between IN and F1, there is no question of data collision, but in general, this may not be true.

Figure 3.16. An example illustrating logic signal separation constraints.

The work in [JC93] proposed an alteration of the clock skew formulation of Fishburn [Fis90] to incorporate constraints that enforce the requirement that no two waves collide during transmission; these constraints are referred to as logic signal separation constraints. Figure 3.16 shows two flip-flops, k and l, and a set of combinational logic gates, q, g and s. Node q is any predecessor gate of g such that flip-flop l lies in its fanin cone, and gate s is any successor of node g. Let the skews at the flip-flops k and l be x_k and x_l, respectively. The logic signal separation constraint under a clock period P is stated mathematically

as follows:

$$x_k + \text{delay}(k, s) \leq P + x_l + \text{delay}(l, q) \tag{3.34}$$

The left hand side of this equation represents the arrival time of the signal from the current wave at the output of the successor of g. This arrival implies that the value at g is no longer important for the current wave. The right hand side is the arrival time at the output of q during the next wave. Simply stated, the inequality says that it is essential for the inputs of g to be stable for as long as its output influences the current wave. In case g has multiple predecessors and successors, the relationship (3.34) must hold between each predecessor-successor pair.

Earlier in this chapter, several techniques for combined sizing and skew were described. None of them explicitly considers the logic signal separation constraints in the form in which they were published. However, it is relatively easy to incorporate them into the formulations. For the linear programming formulation, the output arrival time at each gate is available as a variable, and it is an easy matter to add constraints of the form (3.34) at $m_s \leq p_q + P$. For the convex programming formulation, this constraint can be checked during the application of CPM, and any violation can be flagged as a constraint violation to be reconciled by altering the sizing and/or skew solution.

3.7 DELIBERATE SKEWS FOR PEAK CURRENT REDUCTION

In synchronous circuits, all of the flip-flops switch at the latching edge of the clock. This simultaneous switching of flip-flops requires a large switching current, which in turn creates a significant voltage drop on the power distribution network, creating the need for multiple pins to distribute the power supply. In [VHBM96] deliberate clock skews are used to reduce the dynamic transient current drawn from the supply pins. This reduction in the transient current reduces the number of supply pins required by the chip, and therefore, its packaging cost.

The objective of introducing skews at the flip-flops is to ensure that all of them do not switch at the same time and hence, do not draw current at the same time; this serves to reduce the peak current. The problem is modeled as an integer linear program that minimizes the peak current. Each flip-flop may have one of n possible skew values. Correspondingly, for each flip-flop a set of n binary variables is used, of which exactly one variable is permitted to be 1, and the skew of the flip-flop corresponds to this variable. The clock period of the circuit imposes constraints on the possible skew value for each flip-flop, these timing constraints are incorporated in the linear program. Experimental results presented in [VHBM96] show a significant reduction in the ground bounce using this technique. However, the use of an integer linear programming formulation restricts the technique to system level design with few tens of modules. It is possible that the application of heuristics to this technique may permit it to be extended to larger designs, at the cost of sacrificing optimality.

A method for heuristic minimization of peak current by using clock skew is presented in [VBBD96, BVBD97]. Experimental results on circuits with up to

550 flip-flops indicate an average reduction of about 30% in the peak current. This method only minimizes the current peak directly caused by clock edges using a genetic algorithm. The current waveform is approximated as a triangle to allow efficient calculation of the total peak current in the circuit. The genetic algorithm based solution results in a required skew at each flip-flop. Since clock skew control for individual flip-flops is difficult, the flip-flops are clustered into a user specified number of clusters, and each flip-flop in a cluster has the same skew. A heuristic is used to attempt to cluster the flip-flops in such a way that there is a minimal loss in optimality due to this simplification.

3.8 CONCLUSION

The utility of deliberate skews for optimizing the performance of VLSI circuits has been demonstrated, and algorithms for performing skew scheduling and period minimization have been presented. Deliberate skews can also be used in conjunction with other timing optimization strategies and for minimizing the peak current in the supply network.

Until recently, there has been a great reluctance to alter the clock network and attempt a nonzero-skew solution. However, recently, an increasing number of designers have been willing to utilize skews for performance enhancement. Small amounts of skews can easily be provided by making minor changes in the sizes of the final buffers in the clock tree that feed the clock sinks, and altering skews in this manner is a relatively painless manner in which the optimization could be applied. For larger skew magnitudes, a more careful design of the clock network is essential; for high-performance applications, the gains outweigh the costs of this effort.

Notes

1. The reader is cautioned that the area-delay curves for Stg_1 and Stg_2 are, in reality, distinct from each other. To reduce the amount of clutter in the figure, in this example, we have shown all points on the same curve.

2. In this example, the change in the cost of the clock network has been neglected. It should be noted that it is easy to provide small skews by making minor changes in the sizes of the buffers driving the leaf-level nodes, and it possible to do this cheaply for small skew magnitudes.

3. A posynomial is a function g of a positive variable $\mathbf{w} \in \mathbf{R}^n$ that has the form $g(\mathbf{w}) = \sum_j \gamma_j \prod_{i=1}^n w_i^{\alpha_{ij}}$ where the exponents $\alpha_{ij} \in \mathbf{R}$ and the coefficients $\gamma_j > 0$. Roughly speaking, a posynomial is a function that is similar to a polynomial, except that (a) the coefficients γ_j must be positive. (b) an exponent α_{ij} could be any real number, and not necessarily a positive integer, unlike the case of polynomials.

4. This concept can be generalized to any optimization problem that can be divided into two separate classes $w_1, \cdots, w_n$, and $x_1, \cdots, x_m$, with each constraint is of the form

$$P(w_1, ..., w_n) + C(x_1, ..., x_m) \leq K, \tag{3.35}$$

where P is a posynomial function, C is a convex function, and K is a constant. Such a problem can be transformed into a convex program by performing the substitution $w_i = e^{z_i}$ while leaving the x_i variables alone.

5. Note that the word "delay," when used in reference to flip-flops, is applied in a loose sense here. The "delay" of a flip-flop as defined here is not the propagation delay of the gates within the flip-flop, but is a mathematical tool that can be used to apply CPM to a sequential circuit to check for delay violations in the presence of clock skews.

4 THE BASICS OF RETIMING

4.1 INTRODUCTION TO RETIMING

Retiming is a powerful sequential circuit optimization technique for improving the performance of sequential circuits. The concept of retiming is the notion of moving storage devices across memoryless computational elements to improve the performance without changing the input-output latency. Although retiming can operate on gate level netlists, or on higher-level abstractions such as data flow graphs, communication graphs, and processor schedules, our treatment will focus on circuit-level optimizations.

At the circuit level, the storage devices, or registers, may either be edge-triggered flip-flops or level-sensitive latches, commonly referred to as FF's or latches, respectively. The computational nodes at the circuit level are typically combinational gates. Retiming moves registers across gates without changing the number of registers in any cycle or on any path from the primary inputs to the primary outputs, thereby preserving the input-output latency of the circuit. Since retiming does not directly affect the combinational part of the circuit, the circuit behavior remains unchanged. However, since retiming can change the boundaries of combinational logic, it has the potential to affect the results of combinational synthesis techniques that are applied to the sequential circuit.

This chapter begins with an overview of the retiming procedure. Next, the assumptions and models common to many retiming algorithms are presented, followed by a description of the Leiserson-Saxe theory for retiming

N. Maheshwari et al., *Timing Analysis and Optimization of Sequential Circuits*
© Kluwer Academic Publishers 1999

edge-triggered circuits is presented. Finally, we discuss fast and efficient methods for retiming large edge-triggered and level-clocked circuits to minimize their clock period.

4.1.1 Types of Retiming

Retiming may be performed to improve the circuit behavior with respect to several possible objective functions, some of which are outlined below.

Figure 4.1. Effect of retiming on clock period

Clock Period Algorithmically, the simplest objective function used in retiming is the minimization of the clock period. Since the clock period in a circuit with edge-triggered FF's is given by the maximum combinational stage delay, the FF's may be relocated to reduce the clock period. For the circuit shown in Figure 4.1(a), with unit delay gates and edge-triggered FF's, the clock period is 3 time units. If we relocate register F1 from the output of gate G3 to its input, we obtain the circuit in Figure 4.1(b), with a clock period of 2 units. Notice that the input-output behavior is left unchanged by retiming since the output is produced after two clock cycles in both the original and the retimed circuit. Thus, relocating registers can reduce the clock period of a circuit, and retiming can be used to relocate registers with the objective of minimizing the clock period. A retiming that minimizes the clock period of a circuit is termed a *minperiod retiming*. Retiming a circuit to achieve a specified target clock period is a special case of minperiod retiming, and is often called specified-period retiming.

Area Since retiming does not affect the combinational part of the circuit, the area overhead of the combinational logic remains constant under retiming. The method may, however, affect the overall area of the circuit since it may alter the number of registers in the circuit.

Two retimed versions of the same circuit could have the same input-output behavior and clock period, but could use a different number of registers, as

Figure 4.2. Effect of retiming on number of registers

illustrated in Figure 4.2. The circuit in Figure 4.2(a) requires two registers, while that in Figure 4.2(b) requires only one register.

Therefore, one could apply retiming with the object of minimizing the number of registers in the circuit, while leaving the input-output latency unchanged. This may be done without any constraint on the clock period of the resulting circuit, or subject to a target clock period. The former is called unconstrained minarea retiming while the latter is referred to as constrained minarea retiming or simply *minarea retiming*. In practice, minarea retiming is a more useful form of the retiming transformation than minperiod retiming.

Power The power dissipated in a circuit depends on the product of the switching activity and the load capacitance at the output of a gate, summed over all gates. Since registers can filter out glitches, altering their locations can affect the switching activity at gate outputs; moreover, it can also alter the load capacitance seen by the gates. Thus, retiming can change the power dissipation of a circuit, and an appropriately chosen retiming may be used to optimize the power by placing registers at nodes with high switching activity values and high capacitive loads.

Testability The relocation of registers can change the state encoding in sequential circuits, thus affecting the test generation time and the number of redundant faults. The repositioning of registers also affects the length of the scan chains required for partial or full scan designs. Retiming can, therefore, be used to improve the testability of sequential circuits.

Quality of Logic Optimization Most logic optimization techniques operate on combinational logic blocks separated by register boundaries. Hence, changing these register boundaries by retiming the registers affects the qualityof results obtained by logic optimization.

A brief survey of publications describing these research activities is presented in Section 4.2, and in the next few chapters, we will describe these methods in varying degrees of detail.

4.1.2 Types of Circuits

Algorithms for retiming a circuit must address the specific requirements of a circuit, and the clocking discipline used. Four major classes of circuits are described below.

Edge-triggered circuits In an edge-triggered circuit, the clock period is given simply by the largest combinational delay. The first publications on retiming concentrated on handling this class of circuits.

Level-clocked circuits Level-sensitive latches are, by definition, transparent during the period when the clock signal is active; this transparent nature gives level-clocked circuits the potential to operate at a faster clock period[1]. These circuits require less area than their edge-triggered counterparts, not only because the cycle stealing reduces the amount of transistor sizing required to meet the timing goals, but also because individual latches require a smaller amount of area than edge-triggered FF's. Unfortunately, the analysis of level-clocked circuits is more complicated than edge-triggered circuits, and hence, algorithms for finding an optimal retiming can be computationally expensive.

Control Logic Control logic involves an implementation of finite state machines (FSM's), and hence, the registers in control circuitry are associated with the FSM states. Retiming alters the locations of these registers, and consequently, the state encoding of the FSM. Thus, issues regarding safe replaceability become important. In circuits that have a meaningful initial state, it is important to find a retimed circuit with an equivalent initial state; many retimings of a circuit that are otherwise valid may not have equivalent initial states. To see this, consider the circuit in Figure 4.3(a). If we wish to move FF's A and B across gate G1 (to FF C in Figure 4.3(b)), we must find an initial value of FF C that is equivalent to their initial values. If FF's A and B have conflicting values, no such equivalent initial value exists at FF C. Thus, additional constraints must be imposed to ensure the presence of an equivalent initial state while retiming control logic.

Figure 4.3. Equivalent initial states in reverse retiming.

FPGA's Field Programmable Gate Arrays (FPGA's) present requirements that are different from those in conventional combinational logic. For example, in LUT-based FPGA's the amount of logic is dependent on the number of inputs, and not on the complexity of the logic. Further, since FPGA's have limited resources with memory elements at fixed locations, extra constraints are placed on the movement of memory elements during retiming. The issue of combined synthesis with retiming shows the maximal gains in FPGA optimization.

4.2 A BROAD OVERVIEW OF RESEARCH ON RETIMING

Since retiming was introduced by Leiserson and Saxe [LRS83, LS91], a significant amount of research has been carried out, both in academia [Sax85, Pap90, Cha93, Pap93, Ran93, She93, Deo94, Eve94, Lal96, Sin96, Has97, Ran97] and in industry (for example, at IBM, Philips and Synopsys). In this section, we present a brief literature survey of retiming-related research. For a good introduction and overview of retiming, the reader is also referred to [She97] and to Section 9.3.1 of [De 94]. In the remainder of this book, more detailed descriptions of these issues will be presented.

Edge-triggered Circuits Leiserson and Saxe introduced algorithms for minperiod and minarea retiming of edge-triggered circuits [LRS83, LS91]. The basis for this work is a graph-based representation of the circuit, and a procedure used to map the retiming problem on to polynomial time graph algorithms. This approach will be referred to in this book as the "Leiserson-Saxe approach." circuit is represented by a graph and polynomial time algorithms are presented. The major contribution of this work is in formulating the theory of retiming, rather than in presenting experimental implementations. The minperiod retiming problem is solved by performing a binary search for the best clock period, checking the feasibility of the clock period using a Bellman-Ford-like relaxation algorithm. The minarea problem is formulated as a linear program (LP), which is shown to be the dual of a mincost network flow problem.

Shenoy and Rudell presented an efficient and clever implementation of the Leiserson-Saxe algorithms in [SR94]. Their main contributions include reducing the memory requirements from $O(|G|^2)$ to $O(|G|)$, where $|G|$ is the number of gates in the circuit, and the use of back-pointers to speed up the feasibility check during the binary search for minimum clock period. At about the same time as [SR94], a technique for reducing the number of constraints in the minarea LP was presented in [vVA+94].

The ASTRA algorithm [DS95, SD96] exploited the retiming-skew equivalence for fast minperiod retiming. ASTRA first finds a minimum period achievable by skew optimization, and then translates these skews into retiming. Circuits with 20,000 gates were shown to be retimed in under two minutes.

The Minaret algorithm [MS97a, MS98b] modifies ASTRA to efficiently obtain bounds on the retiming variables, and uses them to significantly reduce the size of the minarea LP. This enables Minaret to perform minarea retiming on circuits with 50,000 gates in under 15 minutes.

Level-Clocked Circuits A signal that flows through a level-sensitive latch during its transparent phase can initiate the computation of the next combinational stage before the beginning of the next clock cycle; this phenomenon is called cycle stealing. Due to cycle stealing, level-clocked circuits have the potential to operate faster, and require less area. Algorithms to retime single phase level-clocked circuits are presented in [SBS91], and techniques based on the Leiserson-Saxe model for retiming multi-phase level-clocked circuits were presented in [ILP92, LE92, LE94]. TIM [PR93b] is a comprehensive timing analysis and optimization CAD tool for level-clocked circuits that is available in public domain, and has been used to empirically compare the performance of edge-triggered and level-sensitive circuits in [PR93a]. The work in [IP96] presented a polynomial-time algorithm for pipelining two-phase, level-clocked circuits under a bounded delay model.

The ASTRA and Minaret algorithms for edge-triggered circuits have been extended to level-clocked circuits in [MS96, MS98a] respectively. This enables minperiod and minarea retiming of level-clocked circuits with tens of thousands of gates in very reasonable time.

Retiming with Equivalent Initial States Traditional retiming algorithms do not pay any regard to initial states or power-on states of circuits, and are not very useful for control logic. Control logic usually has meaningful initial states, and any useful retiming must also find a new initial state for the transformed circuit that is equivalent to the initial state of the original circuit.

A method for minperiod retiming with equivalent initial states was presented in [TB93], using only the so-called forward retiming moves. In some cases, this approach may require modifications in the combinational part of the circuit. An efficient technique for performing these modifications is presented in [SMB96]. An alternative approach, termed reversed retiming [SSE95, ESS96], uses a minimum number of reverse (backward) retiming moves, and precludes the need for any modifications to the combinational part of the circuit. An approach to minimum area retiming that maintains initial states is presented in [MS97b].

Low Power Retiming can affect the power dissipation of a circuit since it alters the amount of switching in a circuit, as also the fanout capacitance driven by various elements in the circuit. A mechanism for reducing power by retiming was presented in [MDG93]; it places FF's on interconnects with high switching activity. The approach in [LP96] presented algorithms to reduce power by retiming only one phase in a two-phase circuit. The advantage of

retiming only one phase is that it preserves the testability of the circuit. A similar approach is taken in [SSWN97] to reduce power in DSP designs.

Testing and Verification Retiming can be used both to improve the testability of a circuit, and as an aid to automatic test generation. In the former case, circuit is actually implemented in the retimed form, while in the latter case the retimed circuit is merely used by the test generator to generate test vectors, but the original circuit is implemented. Various researchers have worked on characterizing the effects of FF relocation on the redundancy of faults [DC94, YKK95a, YKK95b, DB96], and the corresponding effects on ATPG run time [MEMR96]. In [EMRM95, EMRM97], it was shown that retiming preserves testability with respect to a single stuck-at-fault test set by adding a prefix sequence of a pre-determined number of arbitrary input vectors. Retiming has been used to improve testability [DC94] by attempting to convert sequential redundancies to combinational redundancies. Retiming may also be employed for reducing test lengths in scan-based designs [KT94, HKK95, HKK96], and for improving pseudo-exhaustive built-in self test [KTB93, LKA95a, LKA95b, LKA95c].

The work in [SPRB95] shows that while an accurate logic simulation may distinguish a retimed circuit from the original circuit, a conservative three-valued simulator cannot do so. Techniques for verification of retimed circuits are presented in [HCC96, HCC97, RSSB98].

Enhancements to Retiming The traditional retiming approaches assume the gate delays to be fixed, and all FF delays to be equal. Since these are only approximations, much effort has been spent in incorporating improved delay models into retiming. Delay models that incorporate clock skews, register delays, etc., are presented in [SFM93, SF94, LP95a, LP95b, SFM97], while [KO95] presents retiming under a variable delay model.

Retiming has also been extended to handle multiple clocks and registers with enable ports in [Mar96, LVW97], and to handle gated clocks and precharged structures [Ish93]. Various efforts have been made to combine retiming with other logic synthesis techniques, e.g., [MSBS90, MSBS91, PR91, Lin93, LKW93, BC96, Pan97, AB98]. Architectural retiming [Has97] modifies the combinational part of a circuit to increase the number of registers on a critical cycle or path without increasing the perceived latency.

Other Applications Retiming has been used in numerous other applications: during the technology mapping step in FPGA synthesis [TSS92, WR93, CW94, PL95, CW96], to improve circuit partitioning [LSC+93], for scheduling in high level synthesis [CS93a, WBL+94], in multiprocessor scheduling [CS92], for system level throughput optimization include [MH91, DPP95], etc. Retiming has also been used extensively in DSP applications [Dun92, PR92, Geb93, PS96, PSB96].

Figure 4.4. A correlator circuit.

Figure 4.5. Graph model of the correlator example.

4.3 MODELING AND ASSUMPTIONS FOR RETIMING

Any detailed presentation of material on retiming necessarily requires a section explaining the models and preliminaries that are used by most algorithms. In this section, we will present a graph-theoretic foundation that will be utilized later in our exposition on retiming.

In the ensuing discussion, we will consider the example of a digital correlator circuit from [LS91], shown in Figure 4.4, to motivate the procedure of retiming by demonstrating the performance improvements on this example. The design consists of two types of functional elements: adders (denoted by a '+' symbol) and comparators (denoted by a '=' symbol). The boxes between the comparators are registers that act to shift the data. The delay of an adder is 7 units and that of a comparator is 3 units. Although this design is functionally correct, it has poor timing characteristics.

For the purposes of applying a retiming algorithm, it is useful to abstract the problem from the circuit level into a directed graph representation, $G(V, E, d, w)$, where V is the set of vertices and E the set of edges. Each vertex $v \in V$ models a functional element with a fixed propagation delay, $d(v)$, that remains unaltered during the retiming process. A special vertex called the *host* is introduced in the graph, with outgoing edges connected to all primary inputs of the circuit, and incoming edges from all primary outputs to the host vertex. The host ver-

tex is assigned a propagation delay of zero, i.e., $d(host) = 0$. The directed edges model the interconnections between functional elements. Each edge $e_{uv} \in E$ connects an output of functional element u to an input of a functional element v, with a weight of $w(e_{uv})$ that corresponds to the number of registers between u and v. The graph corresponding to the correlator circuit in Figure 4.4 is shown in Figure 4.5.

As an informal definition, a retiming move in a circuit is caused by moving all of the memory elements at the input to a combinational block to all of its outputs, or vice versa. A sequence of such movements is referred to as a retiming of the circuit.

Equivalently, working on the graph representation of the circuit, a retiming is an integer-valued labeling of the vertices, $r : V \rightarrow \mathbf{Z}$, where $\mathbf{Z}$ is the set of integers. The weight of an edge e_{uv} after retiming, denoted by $w_r(e_{uv})$, is given by

$$w_r(e_{uv}) = r(v) + w(e_{uv}) - r(u) \tag{4.1}$$

The retiming label, $r(v)$, for a vertex v represents the number of register stages that have been moved from its outputs to its inputs. Retiming can, therefore, also be viewed as an assignment of a lag value, $r(v)$, to every vertex v in the circuit. A path originating at vertex u and terminating at vertex v is represented as $u \rightsquigarrow v$. For a path $p : u \rightsquigarrow v$, one may define its weight, $w(p)$, as the sum of the weights on the edges on p, and its delay, $d(p)$, as the sum of the weights of the vertices on p. A path with $w(p) = 0$ corresponds to a purely combinational path with no registers on it. Therefore, the clock period, P, can be calculated as

$$P = \max_{\forall\, p | w(p)=0} \{d(p)\} \tag{4.2}$$

The Leiserson-Saxe method presents a systematic technique for computation of the retiming labels, where the problem is formulated as a mixed integer linear program (MILP). An important concept used in the approach is the notion of the W and D matrices. The matrices are defined for all pairs of vertices (u, v) such that there exists a path, $p : u \rightsquigarrow v$, that does not include the host vertex. The formal definition of the matrices is as follows:

$$W(u, v) = \min_{\forall\, p:u \rightsquigarrow v} \{w(p)\} \tag{4.3}$$

$$D(u, v) = \max_{\forall\, p:u \rightsquigarrow v \text{ and } w(p)=W(u,v)} \{d(p)\} \tag{4.4}$$

In plain English, $W(u, v)$ denotes the minimum latency, in clock cycles, for the data flowing from u to v, and $D(u, v)$ gives the maximum delay from u to v over all paths with that minimum latency. The (W, D) pairs are used to generate constraints, and the essential reason why $D(u, v)$ is important is that if all memory elements between u and v are removed during retiming, then the delay between the two vertices on the path corresponding to $D(u, v)$ will be the maximum purely combinational delay between the two vertices. In the next section, we will use this information to ensure that if $D(u, v) > P$, the clock

period, then the value of $W(u,v)$ after retiming is at least 1. The important point here is that the delays on all but the minimum weight path are irrelevant.

The W and D matrices can easily be computed as follows. Each edge, e_{uv}, is reweighted with the ordered pair $(w(e_{uv}), -d(u))$, and the all-pairs shortest paths are computed. Any standard procedure for finding all-pairs shortest paths, e.g., the Floyd-Warshall method, or Johnson's algorithm [CLR90], may be used, with the following change that allows an ordered pair of weights to be correctly interpreted. During the relaxation process, comparisons between ordered pairs are performed in lexicographic order, i.e., amongst two pairs, $p_1 = (a_1, b_1)$ and $p_2 = (a_2, b_2)$, we say that $p_1 < p_2$ if $(a_1 < a_2)$ or $(a_1 = a_2$ and $b_1 < b_2)$. At the end of this computation, if the shortest-path weight between two vertices u and v is (x, y), then $W(u,v) = x$ and $D(u,v) = d(v) - y$.[2]

4.4 MINPERIOD OPTIMIZATION OF EDGE-TRIGGERED CIRCUITS

The algorithms presented in this section are all of polynomial time complexity. We will first present algorithms that are rooted in the techniques presented by Leiserson and Saxe [LS91] for retiming edge-triggered circuits, and follow these by discussing how they may be applied efficiently in practice. Finally, we will describe a method that applies the retiming-skew relationship to find an efficient minperiod retiming solution.

In each of these methods, the minimum period obtainable under retiming is calculated by performing a binary search over all possible clock periods. At each step in the binary search, an attempt is made to retime the circuit for the current value of the clock period. The smallest period for which retiming succeeds is returned as the best clock period.

4.4.1 Leiserson-Saxe-based Algorithms for Minperiod Retiming

Any retiming solution must satisfy the following two inequalities

$$\begin{aligned} r(u) - r(v) &\leq w(e_{uv}) & \forall e_{uv} \in E \\ r(u) - r(v) &\leq W(u,v) - 1 & \forall D(u,v) > P \end{aligned} \tag{4.5}$$

The first constraint ensures that the weight of each edge, e_{uv} (i.e., the number of registers between the output of gate u and the input of gate v), after retiming is nonnegative, i.e., $w_r(e_{uv}) \geq 0$. We will refer to these constraints as *circuit constraints*. The second constraint ensures that after retiming, each path whose delay is larger than the clock period has at least one register on it. These constraints, being dependent on the clock period, are often referred to as *period constraints*.

For a fixed clock period, the set of constraints (4.5) correspond to a system of difference constraints. This system can be represented by a constraint graph that may be solved by applying the Bellman-Ford method [CLR90]. If the system of constraints is infeasible, this will be indicated by the Bellman-Ford procedure. The smallest clock period for which the constraints are feasible is the solution to the minperiod problem.

This notion leads to the first algorithm, OPT1 [LS91]. As mentioned earlier, in the outer loop, a binary search is performed on the value of the clock period, and for each clock period, the Bellman-Ford algorithm is applied to check for feasibility of the constraint set. The binary search for the minimum value of P is justified by the fact that the feasible values of P form a continuous interval. This is due to the fact that the system (4.5) of inequalities is linear in P, making the projection of the solution space is convex in P, i.e., it is an interval. The OPT1 algorithm requires the calculation of the W and D matrices, and is expensive in terms of computation time ($O(V^3 \log V)$) and memory space ($O(V^2)$).

An alternative $O(|V||E|\log|V|)$-time algorithm from [LS91] is a more practical option for obtaining a retiming. The main routine FEAS is invoked during the binary search to check whether a specified clock period, P, is feasible or not. It proceeds by calling a subroutine, CP, that calculates the clock period of the circuit by systematically identifying the purely combinational path that has the largest delay. For any gate (or, in general, functional element) whose delay from an FF exceeds P, the retiming label, r, is incremented.

```
Algorithm FEAS
Given a synchronous circuit G = <V,E,d,w>, and a desired clock
period P, return a retiming r of G such that the clock period
of the retimed circuit Φ(G_r) ≤ P.
{
  1.  For each vertex v ∈ V, set r(v) ← 0.
  2.  Repeat the following |V| − 1 times
      2.1 Compute the graph G_r with the existing values of r.
      2.2 Run Algorithm CP on the graph G_r to determine Δ(v)
          for each vertex v ∈ V.
      2.3 For each v such that Δ(v) > P, set r(v) ← r(v) + 1.
  3.  Run Algorithm CP on the circuit G_r.  If Φ(G_r) > P,
      then no feasible retiming exists.  Otherwise, r is the
      desired retiming.
}

Algorithm CP
This algorithm computes the clock period Φ(G) for a synchronous
circuit G = <V,E,d,w>.
{
  1.  Let G_0 be the subgraph of G that contains precisely those
      edges e such that the register count w(e) = 0.
  2.  Perform a topological sort on G_0, totally ordering
      its vertices so that if there is an edge from vertex u
      to vertex v in G_0, then u precedes v in the total order.
  3.  Go through the vertices in the order defined by the
      topological sort.
      On visiting each vertex v, compute Δ(v) as follows:
```

```
      a.  If there is no incoming edge to v, Δ(v) ← d(v).
      b.  Otherwise, Δ(v) ← d(v) + max{Δ(u) : u -e-> v and e ∈ G0}.
  4.  The clock period Φ(G) is max_{v∈V} Δ(v).
}
```

The advantage of this method is that it does not require the explicit calculation of the W and D matrices, thereby reducing the memory overhead. Step 2 of FEAS can be shown to be equivalent to applying the Bellman-Ford algorithm on the graph G.

Figure 4.6. Retimed graph of the correlator.

The application of either of these retiming procedures may be used to assign the r labels to each vertex in the graph corresponding to the correlator circuit shown in Figure 4.4. For the unretimed circuit, the clock period is 24 units, which corresponds to the sum of the propagation delays along the longest register-free path, $v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7$. The graph obtained on applying min-period retiming to Figure 4.5 is shown in Figure 4.6. The r labels describe the manner in which flip-flops are added or removed from the original graph. For instance, the edge (v_1, v_7) has one more register since $r(v_7) - r(v_1) = 1$ (see relation (4.1)). Since retiming preserves the input-output latency of the circuit, the original and the retimed circuits corresponding to graphs in Figure 4.5 and Figure 4.6, respectively, must have the same input/output behavior. However, the computations performed by v_1 in Figure 4.5 lead, by one clock tick, the same computations performed by v_1 in Figure 4.6. This is the physical meaning of the fact that the vertex v_1 is assigned a lead of one clock tick or equivalently, a lag $r(v_1) = -1$. The clock period of the retimed circuit is 13 time units, corresponding to the sum of the propagation delays along the longest path, $v_2 \rightarrow v_3 \rightarrow v_5$.

Implementational Notes. An efficient implementation of the minperiod retiming algorithm was presented in [SR94], using a predecessor heuristic. The procedure is similar to the method of [Szy92], presented in Section 3.4.5, and its adaptation to this problem is presented here. The predecessor heuristic maintains a predecessor vertex pointer, denoted by `pred()`, for each vertex, which is initialized to the empty set. When $\Delta(v)$ is computed for each vertex

v, a reference to a vertex u is stored, where u satisfies the property that there exists a zero-weight path $p : u \rightsquigarrow v$ and $\Delta(v) = d(p)$. If $\Delta(v) > P$, then the procedure sets `pred(v) = u`.

Consider a cycle in the predecessor subgraph with vertices $u_1, \cdots, u_k = u_0$, i.e., `pred(`u_i`)` $= u_{i-1}$, $i = 1, \cdots, k$.[3] Let $p_{i-1,i}$ denote the path $u_{i-1} \rightsquigarrow u_i$ used in the computation of $\Delta(u_i)$. During each iteration in Step 2 of FEAS, the retiming labels increase by at most 1, and before the update, the weight on each path $w(p_{i-1,i}) = 0$. After the update,

$$w_r(p_{i-1,i}) = r(u_i) - r(u_{i-1}) + w(p_{i-1,i}) = r(u_i) - r(u_{i-1}) \leq 1$$

Therefore, $\sum_{i=1}^{k} w_r(p_{i-1,i}) \leq k$. Since $P < d(p_{i-1,i})$ for each i, we have

$$\begin{aligned} P \cdot k &< \sum_{i=1}^{k} d(p_{i-1,i}) \\ \Rightarrow P \cdot \sum_{i=1}^{k} w_r(p_{i-1,i}) &< \sum_{i=1}^{k} d(p_{i-1,i}) \end{aligned} \tag{4.6}$$

As long as the clock period is chosen so that this condition is true, a feasible retiming will not be possible. Therefore, it is essential to choose a clock period for which this condition is violated; this provides an updated lower bound for the binary search, given by

$$P \geq \frac{\sum_{i=1}^{k} d(p_{i-1,i})}{\sum_{i=1}^{k} w_r(p_{i-1,i})} \tag{4.7}$$

The use of this bound to update the lower bound of the binary search was shown in [SR94] to provide tremendous improvements in the execution time of the algorithm.

4.4.2 The ASTRA Approach

The Relationship Between Clock Skew and Retiming. The basis of the ASTRA approach [DS95, SD96] is the relationship between clock skew and retiming, as illustrated by the following example on a circuit with edge-triggered flip-flops. Let us first consider the use of intentional clock skews for improving the circuit performance. Consider the circuit in Figure 4.7, which has unit delay inverters and negative edge-triggered flip-flops. The delays of the first and the second combinational block are 3 units and 1, respectively. Therefore, the fastest allowable clock has a period of 3 units. If an attempt is made to operate the circuit with a clock period of 2 units, then we have a long path violation at FF1, because the required data arrival time at FF1 is 2 units (assuming zero setup time), but the data arrives at 3 units, as shown in Figure 4.7. However, if a skew of +1 unit is applied to the clock line to FF1, as shown in Figure 4.8, then the circuit can be operated without any long path violation, even for a clock period of two units[4].

Figure 4.7. An example of a circuit with long path violation.

Figure 4.8. Using clock skew to correct the long path violation.

Figure 4.9. Retiming for clock period reduction.

It is easy to see that for the given circuit, the period can also be minimized to two units by retiming, if FF1 is moved to the left across the inverter G3, as shown in Figure 4.9.

In each case, one unit of time is borrowed by the first combinational block from the second; the manner in which cycle-borrowing occurs may either be via the vehicle of clock skew or via retiming. As a prelude to a more formal presentation of the relationship between clock skew and retiming, consider a flip-flop j in a circuit, as shown in Figure 4.10(a). As explained in Section 3.4.1, for every combinational path from a flip-flop i to j with delay $d(i,j)$, the following constraints must hold to prevent zero-clocking and double-clocking, respectively:

$$\begin{aligned} x_i + \overline{d}(i,j) + T_{setup} &\leq x_j + P \\ x_i + \underline{d}(i,j) &\geq x_j + T_{hold}, \end{aligned} \tag{4.8}$$

where x_i and x_j are the skews at flip-flops i and j, respectively. Similar constraints can be written for every combinational path from flip-flop j to k with minimum delay $\underline{d}(j,k)$ and maximum delay $\overline{d}(j,k)$.

Figure 4.10. Relationship between retiming and skew.

Theorem 4.1 *For a circuit that operates at a clock period P, satisfying the double-clocking and zero-clocking delay constraints,*
(a) *retiming a flip-flop by moving it against the direction of signal propagation across a single-input, single-output gate of delay d_1 is equivalent to decreasing its skew by d_1.*
(b) *retiming a flip-flop by moving it in the direction of signal propagation across a single-input, single-output gate of delay d_2 is equivalent to increasing its skew by d_2.*

Proof:

(a) Consider the case where flip-flop j is moved to the left across the single-input gate G_l that has delay d_1. For a combinational path from an FF i to j, if x_j and x'_j are, respectively, the skews at flip-flop j before and after the relocation operation, then the following relationships must be satisfied:

$$\begin{aligned} x_i + [d(i,j) - d_1] + T_{setup} &\leq x'_j + P \\ x_i + [d(i,j) - d_1] &\geq x'_j + T_{hold} \end{aligned}$$

From relation (4.8), this implies that setting $x'_j = x_j - d_1$ achieves the same effect. Similarly, for a combinational path from j to a flip-flop k, it can be shown that setting $x'_j = x_j - d_1$ in the original circuit achieves the same effect. The proof to (b) is analogous and is omitted. □

A generalization of Theorem 4.1 to multi-input multi-output combinational blocks is as follows:

Theorem 4.2
(a) *Retiming transformations may be used to move flip-flops from all of the n inputs of any combinational block to all of its outputs. The equivalent skew of the relocated flip-flop at output j, considering long path constraints only, is given by*

$$x_j = \max_{1 \le i \le n} (x_i + \overline{d}(i,j)) \tag{4.9}$$

where the $x_i, 1 \le i \le n$, are the skews at the input flip-flops, x_j is the equivalent skew at output j, and $\overline{d}(i,j)$ is the worst-case delay of any path from i to j.
(b) *Similarly, flip-flops may be moved from all of the m outputs of any combinational block to all of its inputs, and the equivalent skew at input k, considering long path constraints only, is given by*

$$x_k = \min_{1 \le j \le m} (x_j - \overline{d}(k,j)) \tag{4.10}$$

where the $x_j, 1 \le j \le m$, are the skews at the input flip-flops, x_k is the equivalent skew at input k, and $\overline{d}(k,j)$ is the worst-case delay of any path from k to j.

The proof is omitted and proceeds along the lines of that for Theorem 4.1[5]. Theorem 4.2 implies that if one were to calculate the optimal clock skews, the corresponding retiming could be found by moving flip-flops with positive [negative] skews against [in] the direction of signal propagation until the skews at the flip-flops are nearly equal to zero. It must be noted that since gate delays take on discrete values, it is not possible to guarantee that the skew at a flip-flop can always be reduced to zero through retiming operations.

An alternative view of the same procedure is as follows. Retiming may be thought of as a sequence of movements of flip-flops across gates. Starting from the final retimed circuit, where all of the skews are zero, and the zero-clocking constraints are met, this sequence of movements may be performed in reverse order. This procedure can be used to move all flip-flops back to their initial locations, using Theorem 4.2 to keep track of the altered clock skews at each flip-flop. The optimal retiming is equivalent to applying the set of skews thus obtained to the flip-flops in the circuit.

Note that the optimal clock period provided by the clock skew optimization procedure must, by definition, be no greater than the clock period for the set of clock skews thus obtained. Any differences arise due to the fact that clock skew optimization is a continuous optimization, while retiming is a discrete optimization. This argument leads to the following corollary:

Corollary 2 : The clock period obtained by an optimal retiming can be achieved in ASTRA via clock skew optimization. The clock period provided by the clock skew optimization procedure is less than or equal to that provided by the method of retiming.

Minperiod Retiming by ASTRA. The relationship between skew and retiming motivates the following two-phase solution to the retiming problem:

Phase A: The clock skew optimization problem is solved to find the optimal value of the skew at each FF, with the objective of minimizing the clock period, or to satisfy a given (feasible) clock period, using the procedure described in Chapter 3.

Phase B: The skew solution is translated to retiming and some FF's are relocated across gates in an attempt to set the values of all skews to be as close to zero as possible. Flip-flops with positive skews are moved opposite to the direction of signal propagation (i.e., in the backward direction), and those with negative skews are moved in the direction of signal propagation (i.e., in the forward direction) to reduce the magnitude of their skews.

The skew values obtained in Phase A are reduced in Phase B by applying retiming transformations using Theorem 4.2. This procedure relocates the FF's with nonzero skews across logic gates, while maintaining the optimal clock period previously found.

A gate can be retimed in the forward [backward] direction if it has FF's on each of its inputs [outputs], by removing one FF from each input [output] and adding one FF to each of its outputs [inputs].

The algorithm maintains two sets, one for the gates that are to be forward-retimed, and one for the gates that are to be backward-retimed. The forward retiming set, F, is initialized to contain all gates that have at least one FF on each of their inputs. Similarly the backward retiming set, B, is initialized with gates that have at least one FF on each their outputs. These sets are then processed by taking a gate from the set and retiming it if the skew on the FF's can be reduced by this retiming. After each such move, the sets F and B are updated. The pseudo code for this is given below as the function `RETIME` below:

```
Algorithm RETIME
{
  F = {v|v ∈ V and w(e_uv) ≥ 1 ∀u ∈ FI(v)} /* initialize F */
  B = {v|v ∈ V and w(e_vu) ≥ 1 ∀u ∈ FO(v)} /* initialize B */
  while(∃ u ∈ F) do FORWARD_RETIME(u, F);
  while(∃ u ∈ B) do BACKWARD_RETIME(u, B);
}
```

The functions `FORWARD_RETIME(gate,set)` and `BACKWARD_RETIME(gate,set)` retime the gate if needed, and update the respective sets. The corresponding pseudo code is listed below.

```
Algorithm FORWARD_RETIME(v, F)
{
  F ← F − v; /* remove gate v from F */
  x_i = maximum skew over all FF's at the fanins of gate v;
  x'_i = x_i + d(v);
  if (|x'_i| < |x_i|) do {
```

```
    /* retime gate v */
    for ∀u ∈ FI(v) do {w(e_uv) ← w(e_uv) − 1};
        /* delete an FF from each input */
    for ∀u ∈ FO(v) do {w(e_vu) ← w(e_vu) + 1};
        /* add an FF with skew = X'_i on all outputs */
    for ∀u ∈ FO(v) do
        if (w(e_wu) ≥ 1 ∀w ∈ FI(u)) do F ← F ∪ u; /* update F */
  }
  else
    Set skews of all FF's at the fanins of gate v to x_i.
}

Algorithm BACKWARD_RETIME(v, B)
{
  B ← B − v; /* remove gate v from B */
  x_i = minimum skew over all FF's at the fanouts of gate v;
  x'_i = x_i − d(v);
  if (|x'_i| < |x_i|) do {
    /* retime gate v */
    for ∀u ∈ FO(v) do {w(e_vu) ← w(e_vu) − 1};
        /* delete an FF from each output */
    for ∀u ∈ FI(v) do {w(e_uv) ← w(e_uv) + 1};
        /* add an FF with skew = X'_i on all inputs */
    for ∀u ∈ FI(v) do
        if (w(e_uw) ≥ 1 ∀w ∈ FO(u)) do B ← B ∪ u; /* update B */
  }
  else
    Set skews of all FF's at the fanouts of gate v to x_i.
}
```

As stated in Theorem 4.2, for forward retiming on a gate v, the effective skew before retiming, x_i, is given by the maximum skew at its inputs, while the effective skew after retiming, $x'_i = x_i + d(v)$. For backward retiming, the effective skew before retiming, x_i, is given by the minimum skew at its outputs, and the skew after retiming, $x'_i = x_i - d(u)$. In either case, the gate is retimed only if magnitude of the effective skew after retiming, x'_i, is less than the magnitude of the effective skew before retiming, x_i. The gate is then forward [backward] retimed by removing an FF from each of its inputs [outputs] and adding an FF with skew X'_i to all its outputs [inputs]. After the forward [backward] retiming a gate v, the list of gates have at least one FF on each of its fanouts [fanins] is updated.

Retiming an FF forward across a gate affects the edge weights on only its own fanouts, and not the edge weights on fanouts of any other gate. Therefore, forward retiming a gate cannot enable the backward retiming of any other gate that could not be previously retimed in the backward direction. Moreover,

since a gate is retimed only if the effective skew magnitude is reduced by this move, a gate cannot be backward-retimed after it has been forward retimed once because such a backward retiming would increase the skew magnitude. Therefore, forward retimings have no effect on backward retimings and both types of moves can be carried out independently.

Although this is not shown in the pseudocode, in the final step, each FF whose skew remains larger than $\frac{d}{2}$, where d is the maximum delay of all gates that the FF fans out to, is set to zero skew. This result works since these FF's never became candidates for relocation during the process, since the maximum effective skew at their inputs turned out to be too large for them to be retimed. Therefore, their skew can be set to zero.

The following result is true by construction, since the retiming procedure is stopped at the point where the skew magnitude can no longer be improved by retiming.

Theorem 4.3 *At the end of the retiming procedure in Phase B, the skew at each FF is no more than half a gate delay.*

Theorem 4.4 *If, at the end of the retiming procedure, all skews are set to zero, then the optimal clock period for this circuit is no more than $P_s + d_{max}$, where P_s is the optimal clock period found in Phase A, and d_{max} is the maximum delay of any gate in the circuit.*

Proof: At the end of the retiming procedure, relation (4.8) can be used to infer that the long path constraints will be satisfied if the clock period is modified to:

$$\max(P_s + x_j - x_i) \tag{4.11}$$

where x_j and x_i are the skews of any pair of FF's such that there is a purely combinational path from i to j. By Theorem 4.3, since $|x_k| < d_{max}/2 \; \forall \; k$, the clock period required to ensure that all double-clocking constraints are satisfied is no more than $P_s + d_{max}$. Note also that since any period achievable by retiming is also achievable using skews, and final period can be no better than P, which is the optimal period using skew optimization. □

After Phase B, any skews that could not be set exactly to zero are forced to zero. This could cause the clock period to increase from Phase A; however, it is shown that this increase will be no greater than the maximum gate delay. Note, however, that this is not necessarily suboptimal since the minimum clock period using skews may not be achievable using retiming, since retiming allows cycle-borrowing only in discrete amounts (corresponding to gate delays), while skew is a continuous cycle-borrowing optimization [Fis90].

Experimental Results. Table 4.1 presents experimental results on minperiod retiming using ASTRA. For each circuit, the number of gates $|G|$, the optimal clock period P, and the number of FF's in the optimal circuit $|FF|$, are shown. Also presented are the CPU time (in seconds) in terms of the time

Table 4.1. Results of Minperiod Retiming using ASTRA

Circuit	$\|G\|$	P	$\|FF\|$	T_a	T_b	T_t
s13207.1	8004	51.00	629	0.34	0.20	0.54
s15850.1	9843	63.00	565	0.97	0.24	1.21
s35932	16420	27.00	1729	0.44	0.49	0.93
s38584.1	19595	48.00	1428	1.15	0.66	1.81
s38417	21504	32.00	1619	13.22	0.50	13.72
myex1	35711	35.00	3912	2.31	1.21	3.52
myex2	40993	35.00	7837	3.58	1.69	5.27
myex3	50532	47.00	11856	102.00	2.99	104.99

spent in Phase A (T_a), the time spent in Phase B (T_b) and the total execution time (T_t). All results have been generated on a Sun UltraSparc 1/170 workstation with 128MB of memory. It is seen that a 50,000 gate circuit can be retimed within two minutes.

4.4.3 Retiming for a Target Period

Retiming a circuit for a given target clock period is a special case of the minperiod retiming problem. Given a circuit and a clock period P, if the given clock schedule is feasible, then the method should return a retimed circuit that is correctly clocked, and if the clock schedule is not feasible, this should be indicated by the method. In the Leiserson-Saxe method, a specified-period retiming is obtained by simply running the algorithm FEAS with the target clock period. In the ASTRA approach for this problem, the binary search in Phase A is not performed, but for the target clock period P only, the constraint graph is constructed as before, and the Bellman-Ford algorithm is applied to obtain the set of required skews. If the Bellman-Ford algorithm detects a positive cycle, then the clock period is not feasible, and is reported as such; otherwise Phase B is performed.

A retiming for a given clock period is, in general, not unique, and different retimed circuits can be obtained, all of which satisfy the target clock period. This may be understood through the presence of slacks in the graph at the end of the Bellman-Ford procedure, which may be used to alter the skew values within a specified range without altering the feasibility. In minperiod retiming, the objective is to find one of these possible solutions efficiently. Therefore, it makes sense to minimize the number of retiming moves. This is achieved by initializing the skews to 0 in the Bellman-Ford algorithm, and using the slacks to minimize the number of moves.

The ALAP and ASAP Retimings. Out of the set of all possible specified-period retimings, two are of particular interest. The retiming that moves all registers as far as possible against the direction of signal propagation is called an "as soon as possible" (ASAP) retiming. Similarly, the retiming that moves all the registers as far as possible in the direction of signal propagation is referred to as the "as late as possible" (ALAP) retiming. Both ASAP and ALAP retiming assume that no register is moved across the host node.

In this section we will concentrate on explaining the ASAP and ALAP retimings in the context of edge-triggered circuits. However, the reader should remain aware that the same basic principle applies to level-clocked retimings, even though we do not address the issue in detail here.

The ASAP and the ALAP locations can be seen as the extreme locations of the locations of the registers in the circuit for the specified clock period, and are unique. Their importance is in the fact that they may be utilized to make the procedure of minarea retiming more efficient, as explained in Section 5.2.

For the example circuit in Figure 4.11, with unit delay inverters and edge-triggered FF's, the ASAP and ALAP retimings for a target clock period of 3 units are shown in Figure 4.12 and Figure 4.13, respectively.

Figure 4.11. An example circuit.

Figure 4.12. ASAP retiming of the example circuit, with the ASAP constraint graph.

Figure 4.13. ALAP retiming of the example circuit, with the ALAP constraint graph.

In ASAP retiming, the objective is to move the FF's as far as possible in the backward direction. Since Phase B of ASTRA moves FF's with positive skews in the backward direction, an ASAP retiming would aim to obtain the

maximum possible skew value for each register in Phase A. Similarly, it can be argued that FF's with negative skews would be at their most backward location if the skews were set to be at their maximum (least negative) value. Therefore, in either case, an ASAP retiming corresponds to the maximum allowable skew values; symmetrically, an ALAP retiming corresponds to the minimum possible (least positive and most negative) skew values. The ASAP skews are obtained by running the Bellman-Ford algorithm on the transpose [CLR90] of the original constraint graph, i.e., a graph with the same vertex set as the original graph, but with the edge directions reversed. In this transpose graph, an edge from u to v exists if there is a combinational path from FF v to FF u; the weight of this edge is $\overline{d}(v,u) - P$. The initial skew values for the Bellman-Ford algorithm are set to $-\infty$ for all FF's, except the host which is initialized to 0. Since all of the edge directions have been reversed, the longest path values obtained for each node in the graph must be made to undergo a sign reversal to obtain the correct skew values for the corresponding FF's. The skew values so obtained are the maximum possible for the specified clock period, and no skew can be increased further without violating the clock period. For example, the transposed Phase A constraint graph for ASAP retiming of the circuit in Figure 4.11, is shown in Figure 4.12. The longest path to the node corresponding to FF1 is +1 unit, and hence, the corresponding skew on FF1 of -1 unit is obtained by reversing the sign.

Performing Phase B with these skew values results in the ASAP retiming. For ASAP locations, the available slack is used to avoid moving an FF in the direction of signal flow. The procedure for finding the ASAP (and ALAP) retiming proceeds along the same lines as in Section 4.4.2, with a few variations described below.

To obtain the ASAP locations for the retimed FF's, it is necessary to push the FF's as far as possible in a backward direction. Therefore, each FF with positive skew is moved as far as possible in the backward direction, and each FF with negative skew is moved as little as possible in the forward direction. Therefore,

(1) for an FF with positive skew x that is being moved across a single-fanout gate p in the backward direction, the skew value after the relocation at input i of p is set to $x - \text{delay}(p)$. If this value is nonpositive, then the ASAP location has been found. For gates with multiple fanouts, $x = \min_{\text{all outputs}}(x_i)$, where x_i is the skew of the FF at the ith output, as shown in Figure 4.14(a).

(2) for an FF with negative skew x that is being moved across a single-fanin gate p in the direction of signal propagation, the skew value after the relocation at output i of p is set to $x + \text{delay}(p) + \text{slack}(i)$, where $\text{slack}(i)$ is the slack associated with the output i. This slack is defined as the amount by which the delay at output i may be increased before it becomes the critical output of p; by definition, the critical output has zero slack. If the new skew is non-negative, then the ASAP location has been found. For gates with multiple

fanins, $x = \max_{\text{all inputs}}(x_i)$, where x_i is the effective skew of the FF at the ith output, as shown in Figure 4.14(b).

Applying this procedure to circuit in Figure 4.11 requires us to move FF1, which has a skew of -1 unit, across the unit delay inverter a to the location shown in Figure 4.12. At this point, the skew on FF1 is down to zero, and hence, the ASAP location has been reached. As is shown in this example if the initial circuit does not satisfy the target clock period, then ASAP retiming may require FF's to move forward, and these forward moves will be required by *all* retimings satisfying the target clock period.

Figure 4.14. Effective skews at FF's after ASAP retiming across a gate.

Since the objective in ALAP retiming is to move the FF's as far as possible in the forward direction, Phase A of ASTRA is modified to yield the minimum (most negative and least positive) skew values. All skew values are initialized to $-\infty$, except the host which is set to zero, before applying the Bellman-Ford algorithm to find the longest path in the constraint graph. In Phase B of ALAP retiming, each FF is moved as far as possible in the forward direction, using slacks if any to minimize any FF movements in the backward direction.

The Phase A constraint graph for ALAP retiming of the circuit in Figure 4.11 is shown in Figure 4.13. The longest path to the node corresponding to FF1 is +3 units, and hence, the skew on FF1 is 3 units. Since FF's with positive skews are moved as far forward as possible for ALAP retiming, FF1 moves across gates a, b, and c until its skew is brought down to zero and the ALAP location has been achieved.

4.4.4 *Continuous Retiming*

A related technique to the ASTRA algorithm is the procedure of continuous retiming (c-retiming) [Pan97]. The motivation for c-retiming is its potential use in studying synthesis and optimization problems that involve circuit modifications in conjunction with retiming, and an application of c-retiming to the tree mapping problem [De 94] is presented in [Pan97]. Unlike retiming, this method assigns a real number to each gate, representing the fractional number of FF stages to be moved across the gate. Rounding off this fractional number of FF's will result in a conventional retiming, with the resulting clock period guaranteed to be within the largest gate delay of the optimal clock period. This

provides an efficient way for minperiod retiming, since c-retiming can be solved as a single-source longest path problem. In c-retiming, the edge weights ($w_s(e)$) are also real numbers and denote the number of fractional FF's present on an edge e. The clock period of the c-retimed circuit is P if the final continuous retimed weight on each edge is no smaller than $\frac{d(v)}{P}$.

4.5 LEVEL-CLOCKED CIRCUITS

For edge-triggered circuits, the delays through all combinational logic paths must be less than the clock period, with allowances for the setup time. Therefore, timing constraints need only be enforced between FF's connected by a purely combinational path. For level-clocked circuits, the delay through a combinational logic path may be longer than one clock cycle, as long as it is compensated by shorter path delays in subsequent cycles. To ensure that the extra delay is compensated, timing constraints must be enforced from each latch to every other latch reachable from it, possibly through a path that traverses multiple latches. This greatly increases the complexity of the problem for practical circuits, as was illustrated by the example of an N-stage acyclic pipeline in Section 2.5.

4.5.1 Variation of the Critical Path with the Clock Period

As in the case of edge-triggered circuits, traditional methods [PR93b, LE94] solve the minperiod retiming problem for level-clocked circuits by performing a binary search over all possible clock periods. At each step of this binary search, the feasibility of achieving the clock period by retiming is checked by solving a single source shortest path problem, using the Bellman-Ford algorithm on a constraint graph. For a circuit with $|G|$ gates, this constraint graph consists of $|G|$ vertices and edges between every pair of vertices, constructed by solving an all-pairs shortest path problem on the original circuit graph. This graph must be reconstructed for every binary search point, because will be shown shortly, unlike in the case of edge-triggered circuits, critical paths in level-clocked circuits can be different for different clock periods. Therefore, the methods in [PR93b, LE94] have $O(|G|^2)$ memory requirements and high (although polynomial) time complexity.

In an edge-triggered circuit, the critical path is independent of the clock period, and simply corresponds to the output with the largest delay, except in the uninteresting case when the clock period is satisfied. In a level-clocked circuit, however, critical paths are more difficult to identify since they may vary from one clock period value to another. This was illustrated in [LE94] through the following example. Figure 4.15 shows a situation where three paths, p_1, p_2, and p_3, connect the vertex u to the vertex v; the delay of each node is written within its corresponding vertex. The description of each path is as follows:

Figure 4.15. Example illustrating the change in the critical path with the clock period [LE94] ©1994 IEEE.

Path	Vertices	$w(p)$	Path delay
p_1	$u \Rightarrow v_2 \Rightarrow v_3 \Rightarrow v_4 \Rightarrow v$	4	10
p_2	$u \Rightarrow v_1 \Rightarrow v$	2	3
p_3	$u \Rightarrow v_5 \Rightarrow v$	2	6

where $w(p)$ denotes the number of registers that currently lie on the path.

If a path from u to v has w registers on it, then for a symmetric two-phase clock with an active period of T_ϕ, the amount of time available for data to travel from u to v is $(w+2) \cdot T_\phi$ time units, corresponding to the total active time available from the input register of node u to the output register of node v. This leads to the following statements:

- When a symmetric two-phase clock with a period of 3 units (and therefore, an active time of 1.5 units) is applied, we require that $w_r(p_1) = 5$ and $w_r(p_3) = 2$. Therefore, the path p_1 is critical since satisfying the constraint on p_3 in the given circuit would violate the constraint on p_1.
- When the period of the same clock is changed to 5 units, the requirements are altered to $w_r(p_1) = 2$ and $w_r(p_3) = 1$; in this case, satisfying the former would lead to a violation of the latter.
- It is easily verified that the crossover point for the critical paths occurs at a clock period of 4 units, where both are critical.

Note that the path p_2 is always subservient to path p_3 since it has the same number of registers, but a smaller delay, and can, hence, never be critical.

4.5.2 *Retiming of Single Phase Level-clocked Circuits*

In [SBS91] an MILP formulation for retiming single-phase level-clocked circuits is presented. Constraints for correct clocking of single-phase circuits are laid down, which were essentially similar to the SMO constraints. A new functionality constraint was also introduced in this work, which maintains temporal equivalence between the initial and the retimed circuit. The constraint identifies a set of fundamental cycles in the directed graph corresponding to this

problem, and ensures that the sum of latches on each of these is maintained during retiming. The property of the set of fundamental cycles is that all other cycles can be expressed as a composition of these fundamental cycles. The essential idea of the approach is to build a spanning tree on the directed graph. If the graph has m edges and n vertices, then there are $m - n + 1$ edges that are excluded from the spanning tree. It is proved in [SSBS92] that each of these edges corresponds to a linearly independent cycle by showing that the cycle-edge incidence matrix has rank $m - n + 1$.

4.5.3 The TIM Approach

TIM [Pap93, PR93b] presents polynomial time algorithms for retiming and clock tuning of two-phase level clocked circuits. Since the notation used in the work differs slightly from the SMO model, we will pause for a few definitions. A clocking scheme is defined as $\pi = \langle \phi_0, \gamma_0, \phi_1, \gamma_1 \rangle$, where P is the clock period, ϕ_i the duty cycle of phase i and γ_i is the time gap between the instant where the clock phase i goes low and the clock phase $i(i+1) \bmod k$ goes high. Each vertex, v, corresponds to a gate, and is associated with a phase, $\chi(v) \in \{0, 1\}$, i.e., $\phi_{\chi(v)}$ clocks the last latch on any edge that ends at v.

The constraint for proper clocking on a path $p : u \rightsquigarrow v$ is given by the following relations:
If $\chi(u) \neq \chi(v)$,

$$\begin{aligned} d(p) \leq\ & P\left(\frac{1 + w(p)}{2}\right) + \phi_{\chi(u)} \\ & + P\lfloor \frac{r(v)}{2} \rfloor + (r(v) \bmod 2)\left(\gamma_{\chi(u)} + \phi_{1-\chi(u)}\right) \\ & - P\lfloor \frac{r(u)}{2} \rfloor - (r(u) \bmod 2)\left(\gamma_{\chi(u)} + \phi_{\chi(u)}\right) \end{aligned} \tag{4.12}$$

If $\chi(u) = \chi(v)$,

$$\begin{aligned} d(p) \leq\ & P\left(\frac{2 + w(p)}{2}\right) - \gamma_{1-\chi(u)} \\ & + P\lfloor \frac{r(v)}{2} \rfloor + (r(v) \bmod 2)\left(\gamma_{1-\chi(u)} + \phi_{\chi(u)}\right) \\ & - P\lfloor \frac{r(u)}{2} \rfloor - (r(u) \bmod 2)\left(\gamma_{\chi(u)} + \phi_{\chi(u)}\right) \end{aligned} \tag{4.13}$$

When $\chi(u) \neq \chi(v)$, $1 - \chi(u) = \chi(v)$ and the relation (4.12) enforces the constraint that the delay on any path must be less than the summation of the following terms (the interpretation of relation (4.13) is analogous):

- The first line on the right hand side is the timing allowance on the path p before retiming.
- The second line is the change in the timing allowance of the path p due to the latches that where moved either on or off p through gate v during the

retiming process. The first term ($P\lfloor\frac{r(v)}{2}\rfloor$) is the number of complete clock periods added to the timing allowance, and the second term accounts for fractional periods added. If $r(v)$ is even, no fractional periods are added and the second term is zero ($r(v) \bmod 2 = 0$). If $r(v)$ is positive, latches are shifted on to path p and the timing allowance is increased; if it is negative, latches are being shifted off from path p and the timing allowance is decreased.

If $r(v)$ is odd, the fractional change must also be considered. For a positive value of $r(v)$, the last latch moved on p is of phase $1 - \chi(v)$. Therefore, the latch on the boundary of p (which includes gate v) will be of phase $1 - (1 - \chi(v)) = \chi(v)$ which is same as $1 - \chi(u)$. Thus, the fractional period added by this last latch is $\gamma_{\chi(u)} + \phi_{1-\chi(u)}$. If $r(v)$ is negative, then the last latch moved off p by forward retiming across v is of phase $\chi(v)$, and the quantity $\gamma_{1-\chi(v)} + \phi_{\chi(v)} = \gamma_{\chi(u)} + \phi_{1-\chi(u)}$ must be subtracted from the timing allowance of path p.

- The third line is the change in the timing allowance of the path p due to latches being moved off or on p through gate u, and is derived in a similar manner to the second line.

For a more formal mathematical proof, the reader is referred to [Pap93].

These constraints in relations (4.12) and (4.13), together with the nonnegativity constraint, $r(v) + w(e_{uv}) \geq r(u)$, form an integer monotonic program that can be solved using graph-based algorithms.

A similar method for retiming level-clocked circuits was independently presented in [LE91, LE92, LE94], and was based on the SMO formulation of timing constraints.

4.5.4 *Relation Between Retiming and Skew for Level-clocked Circuits*

The work in [MS96] extends the ASTRA algorithm to level-clocked circuits. To derive timing constraints in the presence of skews, the SMO model of Section 2.5.2 is augmented with new notation. A skew, x_i, is associated with every latch $i \in \Psi$, where Ψ is the set of all latches in the circuit. It is worth pointing out at this juncture that the skew values, in the context of this algorithm, are not physical skews to be applied to the final circuit, but conceptual ideas that will eventually lead to achieve a retiming solution. However, the equations are valid for any multiphase clocking scheme in the presence of skews. No restrictions are placed on the value of x_i, i.e. $-\infty \leq x_i \leq \infty$.

We define a *latch shift* operator $L_{i,j}$, shown in Figure 4.16, much like the phase shift operator in the SMO formulation. This operator converts time from the local time zone of latch i to the local time zone of latch j, taking into account their skews. It is defined as

$$L_{i,j} = \begin{cases} (x_j + e_{p(j)}) - (x_i + e_{p(i)}) & \text{for } i < j \\ T_\Phi + (x_j + e_{p(j)}) - (x_i + e_{p(i)}) & \text{for } i \geq j \end{cases}$$

Figure 4.16. The latch shift operator.

which can be rewritten in terms of the phase shift operator as

$$L_{i,j} = (x_j - x_i) + E_{p(i),p(j)} \tag{4.14}$$

In presence of skews at latches, the timing constraints in relation (2.15) and (2.18), must be modified by using the latch shift operator instead of the phase shift operator; only long path constraints are considered here. Thus, the timing constraints for a level clocked circuit to be properly clocked by a clock schedule, Φ, in presence of skews are

$$\begin{aligned} D_i + d(i,j) - L_{i,j} &\leq D_j \quad \forall\, i \rightsquigarrow j \mid i,j \in \Psi \\ T_\Phi - T_{p(i)} \leq D_i &\leq T_\Phi \quad \forall\, i \in \Psi \end{aligned}$$

For simplicity, this discussion ignores the clock-to-Q delays and setup times, though these may easily be inserted into these equations. These timing constraints can be rewritten as

$$\begin{aligned} (x_i + D_i) + d(i,j) - E_{p(i),p(j)} &\leq (x_j + D_j) \quad \forall\, i \rightsquigarrow j \mid i,j \in \Psi \\ T_\Phi - T_{p(i)} &\leq D_i \leq T_\Phi \quad \forall\, i \in \Psi \\ -\infty &\leq x_i \leq \infty \quad \forall\, i \in \Psi \end{aligned}$$

To make the discussion simpler, we subtract T_Φ from both sides of the first relation, and substitute

$$X_i = (x_i + D_i - T_\Phi) \tag{4.15}$$

The quantity X_i is referred to as the *Global Departure Time* (GDT). Therefore, we have

$$\begin{aligned} X_i + d(i,j) - E_{p(i),P(j)} &\leq X_j \quad \forall\, i \rightsquigarrow j \mid i,j \in \Psi \\ -\infty \leq X_i &\leq \infty \quad \forall\, i \in \Psi \end{aligned}$$

These can be rewritten as the following set of difference constraints:

$$\begin{aligned} X_i - X_j &\leq E_{p(i),p(j)} - d(i,j) \quad \forall\, i \rightsquigarrow j \mid i,j \in \Psi \\ -\infty &\leq X_i \leq \infty \quad \forall\, i \in \Psi \end{aligned} \tag{4.16}$$

The difference constraint between GDT values of two latches given in relation (4.16) is similar to the difference constraints between skews at FF's in

relation (4.8). This suggests a relation between retiming and GDT values of level-sensitive latches, similar to the retiming-skew relationship for edge triggered FF's. The following theorem is similar to Theorem 4.2.

Theorem 4.5
(a) *Retiming transformations may be used to move latches from all of the n inputs of any combinational block to all of its outputs. The equivalent GDT of the relocated latch at output j, considering long path constraints only, is given by*

$$X_j = \max_{1 \le i \le n} (X_i + \overline{d}(i,j))$$

where the $X_i, 1 \le i \le n$, are the GDT's at the input latches, X_j is the equivalent GDT at output j, and $\overline{d}(i,j)$ is the worst-case delay of any path from i to j.
(b) *Similarly, latches may be moved from all of the m outputs of any combinational block to all of its inputs, and the equivalent GDT at input k, considering long path constraints only, is given by*

$$X_k = \min_{1 \le j \le m} (X_j - \overline{d}(k,j)) \tag{4.17}$$

where the $X_j, 1 \le j \le m$, are the GDT's at the input latches, X_k is the equivalent GDT at input k, and $\overline{d}(k,j)$ is the worst-case delay of any path from k to j.

Figure 4.17. The ability of a latch to absorb some skew.

The physical meaning of the GDT is as follows. If arbitrary skews could be applied at latches, the skew, x_i, of a latch could be adjusted so as to force $D_i = T_\Phi$, which is same as a negative edge triggered FF. Since $X_i = x_i + D_i - T_\Phi$, setting $D_i = T_\Phi$ gives $x_i = X_i$. Hence, we can conceive of the GDT, X_i, for a latch to be similar to the skew for an FF.

Note that in reality, we are not compelled to set $D_i = T_\Phi$, and that D_i can be reduced by as much as T_{ϕ_i}, if x_i is increased by the same amount while keeping X_i constant. Since only GDT's, X_i, appear in the timing constraint of relation (4.16), keeping them constant keeps the clock period constant. As

an illustration of this idea, consider Figure 4.17(a) where $x_i = -T_{\phi_i}$ and $D_i = T_\Phi$; this implies that $X_i = -T_{\phi_i}$. The skew here may be increased to zero ($x_i' = 0$), without changing the GDT, as shown in Figure 4.17(b), by reducing the departure time by the same amount $D_i' = T_\Phi - T_{\phi_i}$, thereby leaving the GDT unchanged ($X_i' = X_i = -T_{\phi_i}$). Therefore, a skew of up to $-T_{\phi_i}$ can be absorbed in the D_i without violating the long path constraint. Thus, any GDT value between $-T_{\phi_i}$ and 0 is permissible, and this range will be referred to as the allowable range. If different phases have different active intervals, then the allowable GDT range of a latch will depend on its phase. Therefore, in this model, level-sensitive latches can be conceptualized as FF's that have the capacity to absorb some skew.

At this time, we also note the relation between the GDT, X_i, of a latch i and the corresponding minimum magnitude skew, x_i:

$$x_i = \begin{cases} X_i & \text{if } X_i \geq 0 \\ 0 & \text{if } -T_{\phi_i} \leq X_i \leq 0 \\ X_i + T_{\phi_i} & \text{if } -T_{\phi_i} > X_i \end{cases} \tag{4.18}$$

4.5.5 Minimum Period Retiming Algorithm

The relationship between GDT and retiming presented in Section 4.5.4 can be utilized to map the problem of retiming level-clocked circuits for minimum period to the simpler problem of retiming edge-triggered circuits for minimum period as solved in Section 4.4.2. This mapping motivates the following two-phase method of retiming for minimum clock period under a specified clocking scheme.

Phase A: Find the minimum clock period and a set of GDT values that will achieve this period.

Phase B: Relocate latches across gates to get all the GDT values to be within allowable range.

The advantage of this method is that it constructs a small and sparse graph only once, unlike the traditional methods [PR93b, LE94] that construct multiple large and dense graphs. In Phase A, the skew optimization problem is solved as though the latch were a negative edge-triggered FF. In Phase B, fast local transforms are used to obtain the retiming solution. Although the latching edge in Phase A is fixed to be the negative edge, this is not a strict restriction; it is removed in Phase B when the flexibility in the GDT value is utilized to seek a zero-skew solution. This two-phase method can be used to retime large level-clocked circuits very efficiently.

As in Section 4.4.2, it must be noted that since gate delays take on discrete values, it cannot be guaranteed that the GDT at every latch can be reduced to be within the allowable range through retiming operations. After the GDT values have been reduced as much as possible, the retimed circuit may be implemented either by applying the requisite skews at a latch to get the optimal clock period achievable by skew optimization, or by setting all skews to zero

to get a clock period that is, as will be shown shortly, no more than a fixed bound above the optimal period with skews. As in the case of edge-triggered circuits, this is not necessarily suboptimal since the minimum clock period using skews may not be achievable by retiming alone, since retiming allows cycle-borrowing only in discrete amounts (corresponding to gate delays), while skew is a continuous cycle-borrowing optimization. As will be shown later in this section, if the maximum gate delay is less than the least T_{ϕ_i}, the optimal skew optimization period can always be achieved by retiming.

We first describe the two phases of minperiod retiming, followed by the special case of retiming a circuit for a specified clock period. We then present the bound on the difference between the optimal skew optimization period and the clock period obtained by our method.

Phase A: Clock Period Optimization

The problem of minimizing the clock period, T_Φ, for a specified clocking scheme can be represented as the following linear program:

$$\begin{array}{lll} \text{minimize} & T_\Phi & \\ \text{subject to} & X_i - X_j \leq E_{p(i),p(j)} - d(i,j) & i \rightsquigarrow j \mid i,j \in \Psi \end{array} \tag{4.19}$$

The solution to the above linear program yields the minimum clock period and the corresponding GDT's. Since for a specified clock schedule that has a fixed T_Φ, $E_{p(i),p(j)}$ is constant, the constraint matrix reduces to a system of difference constraints. Thus, the LP is solved as for edge-triggered circuits in Section 3.4.3.

Phase B: GDT Minimization

In Phase B, the GDT values obtained in Phase A are reduced by applying retiming transformations using Theorem 4.5. This procedure relocates the latches with nonzero GDT's across logic gates, while maintaining the optimal clock period previously found. Because of the freedom provided to D_i by the active interval of clock phase $p(i)$, which allows D_i to be set to any value between $T_\Phi - T_{\phi_{p(i)}}$ and T_Φ, $x_i = 0$ can be achieved if $-T_{\phi_{p(i)}} \leq X_i \leq 0$. If x_i cannot be set to zero, then X_i is brought as close to 0 or $-T_{\phi_{p(i)}}$ as possible so as to minimize the magnitude of the final skew x_i.

The method for FF relocation presented in Section 4.4.2 is modified to work for latches. The decision to move a latch is now based on comparing the minimum magnitude skew required before and after retiming, as given by equation (4.18), rather than by comparing the GDT's (which would have been the natural extension of the edge-triggered procedure). The following bound can be placed on the unresolved skew and the final clock period obtained at the end of Phase B.

Theorem 4.6 *At the end of the retiming procedure in Phase B, the magnitude of skew at each latch i, is no more than*

$$\gamma_{p(i)} = \max\left(0, \frac{M - T_{\phi_{p(i)}}}{2}\right)$$

where M is the maximum delay of any gate in the circuit.

Proof: Two cases must be considered:

Case A : $\underline{M \leq T_{\phi_{p(i)}}}$ If the maximum gate delay is less than the active duration of the clock, it must be proved that at the end of Phase B, all latches will have zero skew. This is proved by contradiction, assuming that a latch i has nonzero skew x_i at the end of Phase B. There are two subcases.

Subcase 1: $\underline{x_i > 0}$ In this case, the GDT of latch i is $X_i = x_i$. The new GDT of the latch after it is moved left across a gate of delay d_1 is given by $X_i' = X_i - d_1$. Since $d_1 \leq M \leq T_{\phi_{p(i)}}$, we have $X_i' \geq -T_{\phi_{p(i)}}$; therefore, the effective skew x_i' after this possible move is either zero (if $T_{\phi_{p(i)}} \leq X_i' \leq 0$), or $|x_i'| < |x_i|$. In either case, the latch i can be moved left across the gate and have its skew reduced. This contradicts the assumption that Phase B is complete.

Subcase 2: $\underline{x_i < 0}$ If $x_i < 0$ then the GDT of latch i is negative, i.e., $X_i = x_i - T_{\phi_{p(i)}}$, and the proof proceeds in a similar manner as subcase 1.

Figure 4.18. Worst-case situation for remaining skew.

Case B: $\underline{M > T_{\phi_{p(i)}}}$ If the maximum gate delay is more than the active duration of the clock, we need to prove that at the end of Phase B, the skew magnitude for any latch i is less than $\gamma_{p(i)}$. Phase B is complete only when for every latch i, $|x_i'| \geq |x_i|$, where x_i is the current skew and x_i' is the skew after a possible move across a gate with delay d_1. As shown in Figure 4.18, the largest possible final skew magnitude corresponds to the situation when $|x_i| = |x_i'|$. In this case, we have $d_1 = 2 \cdot |x_i| + T_{\phi_{p(i)}}$, and hence, $|x_i| \leq \frac{d_1 - T_{\phi_{p(i)}}}{2}$. Since M is the maximum delay of any gate in the circuit, this implies that $|x_i| \leq \frac{M - T_{\phi_{p(i)}}}{2}$.

Theorem 4.7 *If, at the end of the retiming procedure, all skews in a k-phase circuit are set to zero, then the final clock period (P_r) satisfies the following*

condition

$$P_r \leq P_s + \sum_{i=1}^{k} \max(0, M - T_{\phi_i})$$

where P_s is the optimal clock period with skews found in Phase A, and M is the maximum delay of any gate in the circuit.

Proof : Each difference constraint for the optimal clock period (with skews) is of the form

$$X_i - X_j \leq E_{p(i),p(j)} - d(i,j).$$

Theorem 4.6 guarantees us that at the end of Phase B, $|X_i|$ and $|X_j|$ are within $\gamma_{p(i)}$ and $\gamma_{p(j)}$ of their optimal values respectively. Therefore, the actual value of $X_i - X_j$ after Phase B must lie within $(\gamma_{p(i)} + \gamma_{p(j)})$ of the required value of $X_i - X_j$ in Phase A. This implies that the inequality that defines the difference constraint can be maintained by increasing $E_{p(i),p(j)}$ by no more than $(\gamma_{p(i)} + \gamma_{p(j)})$. Since each $E_{i,j}$ increases by no more than $(\gamma_{p(i)} + \gamma_{p(i)})$, the clock period $T_\Phi = \sum_{i=1}^{k} E_{p(i),p(j)}$ will increase by no more than $\sum_{i=1}^{k}(2 \cdot \gamma_{p(i)}) = \sum_{i=1}^{k} \max(0, M - T_{\phi_i})$.

4.5.6 Experimental Results

The minimum period retiming algorithm for level-clocked circuits, described above, was implemented as a C program, ASTRA-L [MS96]. Experimental results from applying this program to several two-phase circuits are shown in Table 4.2. For each circuit, the number of gates $|G|$, the optimal clock period P, the number of FF's in the optimal circuit $|FF|$. Also presented are the CPU time (in seconds) in terms of the time spent in Phase A (T_a), the time spent in Phase B (T_b) and the total execution time (T_t). As before, the CPU times reported are for a Sun UltraSparc 1/170 workstation with 128 MB of RAM. The execution times are extremely fast and a 50,000 gate circuit is retimed in about seven minutes.

4.6 CONCLUDING REMARKS

The retiming problem was introduced in this chapter, and some basic formulations and algorithms presented by Leiserson and Saxe have been overviewed. Practical implementational issues related to algorithms for solving these problems have also been examined, both for edge-triggered circuits and for level-clocked circuits. The relationship between clock skew and retiming has been presented for both classes of circuits. Efficient algorithms exploiting this relationship, which can handle circuits with tens of thousands of gates in minutes on an everyday workstation, have been presented.

Table 4.2. Results of Minperiod Retiming of 2 Phase Circuits using ASTRA-L

Circuit	$\|G\|$	P	$\|FF\|$	T_a	T_b	T_t
s13207.1	8004	51.00	1273	0.36	0.27	0.63
s15850.1	9843	63.00	1639	1.01	0.58	1.59
s35932	16420	27.00	5207	6.38	1.36	7.74
s38584.1	19595	48.00	8889	1.19	1.36	2.55
s38417.	21504	31.50	4863	26.68	1.40	28.08
myex1	35711	35.00	11258	2.75	3.16	5.91
myex2	40993	35.00	16291	101.03	2.66	103.69
myex3	50532	46.75	22475	419.39	5.57	424.96

Notes

1. An alternative view of level-clocked latches is as flip-flops that naturally permit a skew range of up to the active range of the latch. If, for example, the rising edge of the clock is set to time zero and the active pulse width of the clock is w, then a level-clocked latch has the same effect as an edge-triggered latch with an allowable skew of 0 through $+w$.

2. Since y is a negative number, $d(v) - y$ will be positive, as expected.

3. The calculation of indices i here span the range $0, \cdots, k$, and in this discussion, $i - 1$ is calculated modulo k.

4. This result is easy to see on this simple example. The reader is referred to Chapter 3 for a general procedure for finding the optimal skews for a general circuit

5. In general, it is not possible to come up with an equivalent skew value that satisfies both long and short path constraints. For example, when we consider short path constraints, moving flip-flops from the input to the output requires that the new skew be

$$\min_{1 \le i \le n} (x_i + \underline{d}(i,j))$$

(using the same notation as Theorem 4.2(a)), which is incompatible with the requirement stated above except in the special case where all paths from i to j have the same delay. This is not an impediment here since the retiming problem as stated by Leiserson and Saxe in [LS91] considers long path constraints only. Note that Theorem 4.1 is valid for both long path and short path constraints, and can be considered to be a special case where the combinational subcircuit has only one path, where clearly the longest and shortest path delays through the gate are equal.

5 MINAREA RETIMING

As shown in Section 4.1, retiming can reduce the clock period of a circuit. However, in doing so, it is possible that it may greatly increase the number of registers. In fact, given any clock period for a general circuit, there will, in general, be not one, but several retiming solutions that satisfy the clock period; these differ in the manner in which they utilize the slack on noncritical paths. The ASAP and ALAP retimings from Section 4.4.3 are just two of these possible retimings, and each of these differs in terms of the number of registers that it employs.

The objective of the minarea retiming problem is to find one of these retiming solutions at a given clock period, chosen in such a way that it has the minimum number of registers over all feasible retimings satisfying the period. This is a practical and meaningful objective, since minimizing the number of registers would minimize the area of the circuit, as the retiming transformation leaves the combinational logic untouched.

In this chapter, we will first overview the Leiserson-Saxe framework for minimum area retiming. Next, an efficient method for reducing the complexity of this method, using bounds on the retiming variables derived from the equivalence between clock skew and retiming, is presented. Finally, the extension of this approach to level-clocked circuits is described.

N. Maheshwari et al., *Timing Analysis and Optimization of Sequential Circuits*
© Kluwer Academic Publishers 1999

5.1 THE LEISERSON-SAXE APPROACH

5.1.1 A Basic Formulation

A mathematical programming formulation of the minarea retiming problem was presented in [LS91], and is reproduced here. Let the total number of registers in a circuit G be given by $S(G) = \sum_{e \in E} w(e)$. The reader is referred back to Section 4.3 for the basic concepts and terminology used in the representation of a circuit in terms of a retiming graph. Using this notation, the total number of registers in a circuit after retiming, $S(G_r)$, can be calculated as follows:

$$S(G_r) = \sum_{e \in E} w_r(e) \tag{5.1}$$

$$= \sum_{e:u \to v \ \in E} w(e) + r(v) - r(u) \tag{5.2}$$

$$= S(G) + \sum_{v \in V} r(v)(|FI(v)| - |FO(v)|) \tag{5.3}$$

where $FI(v)$ and $FO(v)$ represent the fanin and fanout sets of the gate v. Since $S(G)$ is a constant, the minarea retiming problem for a target period P can be formulated as the following linear program (LP):

$$\begin{array}{lll} & \text{minimize } \sum_{v \in V} [(|FI(v)| - |FO(v)|) \cdot r(v)] & \\ \text{subject to} & r(u) - r(v) \leq w(e_{uv}) & \forall e_{uv} \in E \\ & r(u) - r(v) \leq W(u,v) - 1 & \forall D(u,v) > P \\ & -\infty \leq r(u) \leq \infty & \forall u \ \in V \end{array} \tag{5.4}$$

The significance of the objective function and the constraints is as follows.

- The objective function represents the number of additional registers added to the retimed circuit, with reference to the original circuit.

- The first constraint ensures that the weight e_{uv} of each edge (i.e., the number of registers between the output of gate u and the input of gate v) after retiming is nonnegative. We will refer to these constraints as *circuit constraints.*

- The second constraint ensures that after retiming, each path whose delay is larger than the clock period has at least one register on it. These constraints, being dependent on the clock period, are often referred to as *period constraints.*

This problem formulation is in the form of a dual of a minimum-cost network flow problem. Hence, the LP can be solved efficiently by solving this dual [BJS77, AMO93].

Figure 5.1. Unconditional register sharing at multiple fanouts.

5.1.2 A More Accurate Area Model Using Mirror Vertices

The cost function in the LP (5.4) assumes that each FF has exactly one fanout. However, in practice an FF can have multiple fanouts, allowing the FF's on different fanout edges of a gate to be shared. This sharing must be taken into account for an accurate area model.

As an example, consider gate A in Figure 5.1 with three fanouts, B, C, and D, having three, two and two FF's, respectively. The LP in (5.4) will model the total number of FF's as seven as shown in Figure 5.1(a). However, the FF's can be merged or shared as shown in Figure 5.1(b), resulting in a total cost of only three FF's.

To model the maximal FF sharing[1], the work in [LS91] introduces a mirror vertex m_i for each gate i that has more than one fanout, as shown in Figure 5.2, the details of which can be found in [Sax85]. Each edge e_{ij}, in addition to having a weight $w(e_{ij})$, now also has a width $\beta(e_{ij})$. In Figure 5.2, the edge weights are shown above the edges while the edge widths are shown below the edges. Consider a gate u with k fanouts to gates v_j, $j = 1, \cdots, k$. To model the maximum sharing of FF's, an extra edge is added from each fanout gate, v_j, to the mirror vertex, m_u, with a weight of $w(e_{v_j m_u}) = w(\max_u) - w(e_{uv_j})$. Here, $w(\max_u) = \max_{\forall i \in FO(u)}(w(e_{ui}))$ is the maximum weight on any fanout edge of gate u. Each of the edges from the gate i to its fanouts j, and from the fanouts to the mirror vertex has a width of $1/k$, i.e.,

$$\beta(e_{uv_j}) = 1/k \text{ and } \beta(e_{v_j m_u}) = 1/k \quad \text{for } j = 1, \cdots, k.$$

The original LP in (5.4) is modified to include the effect of register sharing as follows:

$$\min \sum_{v \in (V \cup M)} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right]$$

$$\text{subject to} \quad \begin{array}{ll} r(u) - r(v) \leq w(e_{uv}) & \forall e_{uv} \in E \\ r(u) - r(v) \leq W(u,v) - 1 & \forall D(u,v) > P \end{array} \tag{5.5}$$

Figure 5.2. Model for maximum register sharing at multiple fanouts.

$$\begin{aligned} r(j) - r(m_i) &\leq w(\max_i) - w(e_{jm_i}) && \forall (m_i) \in M \text{ and } \forall j \in FO(i) \\ -\infty \leq r(u) &\leq \infty && \forall u \in (V \cup M) \end{aligned}$$

where $M = \{m_v | v \in V \text{ and } |FO(v)| > 1\}$ is the set of all the mirror vertices, and additional constraints due to the mirror vertices are called the mirror constraints. For simplicity, we can rewrite the above LP as follows

$$\min \sum_{v \in (V \cup M)} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right]$$

$$\begin{aligned} \text{subject to} \quad r(u) - r(v) &\leq c_{uv} && \forall (u,v) \in C \\ -\infty \leq r(u) &\leq \infty && \forall u \in (V \cup M) \end{aligned} \tag{5.6}$$

where $C = C_p \cup C_c \cup C_m$ is the constraint set of the LP in (5.5), and includes the period constraint set (C_p), the circuit constraint set (C_c) and the mirror constraint set (C_m). A constraint (i,j) in the constraint set C is of the form

$$r(i) - r(j) \leq c_{ij} \forall (i,j) \in C$$

where

$$\begin{aligned} c_{ij} &= w(e_{ij}) && \forall (i,j) \in C_c, \text{ i.e., } e_{ij} \in E \\ c_{ij} &= W(i,j) - 1 && \forall (i,j) \in C_p, \text{ i.e., } D(i,j) > P \\ c_{ij} &= w(\max_i) - w(e_{jm_i}) && \forall (i,j) \in C_m, \text{ i.e., } m_i \in M, \forall j \in FO(i) \end{aligned} \tag{5.7}$$

The objective function of the LP in (5.6) now denotes the increase in the number of FF's assuming maximal sharing of FF's at the output of all gates. The weights on all paths from gate u to its mirror vertex m_u are the same before retiming, i.e., $w(e_{uv_i}) + w(e_{v_i m_u}) = w(\max_i), 1 \leq i \leq k$, and therefore, the weights on all paths from gate u to its mirror vertex m_u must be equal after retiming. Since the mirror vertex m_u is a sink in the graph, the register count on one of the edges from the fanout nodes to m_u will be zero, i.e., $\exists i \; |w(e_{v_i,m_u}) = 0$. Thus, the weight on all paths from gate u to mirror vertex m_u after retiming will be $w_r(\max_u) = \max_{\forall j \in FO(u)}(w_r(e_{uj}))$, since all of the retimed edge weights $w_r(e_{jm_u}) \geq 0 \forall j \in FO(u)$. As there are k paths, each with width $1/k$, the total cost of all paths will be $w_r(\max_u)$ as desired. Like the LP in (5.4), the LP in (5.6) is also the dual of a minimum cost network flow problem.

An alternative view of this model is as follows. The change in cost function due to adding or removing FF's from the fanout junction of gate u is modeled by two retiming variables: one for the gate, $r(u)$ and other for the mirror vertex, $r(m_u)$. Any change in the cost function due to FF's moving across the multi-fanout gate itself are modeled by $r(u)$, while any change due to FF motion across its fanout gates v_i, $1 \leq i \leq k$ is modeled by the mirror variable $r(m_u)$.

The change in the number of FF's in the circuit, under maximal sharing obtained by retiming a gate u by one unit can be calculated as follows. The decrease in the cost function obtained by removing an FF from each of the fanouts of a gate is one unit, even for multiple fanout gates since the FF's on all the fanouts were shared. The increase in the cost function from adding an FF to all the inputs of a gate u is equal to the number of fanins of u that have only one fanout, since any FF added to a fanin j of gate u that has more than one fanout ($|FO(j)| > 1$) is already modeled by the mirror variable of that fanin gate m_j. Thus, the cost contribution of any single fanout gate u is given by $(|FI'(u)| - 1) \cdot r(u)$, while that of a multi-fanout gate is given by $(|FI'(u)| - 1) \cdot r(u) + r(m_u)$, where $FI'(u)$ is the set of fanins that have only a single output, i.e., $FI'(u) = \{v | v \in FI(u) \text{ and } |FO(v)| = 1\}$. Therefore, the cost function may be written as the summation of these terms over all gates.

5.1.3 *Efficient Implementation*

Although the LP in (5.4) can be efficiently solved by solving the dual, the computation of W and D matrices requires $O(|V|^3)$ time and $O(|V|^2)$ memory. Further, the number of period edges required can be very large, destroying the sparsity of the circuit graph. Because of these reasons, the minarea retiming problem cannot be solved in reasonable time using the method from [LS91]. Shenoy and Rudell in [SR94] presented an efficient implementation, where they proposed a constraint generation method requiring $O(|V|^2|E|log|V|)$ time but only $O(|V|)$ memory, and a technique to prune the number of period constraints.

The algorithm uses a combination of the Dijkstra's algorithm and the Bellman-Ford algorithm. It works by generating one row, say the (s^{th}) row, of the W and the D matrix at a time. An ordered pair $(w(e_{ij}), -d(i))$, denoted by (a_i, b_i),

is associated with each edge e_{ij} and is used to compute the shortest distance from a source vertex s. A heap is maintained for each distinct value of a_i and is indexed by this value. Until all heaps are empty, the node u at the top of the minimum index heap is extracted using the function `pop-min(heap index)`. The fanouts of u are then added to the appropriate heaps if their a_u or b_u values are updated by Bellman-Ford relaxation. At the end of this procedure $D(s,u) = -b_u$ and $W(s,u) = a_u$.

Note that to satisfy a clock period, P, the only requirement is to ensure that each path with delay greater than P has at least one FF on it. The number of FF's on any path is monotonic with respect to the path length since negative edge weights are not allowed. Due to this monotonicity of edge weights, if at least one FF is placed on a sub-path, then it is guaranteed that at least one FF will exist on all paths containing this sub-path. Adding a period constraint from s to u is one way to ensure at least one FF on all paths from s to u. The idea is to add a period edge to only the vertex v, reachable from s, that satisfies the following:

$$D(s,v) > P \text{ and } D(s,u) \leq P \; \forall \, u \text{ on } s \rightsquigarrow FI(v) \tag{5.8}$$

where $s \rightsquigarrow FI(v)$ is a path from s to a fanin of v. Thus, if the period constraint is added, the fanouts of u need not be relaxed. The pseudocode for constraint generation is as follows:

```
Algorithm CONSTRAINT_GENERATION
{
  P = target clock period;
  S_k= the k^th heap;
  ∀s ∈ V {
    s = current vertex;
    ∀v ∈ V, a_v = ∞ and b_v = 0;
    S_0 = {s}; a_s = 0,; b_s = -d(s);
    k = current register weight;
    do {
       k = min{p | S_p ≠ ∅};
       u = pop-min(S_k) ;
       if (-b_u > P)
           add a period edge c(s,u) with weight a_u - 1
       else {
           ∀v ∈ FO(u) {
                if((a_v, b_v) > (a_u + w(e_{u,v}), b_v - d(v)))
                heap-insert(S_{a(u)+w(e_{u,v})}, v);
           }
       }
    } while(∃ p | S_p ≠ ∅)
  }
}
```

The work in [vVA+94], published a few months before [SR94], used the same idea and referred to it as "clock-period limited labeling." It was demonstrated that the use of this method caused a significant speedup in constraint generation. A second method presented in the work was termed "relevant path labeling." Starting from a source node u, this technique labels each vertex v with the maximum value of the label $l(u,v) = \frac{D(p(u \rightsquigarrow v))}{P} - W(p(u \rightsquigarrow v))$. The rationale behind this approach is that for every path originating at u, it must be true that $\frac{D(p(u \rightsquigarrow v))}{P} \leq W(u \rightsquigarrow v)$, and therefore, the largest value of $l(u,v)$ must be nonpositive. However, unlike the method of clock-period limited labeling, this must traverse all vertices, and while it generates a smaller number of constraints, the amount of time required to generate these constraints was reported to be extremely large.

5.2 THE MINARET ALGORITHM

Although the techniques of the previous section make minarea retiming efficient, they cannot handle large circuits with tens of thousands of gates. In [MS97a, MS98b], an amalgamation of the Leiserson-Saxe approach and the ASTRA approach is used for efficient minarea retiming. By utilizing the merits of both approaches, an efficient algorithm for constrained minarea retiming was developed. This algorithm, called MINArea RETiming (Minaret), is capable of handling very large circuits in very reasonable runtime. The basic idea of the approach is to use the ASTRA approach to find tight bounds on the retiming variables. These bounds help reduce both the number of variables and the number of constraints in the problem without any loss in accuracy. By spending a small amount of additional CPU time on the ASTRA runs, this method leads to significant reductions in the total execution time of the minarea retiming problem. The reduction in the problem size also reduces the memory requirements, thus enabling retiming of large circuits.

The approach in Minaret is to find tight bounds on the r variables, and to use these bounds to avoid generating redundant constraints. By appropriate application of these bounds, not only is the constraint set pruned but the number of variables is also reduced. In this way, the size of the LP is reduced enabling it to be solved more efficiently. The reduced constraints can be generated efficiently by using these bounds. Note that the exactness of the solution is not sacrificed in doing so, since none of the essential constraints are removed.

We will now show the relation between the Leiserson-Saxe approach and the ASTRA approach, and how a modified version of ASTRA can be used to derive bounds on the r variables in the Leiserson-Saxe method. Next, we show how these bounds can be used to prune the number of constraints in minarea Leiserson-Saxe retiming. Finally, we present an example to illustrate the method.

5.2.1 *The Concept of Restricted Mobility and Bounds for the r Variables*

For the circuit in Figure 5.3, to achieve the minimum clock period of 4.0 units, one must move one copy of FF B to the output of gate G4. The possible locations for FF's along the other path to FF C are at the input to gate G8, or at the output of gate G8, or the inputs of gates (G9,G10) or the outputs of gates (G9,G10); no other locations are permissible

Figure 5.3. Possible FF locations after retiming.

Therefore, it can be seen that the FF's cannot be sent to just any location in the circuit; rather, there is a restricted range of locations into which each FF may be moved, and the mobility of each FF is restricted. This restricted mobility may be used to reduce the search space, and hence the number of constraints.

The concept of restricted mobility is related to the ASAP and ALAP retimings presented in Section 4.4.3 in that the ASAP and ALAP retiming solutions define the boundaries of the region into which an FF can move during retiming while satisfying the target clock period. For example, Figures 5.3(a) and (b) are the ASAP and ALAP retiming for the given clock period of 4.0 units and the FF can move only in the region defined by these locations. These ASAP and ALAP retiming solutions can be used to obtain bounds on the retiming variables of the Leiserson-Saxe approach, r, associated with the gates in the circuit as illustrated by the following example.

Example: For the circuit in Figure 5.3, the locations for the FF's in the retimed circuit corresponding to the ASAP and ALAP retiming solutions are shown in Figure 5.3(a) and (b), respectively. This implies that during retiming, no FF will move across gates G1, G2, G5, G6, G7, G11 and G12; one FF each will move from the input to the output of gates G3 and G4, and either no FF

or one FF will move from the input to the output of gates G8, G9 and G10. Referring to Section 4.3 for the definition of the r variables, this implies that one may set the following bounds on the r variables.
(1) $r(u) = 0$ for $u \in \{G1, G2, G5, G6, G7, G11, G12\}$
(2) $r(u) = -1$ for $u \in \{G3, G4\}$, and
(3) $-1 \leq r(u) \leq 0$ for $u \in \{G8, G9, G10\}$. □

While moving the FF's in Phase B of ASAP and ALAP retimings, subject to the specified clock period P, the number of FF's that traverse each gate is counted; this count is the upper and lower bound, respectively, on the r variables for each gate. An FF moving from the inputs to the output of a gate decrements the count by one, while one moving from the output to the inputs increments it by one. For the ASAP case, FF's are moved as far as possible in the backward direction. In other words, we relocate the largest number of FF's possible from the output to the inputs of a gate. By the definition of the r variables, this gives an upper bound on r for the gates. Similarly, the ALAP retiming relocates the largest number of FF's that can move from the inputs of a gate towards its output, and this provides a lower bound on the r values for the gates in the circuit. The bounds on the r variable corresponding to each gate y are of the form.

$$L_y \leq r(y) \leq U_y \tag{5.9}$$

We will refer to L_y as the lower bound for gate y and to U_y as the upper bound of gate y. Like the ASAP and ALAP retimings, these bounds are with reference to a fixed host vertex, i.e., $L_{\text{host}} = U_{\text{host}} = 0$. If $U_u = L_u = k_u$, we say that gate u is *fixed* or immobile, since $r(y) = k_y$ is not really a variable any more. On the other hand, if $U_y \neq L_y$, we say that gate y is *flexible* or mobile. Thus, we can reduce the variable set V of the Leiserson-Saxe model to $V' \subseteq V$, the variable set of Minaret where

$$V' = \{v \in V | U_v \neq L_v\} \tag{5.10}$$

Bounds on the mirror vertices, introduced to model the maximal latch sharing can be obtained directly from the bounds on fanout gates (as given by Theorem 5.1). The mirror variable set M is also reduced to $M' \subseteq M$ the mirror variable set of Minaret where

$$M' = \{m \in M | U_m \neq L_m\} \tag{5.11}$$

Theorem 5.1 *The bounds on the r value of a mirror vertex m_i of gate i in Figure 5.2 can easily be derived from the bounds on the fanout gates and are given by*

$$\begin{aligned} U_{m_i} &= \max_{\forall j \in FO(i)} (U_j + w(e_{ij})) - w(max_i) \\ L_{m_i} &= \max_{\forall j \in FO(i)} (L_j + w(e_{ij})) - w(max_i) \end{aligned}$$

Proof: After optimal retiming, the weight on at least one of the edges to the mirror vertex (see Figure 5.1) will be zero [LS91]. Hence,

$$\min_{\forall j \in FO(i)} (w_r(e_{jm_i})) = 0$$

$$\begin{aligned}
\min_{\forall j \in FO(i)} (w(e_{jm_i}) + r(m_i) - r(j)) &= 0 \\
\text{i.e.,} \min_{\forall j \in FO(i)} (w(e_{jm_i}) - r(j)) &= -r(m_i) \\
\text{i.e., } r(m_i) &= \max_{\forall j \in FO(i)} (r(j) - w(e_{jm_i})) \\
\text{Since } r(j) &\leq U_j \ \forall j \in FO(i) \\
\text{we have } \max_{\forall j \in FO(i)} (r(j) - w(e_{jm_i})) &\leq \max_{\forall j \in FO(i)} (U_j - w(e_{jm_i})) \\
\text{Therefore, } r(m_i) &\leq \max_{\forall j \in FO(i)} (U_j - w(e_{jm_i}))
\end{aligned}$$

Thus, the upper bound is

$$U_{m_i} = \max_{\forall j \in FO(i)} (U_j + w(e_{ij})) - w(max_i)$$

After retiming, all edge weights including edges to mirror vertices must be nonnegative, i.e.,

$$\begin{aligned}
w_r(e_{jm_i}) &\geq 0 \quad \forall j \in FO(i) \\
\text{or } w(e_{jm_i}) + r(m_i) - r(j) &\geq 0 \quad \forall j \in FO(i) \\
\text{i.e., } r(m_i) &\geq r(j) - w(e_{jm_i}) \quad \forall j \in FO(i) \\
\text{i.e., } r(m_i) &\geq r(j) + w(e_{ij}) - w(max_i) \quad \forall j \in FO(i) \\
\text{or } r(m_i) &\geq \max_{\forall j \in FO(i)} (r(j) + w(e_{ij})) - w(max_i)
\end{aligned}$$

Therefore, the lower bound is

$$L_{m_i} = \max_{\forall j \in FO(i)} (L_j + w(e_{ij})) - w(max_i)$$

5.2.2 *Eliminating Unnecessary Constraints*

In this section, we illustrate how the addition of bounds (derived previously) to the LP of (5.6) in Section 5.1.2 may be used to reduce the constraint set by dropping redundant constraints. It can be seen from the bounds on $r(i)$ and $r(j)$ in relation (5.9) that $r(i) - r(j) \leq U_i - L_j$. Therefore, if $U_i - L_j \leq c_{ij}$ then $r(i) - r(j) \leq c_{ij}$ is also true, and the constraint (i, j) can be dropped. Thus, the Leiserson-Saxe constraint set C can be reduced to the Minaret constraint set $C' \subseteq C$ where

$$C' = \{(i, j) \in C | \ U_i - L_j > c_{ij}\} \tag{5.12}$$

Notice that constraints associated with fixed or immobile gates can be treated as bounds and need not be included in C'. Like the Leiserson-Saxe constraints, the Minaret constraints also consists of circuit, period and mirror constraints, i.e., $C' = C'_c \cup C'_p \cup C'_m$, where C'_c is the reduced circuit constraint set, C'_p is the reduced period constraint set, and C'_m is the reduced mirror constraint set.

5.2.3 The Reduced Linear Program

We use the relations (5.10), (5.11) and (5.12) to reduce the LP in (5.6) to the following LP in Minaret

$$\min \sum_{v \in \{V' \cup M'\}} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] \tag{5.13}$$

$$\text{subject to} \quad \begin{aligned} & r(u) - r(v) \leq c_{uv} \;\; \forall (u,v) \in C' \\ & L_u \leq r(u) \leq U_u \;\; \forall u \in (V' \cup M') \end{aligned}$$

An Example. We now illustrate this method and show how the number of constraints can be reduced using our approach.

Consider the circuit example shown in Figure 5.4. As in the previous examples, we make the assumption that the gates have unit delays. We consider two possible clock periods of 2 units and 3 units in this example.

Figure 5.4. Example illustrating the approach.

When $P = 2$ units. For a clock period of two units, the list of constraints generated by the approach in [SR94] is listed below.

<u>Circuit constraints</u>	$r(h) - r(a)$	≤ 1
	$r(a) - r(b)$	≤ 0
	$r(b) - r(c)$	≤ 0
	$r(c) - r(d)$	≤ 0
	$r(d) - r(h)$	≤ 0
<u>Period constraints</u>	$r(h) - r(c)$	≤ 0
	$r(a) - r(c)$	≤ -1
	$r(b) - r(d)$	≤ -1

Note that
(a) the delay associated with the host node is zero, and
(b) the value of $r(h)$ is set to zero as a reference, so that it is not really a variable.
Therefore, this is a problem with <u>four</u> variables and <u>eight</u> linear constraints (of which three act as simple bounds).

In our approach, for a clock period of 2, we first find the bounding skews. The FF's at the input and output may not be moved, and therefore, the only movable FF is FF1, which is assigned a skew of -2 units. The correctness of this skew value is easy to verify since the only feasible location of FF1 under $P = 2$ is two delay units to the right of its current location. Therefore, we find that by using the concept of restricted mobility,

$$\begin{aligned} -1 \leq r(a) \leq -1 &\Rightarrow r(a) = -1 \\ -1 \leq r(b) \leq -1 &\Rightarrow r(b) = -1 \\ 0 \leq r(c) \leq 0 &\Rightarrow r(c) = 0 \\ 0 \leq r(d) \leq 0 &\Rightarrow r(d) = 0 \end{aligned}$$

Since all nodes are fixed, and all the constraints can be dropped, all of the constraints and variables have been eliminated!

When $P = 3$ units. With the clock period is set to 3 units, the list of constraints is

$$\begin{array}{lll} \underline{\text{Circuit constraints}} & r(h) - r(a) & \leq 1 \\ & r(a) - r(b) & \leq 0 \\ & r(b) - r(c) & \leq 0 \\ & r(c) - r(d) & \leq 0 \\ & r(d) - r(h) & \leq 0 \\ \underline{\text{Period constraints}} & r(h) - r(d) & \leq 0 \\ & r(a) - r(d) & \leq -1, \end{array}$$

As before, $r(h) = 0$ is set as a reference, giving a problem with <u>four</u> variables (as before) and <u>seven</u> linear constraints (of which three act as simple bounds).

Under our approach, the relocated FF can reside either at the input of gate b, the output of gate b, or the output of gate c. Therefore, we have

$$\begin{aligned} -1 \leq r(a) \leq -1 &\Rightarrow r(a) = -1 \\ -1 \leq r(b) \leq 0 & \\ -1 \leq r(c) \leq 0 & \\ 0 \leq r(d) \leq 0 &\Rightarrow r(d) = 0 \end{aligned}$$

Using these bounds we drop all constraints but

$$r(b) - r(c) \leq 0$$

Therefore, we have reduced the problem complexity to <u>two</u> variables, each with fixed upper and lower bounds and <u>one</u> linear constraint. (Note that upper/lower bound constraints are typically much easier to handle in LP's than general linear constraints; in fact, in many cases, upper and lower bounds are actually helpful in solving the LP.)

5.2.4 Generating the Linear Program

Using the alternative description of the maximal FF sharing in Section 5.1.2, the objective function coefficients are obtained by inspection of the circuit, without explicitly adding the mirror vertices. The circuit and the mirror constraints in C' are obtained from direct inspection of the circuit graph using relation (5.12). Because the bounds on the mirror vertices can also be obtained directly from the bounds on the gate vertices, we do not need to explicitly add the mirror vertices to the circuit graph. Since every multi-fanout gate has a mirror vertex, this gives us important savings in terms of the space and time requirements. We now describe how to obtain the period constraints in C'.

We take advantage of the bounds obtained in Section 5.2.1 to modify the method from [SR94] to run faster, generating only the reduced constraint set C'.

As noted earlier, due to the monotonicity of edge weights, if we ensure at least one FF on any sub-path, we are assured of having at least one FF on all paths that contain this sub-path. Therefore, if the bounds on the r variables guarantee us at least one FF on any sub-path, we need not process any path containing this sub-path. We use this observation in addition to relation (5.8) to reduce the number of period constraints.

At the end of the ASTRA run for obtaining the lower bounds, all FF's are in their ALAP locations. If the delay of all the gates is not the same, it is possible that retimed circuit obtained by ASTRA with FF's in the ALAP locations may have some purely combinational paths with delays that are greater than the target clock period P. However, in practice, most of the other paths satisfy the target clock period. We will use this observation to further speed up the constraint generation process.

Consider a fixed gate a in the circuit at the end of the ALAP run. If none of the combinational paths starting at this gate violate the clock period, we have $W_{ALAP}(a,i) \geq 1$ if $D(a,i) > P \quad \forall i$. Since $W_{ALAP}(a,i) = W(a,i) + L_i - L_a$ we have $L_a - L_i \leq W(a,i) - 1$, or $L_a - L_i \leq c_{a,i}$. Since gate a is fixed $U_a = L_a$, we obtain $U_a - L_i \leq c_{a,i} \quad \forall i \in V$, which is guaranteed to be true, and hence, all constraints starting from fixed gate a are redundant, and we do not need to generate them. Thus, we must generate period constraints only from those fixed gates which have at least one purely combinational path starting from it with delay more than the clock period. Let us call this set V''.

The method from [SR94] presented in Section 5.1.3 is modified to take advantage of the bounds on the r variables to generate the reduced constraint set C' efficiently. The pseudocode is presented below with the modifications shown in boldface.

```
Algorithm CONSTRAINT_GENERATION_WITH_BOUNDS
{
  P = target clock period;
  L_i ≤ r(i) ≤ U_i  ∀i ∈ V;
  S_k= the k^th heap;
  L_min = min(L_i)  ∀i ∈ V;
  ∀s ∈ V' ∪ V''  {
      s = current vertex;
      ∀v ∈ V, a_v = ∞ and b_v = 0;
      S_0 = {s}, a_s = 0,  and  b_s = -d(s);
      k = current register weight;
      do {
          k = min{p | S_p ≠ ∅};
          if (k ≥ (U_s - L_min + 1)) break;
          u = pop-min(S_k) ;
          if ((U_s - L_u) ≤ k - 1) continue;
          if (-b_u > P)
          add a period edge c(s,u) with weight a_u - 1
          else {
               ∀v ∈ FO(u) {
                    if ((k - U_s + L_v) < 1)  {
                       if((a_v, b_v) > (a_u + s_u,v, b_v - d(v)))
                       heap-insert(S_a(u)+s_u,v, v);
                    }
               }
          }
      } while(∃p | S_p ≠ ∅)
  }
}
```

5.2.5 *Solving the Linear Program*

Like (5.6), the LP in (5.13) is also a dual of a minimum cost network flow problem. We found that it could be solved very efficiently using the network simplex algorithm from [BJS77]. The network simplex method is a graph based adaptation of the LP simplex method that exploits the network structure to achieve very good efficiency. The upper and lower bounds on the r variables provide an initial feasible spanning tree. This tree has two levels only, with the host node as the root and all other nodes as leaves. To prevent cycling we construct the initial basis to be strongly feasible by using the appropriate bound (upper or lower) to connect a node to the root (host node). It is easy to maintain strongly feasible trees during the simplex operations, and details are given in [BJS77].

Using the first eligible arc pivot rule with a wraparound arc list from [AMO93] (page 417) gave us significant improvements in the run time. The dual variables (r variables) are directly available from the min cost flow solution. We could

solve problems with more than 57,000 variables and 3.6 million constraints in about 2.5 minutes.

5.2.6 Experimental Results

Table 5.1 presents experimental results on minimum area retiming using Minaret [MS97a, MS98b]. For each circuit, the number of gates $|G|$, the percentage improvement in the Area A_{red}, the percentage reduction in the size of the LP in terms of variables V_{red}, and constraints C_{red} is presented. Also presented is the breakup of the CPU time (in seconds), in terms of the time spent in using ASTRA to arrive at the bounds for the r variables (T_b), the time spent in generating the LP of (5.13) (T_c) and the time needed to solve this LP by the network simplex method (T_s), and the total execution time (T_t).

Table 5.1. Results of Minarea Retiming using Minaret

Circuit	$\|G\|$	A_{red}	V_{red}	C_{red}	T_b	T_c	T_s	T_t
s13207.1	8004	29.1%	20.5%	92.1%	0.6s	9s	1s	10s
s15850.1	9843	8.1%	22.9%	96.3%	1.0s	36s	1s	39s
s35932	16420	0%	52.5%	86.4%	1.1s	5s	2s	8s
s38584.1	19595	0%	12.4%	99.2%	2.2s	54s	8s	1m:5s
s38417	21504	15.2%	0.7%	7.5%	2.9s	92s	52s	2m:27s
myex1	35711	64.2%	8.1%	97.7%	5.9s	401s	128s	8m:55s
myex2	40993	75.8%	1.6%	67.8%	8.2s	267s	193s	7m:48s
myex3	50532	70.8%	4.6%	85.2%	12.9s	600s	272s	14m:45s

Minaret has relatively small memory requirements since a significant number of constraints are unnecessary, and therefore, do not need to be stored. While running the mincost flow problem with the constraint set from previous apporoaches (e.g., [SR94]), it was found that the constraints numbered in the millions, and it was found that the memory requirement becomes a bottleneck for the mincost flow solver. This was not a problem for the constraint set generated on any of the above circuits. Therefore, the reduction in the number of constraints serves to reduce both the problem generation and the problem solution time.

5.3 MINAREA RETIMING OF LEVEL-CLOCKED CIRCUITS

The minarea retiming LP for level-clocked circuits is similar to the LP for edge-triggered circuits given in (5.6). Unfortunately, under general clock schedules with unequal phases, it must be modeled as a general integer linear program of the type given in [LE91], while restricting the clock scheme to a symmetric

multi-phase clock enables us to model the minarea retiming problem as an efficiently solvable LP (dual of min-cost flow problem) [PR93b].

Like the minarea LP of (5.6), the minarea LP for level-clocked circuits also contains circuit constraints, period constraints and mirror constraints. The circuit and mirror constraints are defined in the same way as in Section 5.1.2. Since the timing constraints in level-clocked circuits are different from those in edge-triggered circuit, the period constraints are derived as follows.

For a k phase symmetric clock we have $T_{\phi_i} = T_\phi \; \forall i = 1, \cdots, k$ and $P = \frac{T_\phi}{k}$. For a level-clocked circuit to be properly clocked the delay on any path should be less than the time available, i.e.,

$$d(p) \leq \quad (w_r(p) + 1) \cdot P + T_\phi \quad \forall\, p : u \rightsquigarrow v \tag{5.14}$$

This constraint can be rewritten after substitution of equation (4.1) as

$$r(u) - r(v) \quad \leq w(p) - \tfrac{d(p)}{P} + 1 + \tfrac{T_\phi}{P} \quad \forall\, p : u \rightsquigarrow v \tag{5.15}$$

Clearly if there are multiple paths from u to v, only the tightest constraint, i.e., the one with the minimum right hand side, is irredundant[2]. We denote the minimum value of $\left[w(p) - \frac{d(p)}{P}\right]$ over all paths from u to v by $\delta(u, v)$, i.e.,

$$\delta(u, v) = \min_{\forall p : u \rightsquigarrow v} \left(w(p) - \frac{d(p)}{P} \right) \tag{5.16}$$

Let us define $\Delta(u, v)$ as

$$\Delta(u, v) = \left\lfloor \delta(u, v) + \frac{T_\phi}{P} + 1 \right\rfloor \tag{5.17}$$

Since the retiming variables $r(u)$ and $r(v)$ are integers, we can rewrite relation (5.15) as the period constraints

$$r(u) - r(v) \quad \leq \Delta(u, v) \quad \forall\, p : u \rightsquigarrow v \tag{5.18}$$

We now have the constrained minarea retiming LP as:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V \cup M} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] & \\ \text{subject to} & r(u) - r(v) \leq c_{uv} \quad \forall (u, v) \in C & \\ & -\infty \leq r(u) \leq \infty \quad \forall u \in (V \cup M) & \end{array} \tag{5.19}$$

where $C = C_p \cup C_c \cup C_m$ is the constraint set of the LP, and includes the period constraint set (C_p), the circuit constraint set (C_c) and the mirror constraint set (C_m). A constraint (i, j) in the constraint set C is

$$\begin{array}{lrcl} & r(i) - r(j) & \leq & c_{ij} \quad \forall (i, j) \in C \\ \text{where} & c_{ij} & = & w(e_{ij}) \quad \forall (i, j) \in C_c, \text{ i.e., } e_{ij} \in E \\ & c_{ij} & = & \Delta(i, j) \quad \forall (i, j) \in C_p, \text{ i.e., } i \in V \text{ and } j \in V \\ & c_{ij} & = & w(\max_i) - w(e_{jm_i}) \quad \forall (i, j) \in C_m, \\ & & & \qquad \text{i.e., } m_i \in M \text{ and } \forall j \in FO(i) \end{array} \tag{5.20}$$

5.3.1 *Efficient Minarea Retiming using Minaret-L*

To reduce the space requirements of minarea LP, it is imperative that we have some techniques to prune the constraints as they are generated, rather than after all the constraints have been generated. In this section we will take advantage of the relationship between retiming and GDT presented in Section 4.5.4 to reduce the size of the LP by using bounds on the r variables.

As in Section 5.2.1, the ALAP and ASAP retimings described in Section 4.4.3 (as applied to level-clocked circuits) give us bounds on the r variables, of the form

$$L_u \leq r(u) \leq U_u \quad \forall v \in V \tag{5.21}$$

These bounds give us a reduced variable set $V' \subseteq V$ as

$$V' = \{v \in V | U_v \neq L_v\} \tag{5.22}$$

We use Theorem 5.1 to obtain bounds on the mirror variables and thus obtain the reduced mirror variable set $M' = \{v \in M | U_v \neq L_v\}$.

Figure 5.5. An example circuit.

Figure 5.6. Retiming for clock period optimization.

Example: For the circuit in Figure 5.5, the ASAP location for the latch $L1$ is at the output of gate $G1$ as shown in Figure 5.7. The number of latches moved across each gate in arriving at this ASAP location, and hence, the upper bounds are: $U_{G1} = 0$, $U_{G2} = 1$, $U_{G3} = 1$, and $U_{G4} = 0$. The ALAP location of latch $L1$, as shown in Figure 5.6, is at the output of gate $G2$. The number of latches moved across each gate in arriving at this ALAP location, and hence, the lower

Figure 5.7. Alternate retiming for clock period optimization.

bounds are: $L_{G1} = 0$, $L_{G2} = 0$, $L_{G3} = 1$, and $L_{G4} = 0$, i.e.

$$\begin{aligned} V &= \{G1, G2, G3, G4\} \\ V' &= \{G2\} \end{aligned}$$

The presence of the bounds obtained in relation (5.21) makes a large number of constraints redundant, since these constraints are implied by the bounds. We now present a rule to identify these redundant constraints.

Rule 5.2 *Any constraint (i, j) of the form $r(i) - r(j) \leq c_{ij}$ is redundant in the presence of the bounds of relation (5.21) and can be dropped if $U_i - L_j \leq c_{ij}$.*

Proof : It can be seen from the bounds on $r(i)$ and $r(j)$ in relation (5.21) that

$$r(i) - r(j) \leq U_i - L_j$$

Therefore, if $U_i - L_j \leq c_{ij}$ then $r(i) - r(j) \leq c_{ij}$ must also be true. Thus, any constraint (i, j) is redundant and can be dropped if $U_i - L_j \leq c_{ij}$.

To obtain the reduced constraint set $C' \subseteq C$ we accept only those constraints from C that are not dropped by the application of Rule 5.2[3]. Thus

$$C' = \{(i, j) \in C | U_i - L_j > c_{ij}\} \tag{5.23}$$

Thus, the original LP in (5.19) is reduced to a much smaller yet equivalent LP given below

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V' \cup M'} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] \\ \text{subject to} \quad & r(u) - r(v) \leq c_{uv} \quad \forall (u, v) \in C' \\ & L_u \leq r(u) \leq U_u \quad \forall u \in V' \end{aligned} \tag{5.24}$$

5.3.2 Generating the Reduced Linear Program

A major portion of the effort in retiming a level-clocked circuit for minimum area is spent in generating the period constraints set C'_p. We now describe efficient techniques for generating this set C'_p. The generation of period constraints requires computation of δ values for all-pairs of gates in the circuit. However, if the ALAP retiming satisfies the target clock period[4], then we need to compute δ values only from flexible gates, as stated in the following theorem.

Theorem 5.3 *If the ALAP retiming satisfies the target clock period, any period constraint from a fixed node a (i.e., $U_a = L_a$) is redundant in the presence of the bounds of relation (5.21) and need not be generated.*

Proof : Since ALAP positions are feasible solutions the following holds for all constraints in C_p.

$$L_i - L_j \leq \Delta(i,j) \ \forall \ (i,j) \in C_p \tag{5.25}$$

Consider any period constraint $(a,b) \in C_p$ from a fixed gate a, to any other gate b of the form $r(a) - r(b) \leq \Delta(a,b)$. By relation (5.25) $L_a - L_b \leq \Delta(a,b)$, and by relation (5.21) $r(a) - r(b) \leq U_a - L_b$. Because gate a is fixed $U_a = L_a$, and therefore, $r(a) - r(b) \leq L_a - L_b \leq \Delta(a,b)$. Thus, the constraint (a,b) is redundant and can be dropped. Since this is true for any period constraint from gate a, we do not need to generate any period constraint from a fixed gate, as they will all be redundant.

5.3.3 Computing the δ Values

The δ values can be obtained by re-weighting each edge e_{ij} with $w'(e_{ij}) = \left\lceil w(e_{ij}) - \frac{d(i)}{P} \right\rceil$ and computing all-pair shortest paths. We use Johnson's algorithm [CLR90] which has $O(|V|)$ memory requirement, since $O(|V|^2)$ memory is not practical for large circuits with tens of thousand gates. Johnson's algorithm first re-weights all edges to ensure nonnegative edge weights. The shortest paths between all pair of gates are then computed by running Dijkstra's algorithm for each gate as source.

Let us consider a particular run of Dijkstra's algorithm with gate a as the source, and let b be a gate to which the shortest path $\delta(a,b)$ has been obtained. Let c be any other gate in the circuit, reachable from gate a.

$$\begin{aligned} \text{By definition,} \quad r(a) - r(b) &\leq U_a - L_b \\ \text{If} \quad U_a - L_b &\leq \delta(a,b), \\ \text{then} \quad r(a) - r(b) &\leq \delta(a,b). \end{aligned} \tag{5.26}$$

From relation (5.15) and (5.16)

$$r(b) - r(c) \leq \delta(b,c) + \frac{T_\phi}{P} + 1,$$

which when combined with relation (5.26) gives

$$r(a) - r(c) \leq \delta(a,b) + \delta(b,c) + \frac{T_\phi}{P} + 1 \tag{5.27}$$

If the shortest path from gate a to gate c does not go through gate b, then $\delta(a,b) + \delta(b,c) \geq \delta(a,c)$ and we do not need to process the fanouts of gate b to obtain $\delta(a,c)$. On the other hand, if the shortest path from gate a to gate c is indeed through gate b then $\delta(a,b) + \delta(b,c) = \delta(a,c)$ and relation (5.27) is same as the period constraint $r(a) - r(c) \leq \Delta(a,c)$. If $U_a - L_b \leq \delta(a,b)$ then this period constraint is redundant. In either case we need not process the fanouts of gate b. Since this is true for any c, reachable from gate a, and we are interested only in gates reachable from gate a, we get the following rule.

Rule 5.4 *If during the shortest path calculations from source a using the Dijkstra's algorithm, for any gate b we have $U_a - L_b \leq \delta(a,b)$, we do not need to process the fanouts of gate b.*

We take advantage of the bounds on r variables to speed up the computations, by applying Theorem 5.3 to compute δ values only from the flexible gates, and using Rule 5.4 to reduce the computation for the δ values actually computed. We found that this significantly improved the time taken to generate the period constraints.

5.3.4 Reusing δ Computations

We now describe how to reuse some of the computations performed in obtaining the δ values to further speed up the generation of period constraints. The idea is motivated by the fact that in most practical circuits a significant percentage of gates are single-fanout gates. Consider one such single-fanout gate a with fanout b. For the gate a, the shortest paths to all other gates must be via gate b, which implies that $\delta(a,c) = w'(e_{ab}) + \delta(b,c)$. Therefore, we can obtain the shortest paths from gate a by simply adding $w'(e_{a,b})$ to the shortest paths from gate b. Thus, if we somehow ensure that shortest paths from gate b are obtained just before those from gate a, we will save one complete execution of Dijkstra's algorithm for gate a as source. We call this approach "chaining" and the set of gates for which only one set of δ computations is performed as "chains". We now define a simple chaining technique that stores δ values from only one source, called "1-chaining".

For 1-chaining a graph $G(V,E)$ we preprocess it to get a set of chains $\Omega = \{\omega^1, \omega^2 \ldots \omega^{|\Omega|}\}$ such that every vertex in the graph is included in exactly one chain, i.e.,

$$\omega^i \cap \omega^j = \emptyset \;\; \forall i \neq j \text{ and } \omega^1 \cup \omega^2 \cup \ldots \omega^{|\Omega|} = V$$

Each chain ω^i itself is an ordered list (of size $|\omega^i|$) of vertices in the graph, i.e., $\omega^i =< \omega^i_1, \omega^i_2 \ldots \omega^i_{|\omega^i|} >$. Thus, ω^i_j is the jth gate in the ith chain. The first gate ω^i_1 in a chain ω^i is called its head, and all gates in a chain except the head must have only one fanout, i.e., $|FO(\omega^i_j)| = 1 \; \forall i$ and $\forall \; j > 1$. The gates in a

chain are ordered such that any fanin of a gate appears after it in the chain, i.e.,

$$e_{\omega^i_{j+1}\omega^i_j} \in E \ \ \forall i \text{ and } 1 \leq j < |\omega^i|$$

We only need to obtain the δ values from the gates that are at the head of a chain, i.e., we only need to compute the values $\delta(\omega^i_1, u)$ $\forall u \in V$ and $1 \leq i \leq |\Omega|$. For all other gates the δ values can be obtained by adding the re-weighted edge weight to the δ values from its fanouts, i.e.,

$$\delta(\omega^i_{j+1}, u) = \ \ \delta(\omega^i_j, u) + w'(e_{\omega^i_{j+1}\omega^i_j}) \quad \forall u \in V \text{ and } 1 \leq j < |\omega^i| \text{ and } 1 \leq i \leq |\Omega|$$

Notice that for a gate that is not at the head of any chain we obtain the δ values by a simple addition, instead of a full run of Dijkstra's algorithm. Since we need to run Dijkstra's algorithm only for gates at the head of a chain we need to perform only $|\Omega|$ single-source shortest path computations ($|\Omega| \leq |V|$). Thus, our goal in obtaining these chain is to reduce their number, i.e., minimize $|\Omega|$. In the worst case where every gate in the circuit has more than one fanout, each chain contains only one gate, and $|\Omega| = |V|$, then we need to perform the complete Johnson's algorithm. The idea of chaining can be further generalized. Conceptually there are two extremes of chaining:

- No information about the δ values is stored, e.g., repeated single-source shortest paths algorithms like Johnson's algorithm with $O(|V|)$ memory requirements.
- All information about the δ values is stored, e.g., direct all-pairs shortest path algorithms like Floyd Warshall algorithm [CLR90] with $O(|V|^2)$ memory requirements.

The 1-chaining described above is an intermediate method in which we save δ values from only one source. Conceptually we can define k-chaining as a method that stores δ values from k appropriately chosen sources. This k-chaining in general will require $O(k \cdot |V|)$ memory and careful selection of the k sources, and is not consideredl here.

We now describe a simple preprocessing technique to obtain 1-chaining. This preprocessing step first assigns a label to each gate which indicates the number of gates that can reuse its δ computation. All the gates have their labels initialized to 0. These labels are updated by a relaxation step, in which every single-fanout gate relaxes the label of its fanout gate by increasing it (to its own label plus one). Since multiple-fanout gates cannot reuse δ computations of their fanout gates in 1-chaining, they do not relax the labels of their fanout gates. This relaxation process is finite because we cannot have cycles containing only single-fanout gates. The chains are then formed by initializing a queue with all multiple-fanout gates. Every gate in this queue starts a new chain. For the gate at the end of the chain, we process the fanin gates, adding the single-fanout gate with the highest label (amongst the fanins) to the chain; all other gates in the fanin are added to the queue. The fanins of the gate that

now lie at the end of the chain are processed similarly, until no more gates can be added to this chain. This procedure is repeated until the queue is empty.

We found that, on an average we could reduce the time spent in generating the period constraints by about 50% using the simple 1-chaining technique described above. The time spent in preprocessing to obtain 1-chaining is very small, making it a useful procedure even if only a small number of gates have single fanout. As a side note, Rule 5.4 must be modified for use with chaining to ensure that it holds for all gates that reuse the δ computation.

5.3.5 *Additional Constraint Pruning Techniques*

We now present some more techniques to remove redundant period constraints. Consider three gates a, b and c, such that gate b lies on the path from gate a to gate c.

If gate b is a fanin of gate c then we have

$$\begin{aligned} C1: &\quad r(a) - r(b) \leq \quad \Delta(a,b) \\ C2: &\quad r(b) - r(c) \leq \quad w(e_{bc}) \\ C3: &\quad r(a) - r(c) \leq \quad \Delta(a,c) \end{aligned}$$

If $\Delta(a,b) + w(e_{bc}) \leq \Delta(a,c)$ then constraint $C3$ is redundant and can be dropped. This leads us to the following rule

Rule 5.5 *If b and c are two gates reachable from gate a, such that gate b is a fanin of gate c and $\Delta(a,b) + w(e_{bc}) \leq \Delta(a,c)$ then the period constraint from gate a to c is redundant and can be dropped.*

Since we generate the period constraints from one gate (say gate a) at a time, both $\Delta(a,b)$ and $\Delta(a,c)$ are available in the same iteration. Further because gate b is a fanin of gate c the value $w(e_{bc})$ is available directly from the circuit graph. Therefore, Rule 5.5 can be efficiently applied to drop redundant period constraints as they are generated. This reduces the space (memory) requirement of the period constraints.

If gate b is a fanout of gate a then we have

$$\begin{aligned} C4: &\quad r(a) - r(b) \leq \quad w(e_{ab}) \\ C5: &\quad r(b) - r(c) \leq \quad \Delta(b,c) \\ C6: &\quad r(a) - r(c) \leq \quad \Delta(a,c) \end{aligned}$$

If $w(e_{ab}) + \Delta(b,c) \leq \Delta(a,c)$ then constraint $C6$ is redundant and can be dropped. This leads us to the following rule.

Rule 5.6 *If gate b is a fanout of gate a and gate c is some gate reachable from gate b, then if $w(e_{ab}) + \Delta(b,c) \leq \Delta(a,c)$ then the period constraint from gate a to c is redundant and can be dropped.*

To apply Rule 5.6 we require the value of $\Delta(b,c)$ and $\Delta(a,c)$. Since we generate period constraints from one gate at a time, the period constraints to a gate (c)

from two different sources (gate a and b) cannot be efficiently accessed. Thus, it would appear that Rule 5.6 cannot be efficiently applied. However, because of the reuse of δ computation described in Section 5.3.4, Rule 5.6 can be efficiently applied if gate a has only one fanout (gate b). This is possible because $\Delta(a, c)$ is derived from $\Delta(b, c)$, and hence, both are available when the period constraint from a to c is being generated. Thus, we can drop redundant period constraints from gate a as they are generated.

Rule 5.2 is valid only in presence of the bounds and it prunes the constraint sets because the information in these bounds make some constraints redundant. Rule 5.5 and Rule 5.6 on the other hand do not depend on bounds and, they prune the period constraint set because of the discrete nature of the Δ values. Rule 5.5 and Rule 5.6, can be generalized to include implication by more than two constraints; these generalized rules will, however, be computationally expensive to apply.

5.3.6 Experimental Results

Table 5.2. Results of Minarea Retiming using Minaret-L for 2-Phase Circuits

Circuit	$\|G\|$	A_{red}	V_{red}	C_{red}	T_b	T_c	T_s	T_t
s13207.1	8,004	68.6%	16.6%	99.68%	1s	18s	1.45s	20.85s
s15850.1	9,843	40.0%	20.5%	99.8%	2s	46s	3s	51s
s35932	16,420	25.3%	21.9%	99.9%	2s	36s	7s	45s
s38584.1	19,595	61.5%	0.7%	99.9%	4s	100s	25s	129s
s38417	21,504	60.3%	1.3%	97.1%	6s	307s	212s	525s
myex1	35,711	52.1%	21.8%	98.4%	10s	1438s	405s	1852s
myex2	40,993	63.3%	1.0%	97.2%	13s	2390s	1134s	3542s
myex3	50532	53.2%	0.6%	98.9%	19s	3194s	794s	4007s

Table 5.2 presents experimental results on minimum area retiming of two phase circuits using Minaret. For each circuit, the number of gates $|G|$, the percentage improvement in the Area A_{red}, the percentage reduction in the size of the LP in terms of variables V_{red}, and constraints C_{red} is presented. Also presented are the breakup of the CPU time (in seconds) in terms of the time spent in using ASTRA-L to arrive at the bounds for the r variables (T_b), the time spent in generating the LP of (5.13) (T_g) and the time needed to solve this LP by the network simplex method (T_s), and the total execution time (T_t).

Notes

1. This model is valid only for unconditional sharing of FF's. In Section 6.3.3, we will present a model for conditional sharing of FF's.

2. This approach is similar to the relevant path labeling approach of [vVA+94], but unlike their results, is found to be highly efficient since it is used in conjunction with the bounds derived from the solution to the clock skew optimization problem.

3. Some additional techniques to prune redundant period constraints are presented later in Section 5.3.5.

4. Notice that if the maximum gate delay in the circuit is more than the active period T_ϕ, it is possible for ALAP retiming to violate the target clock period even if the target clock period is feasible by retiming alone. This is because the method of finding ALAP retiming converts a (continuous) skew optimization solution to a (discrete) retiming solution. This, however, does not imply that these ALAP bounds are wrong, but merely that they are not tight enough. In level clocked circuits, due to the flexibility offered by the transparent nature of latches it is very unlikely that the ALAP retiming will violate the target clock period, and in our experiments no ALAP retiming violated the target clock period.

6 RETIMING CONTROL LOGIC

A major problem associated with the application of retiming to control logic is the preservation of the initial (reset) state of a circuit, which is determined by the initial values of the registers in the circuits. For control logic, it is necessary to find an equivalent initial state for the retimed circuit since the preservation of this equivalence is imperative to maintain the circuit behavior.

Figure 6.1. (a) Original circuit. (b) Retimed circuit.

Before proceeding to describe techniques that achieve this objective, it would be appropriate to define the notion of equivalence. An initial state in the retimed circuit is equivalent to that in the original circuit if, for any input sequence applied to both the circuits, with the original circuit started in the

N. Maheshwari et al., *Timing Analysis and Optimization of Sequential Circuits*
© Kluwer Academic Publishers 1999

initial state and the retimed circuit started in the equivalent initial state, the same sequence of outputs is produced.

It is not always possible to find an equivalent initial state for the retimed circuit without modifying it. For example, consider the circuit in Figure 6.1(a) taken from [TB93]. If the initial value stored in flip-flops (FF's) A and B are 1 and 0, respectively, then the retimed circuit, shown in Figure 6.1(b), cannot be initialized to have the same behavior as the original circuit since an equivalent initial value of FF C in the retimed circuit cannot be found.

The phrase "retiming an initial state" refers to the problem of finding a retiming with an initial state such that the original circuit and the retimed circuit have the same behavior when started in their respective initial states. In the presence of such initial state constraints, the minperiod and minarea retiming problems are referred to as *minperiod initial state retiming* and *minarea initial state retiming*, respectively, and can be defined as follows:

Minperiod initial state retiming

Given

A circuit C

An initial (reset) state s

Find

The minimum clock period P

The retimed circuit C' satisfying the clock period P

A new initial state s' such that $C(s) \equiv C'(s')$

Minarea initial state retiming

Given

A circuit C

An initial (reset) state s

A target clock period P

Find

The retimed circuit C' with minimum number of FF's

A new initial state s' such that $C(s) \equiv C'(s')$

subject to C' satisfying the clock period P

The organization of this chapter is as follows. Methods that perform minperiod initial state retiming, via the state transition graph and via reverse retiming, are described in Sections 6.1 and 6.2, respectively. Next, an approach to minarea initial state retiming is presented in Section 6.3. Finally, the use of explicit reset circuitry to maintain initial states is reviewed in Section 6.4.

6.1 MINPERIOD INITIAL STATE RETIMING VIA THE STATE TRANSITION GRAPH

Figure 6.2 shows a situation where FF's with initial states a and b are moved forward across a gate that performs the logic function f. For this type of

Figure 6.2. Forward retiming of a combinational logic node [TB93] ©1993 IEEE.

forward motion of FF's, which corresponds to a negative r value for the combinational block f, an equivalent initial state is given by $f(a, b)$. This implies that for negative retimings, an equivalent initial state can always be found. Thus, one way to ensure the equivalence of initial states during retiming is to permit only forward retiming moves.

This concept was used by Touati and Brayton in [TB93] to compute initial states of retimed circuits. In their approach, FF's may be removed from all of the primary inputs and inserted at all of the primary outputs, corresponding to a motion across the host node (defined in Section 4.3). The problem is then reduced to determining the initial values for the FF's inserted at the primary inputs.

If k FF's are inserted at each primary input, then the values that are applied at the input will reach the corresponding logic k time units later. The approach utilizes these k time units to return the circuit to the initial state by identifying a sequence of k transitions in the state diagram of the circuit. This sequence is chosen in such a way that it will bring the machine into the reset state, irrespective of the starting state to which this sequence was applied. In other words, the k sets of input values associated with these transitions are chosen as the reset states of the corresponding FF's in the retimed circuit, so that after the first k time units, the circuit is guaranteed to return to its reset state.

The desired sequence exists if the initial state is reachable from any other state in k transitions; otherwise the circuit has to be modified by incorporating additional logic to obtain such a sequence. The circuit is modified in a three-step approach. First, an external input is added to the circuit. This input can be viewed as an explicit, synchronous reset signal, and it expands the input space, making it possible to create additional state transitions that will force the initial state to be reachable from itself. The circuit is then retimed, adding the k FF's to every primary input including this extra input. Finally the external input is replaced by a zero constant node, and this constant is propagated whenever possible. This replacement of the input by a constant value amounts to setting the reset signal to zero after the initialization phase is done. The initial values on the k FF's added to the extra input produce the reset signal on the added line.

Example [TB93]. Consider the circuit in Figure 6.3, with the initial state of 01 which is not reachable from itself, as shown in the state transition graph drawn alongside the circuit. Assuming unit delay gates, the clock period can

be reduced to 2 units by retiming the two FF's across the inverter, as indicated in the figure. The problem, however, is that an equivalent initial state of this new FF after retiming does not exist.

To apply the method presented in the preceding discussion, a reset input and reset logic are added to the circuit to take it from state 00 (arbitrarily chosen among all states that have a minimal Hamming distance from the initial state) to the original initial state 01. The effect of this transformation on the circuit and the state transition graph is shown in Figure 6.4.

Figure 6.3. Example of an impossible retiming [TB93] ©1993 IEEE.

Figure 6.4. Adding minimal reset logic to make the retiming possible [TB93] ©1993 IEEE.

The retiming intended for the original circuit in Figure 6.3, when applied to the transformed circuit, is shown in Figure 6.5. This backward retiming is to be obtained by a series of forward moves as shown in Figure 6.6. Before moving any of the FF's, the circuit is initialized to a state 11 that can lead to the original initial state of 01 after one transition. This is shown as Step 1 in Figure 6.6. In Step 2, the FF at the primary output is retimed across the host vertex to the primary inputs. Since an input combination of 01 will take the circuit from the current (modified) state, 11, to the initial state, 01, these FF's

are initialized to 01. In Step 3, the FF's are moved forward until they reach the desired positions in the final retimed circuit, and the initial state of each FF is calculated in accordance with Figure 6.2. Finally, in Step 4, thei reset input is replaced by a constant noncontrolling value to obtain the final retimed circuit. Notice that the FF on the reset line is intialized to 1 and serves the purpose of resetting the circuit in the first clock cycle, after which the reset line becomes a constant zero and eliminates the effect of the reset circuitry on the functionality of the circuit.

If k FF's are moved across the host node, then the reset line may contain up to k FF's on it[1], and thus will bring the circuit to the required initial state in k clock cycles. The external enviornment of the circuit is not modified, since this reset sequence of size k is stored in the FF's on the reset line and is not explicitly applied by the environment.

Figure 6.5. Retiming of the modified circuit [TB93] ©1993 IEEE.

Figure 6.6. Determining the retimed circuit that preserves the initial state [TB93] ©1993 IEEE.

Results presented in [TB93] indicate that adding a moderate amount of extra logic enabled the minperiod initial state retiming of several circuits using only forward retiming moves, and replacing the original initial state by an equivalent initial state helped reduce the amount of the extra logic in some cases. However, the results are reported on a few small circuits, and it is not clear how much extra logic may be required for large practical circuits.

6.2 MINPERIOD INITIAL STATE RETIMING VIA REVERSE RETIMING

Permitting only forward moves is too restrictive; while it is true that all backward moves do not have equivalent initial states, there are many such moves for which an equivalence exists. For example, if in the circuit in Figure 6.1,

both FF A and B had the same initial value, then the backward move across gate G1 would have been possible while maintaining an equivalent initial state. Hence, another retimed circuit that requires some backward moves may exist, with an equivalent initial state that may be found without any modifications to the circuit. This is in contrast with the results of [TB93], which would require circuit modifications for any backward move. The procedure of reverse retiming [SSE95, ESS96] finds this retiming by disallowing FF moves across the primary outputs and by minimizing their backward motion.

The basic difficulty in backward moves is in the process of finding a mapping from the initial state in the current circuit to one in the retimed circuit with the backward moves. Finding this mapping is an NP-hard problem, as it is similar to the phase of justification in the process of automatic test pattern generation (ATPG) [ABF90]. Thus, a good criterion is to minimize the number of registers that move backward across each gate. To achieve this purpose, reverse retiming modifies the FEAS algorithm of [LS91] (see Chapter 4). Unlike the FEAS algorithm, which moves FF's backwards from the outputs to the inputs as far as possible, reverse retiming moves FF's forward from the inputs towards the outputs, as far as possible, as it attempts to satisfy the clock period. The retiming procedure differs in that reverse retiming calculates the required time $t_r(u)$ for every node instead of calculating the arrival time, $\Delta(u)$. For any node u, the required time is defined as the difference between the smallest required time of its successors and the propagation delay of the node itself, i.e., $t_r(u) = \min_{(u,v)\in E} t_r(v) - d(u)$; the required time of the host and each registers is the required clock period P. The following pseudo code describes the reverse retiming algorithm.

```
Algorithm REVERSE-RETIME
Given a synchronous circuit G = <V,E,d,w>, and a desired clock
period P, return a reverse retiming r of G such that the clock
period of the retimed circuit Φ(G_r) ≤ P.
{
  1.  For each vertex v ∈ V, set r(v) ← 0.
  2.  Repeat the following |V| − 1 times
      2.1 Compute graph G_r with existing values of r.
      2.2 Determine t_r(v) for each vertex v ∈ V.
      2.3 For each v such that t_r(v) < 0, set r(v) ← r(v) − 1.
  3.  Run Algorithm CP on the circuit G_r.  If Φ(G_r) > P, then
      no feasible retiming exists.  Otherwise, r is the desired
      retiming.
}
```

It can be shown that if the clock period P is feasible and all nodes are reachable from the host, then reverse retiming finds the retiming for which $R^* = \max_{v\in V} r(v) - r(host)$ is the minimum. Thus, reverse retiming finds a retiming in which the maximum number of backward moves, calculated over all nodes, is minimized. The obtained retiming is then normalized to make $r(host) =$

0, so that no FF's are required to move across primary inputs/outputs, and the FF's are moved to satisfy these normalized lags. Each time an FF is moved backwards across a node, a mapping for the initial state is found. Since `REVERSE-RETIME` did not take the initial (reset) state into account, it is possible that a mapping may not be found for some backward moves. If the ith backward move is not feasible across a gate v, then its lag is bounded to $i-1$, i.e., $r(v) \leq i-1$. This is achieved by adding an edge of weight $i-1$ from the gate v to the host vertex. The process of reverse retiming, moving FF's and bounding lags is repeated until one of the two following conditions is true:
(a) no more bounding is required, in which case the required retiming has been obtained, or
(b) `REVERSE-RETIME` indicates no feasible retiming, in which case the circuit cannot be retimed for the given clock period P with an equivalent initial state.

It is shown in [ESS96] that this method may be applied to perform minperiod initial state retiming on large circuits. Unlike [TB93], the method does not require the computationally expensive step of state machine extraction and modification to find the initial value to be inserted at the primary inputs. Note that instead of taking this approach and minimizing R^*, which is a heuristic way of minimizing the number of justifications, an exact approach would find the ALAP locations of the FF's, which, by definition, would minimize the justification effort.

6.3 MINAREA INITIAL STATE RETIMING

6.3.1 Overview

For digital circuit design, the most useful objective function is that of constrained minarea retiming. However, none of the methods described so far for initial state retiming considers the area penalty during retiming. The standard minarea algorithms, e.g., [LS91, SR94] or Minaret [MS98b] pay no regard to the initial states, and while they have applications in datapaths where the initial state is unimportant, they cannot be used to optimize control logic since an equivalent initial state is not guaranteed to exist in the retimed circuit.

An approach to minarea initial state retiming [MS97b] is presented in this section. It also has applications to minarea retiming of circuits containing different types of memory elements that cannot be shared with each other, e.g., load enable registers.

Ensuring Equivalent Initial States. The requirement of initial state equivalence imposes restrictions in addition to those in the conventional minarea retiming problem. Thus, the number of FF's obtained by the conventional minarea retiming is a lower bound on the number of FF's obtainable by a minarea equivalent state retiming. This lower bound will be denoted by Γ.

However, it may or may not be possible to achieve this lower bound, depending on the initial state. To illustrate this, consider the circuit shown in Figure 6.7(a), and the conventional minarea retiming (without regard to initial

Figure 6.7. Conventional minarea retiming.

Figure 6.8. Example of variation in the number of FF's with the initial state.

Figure 6.9. Another example of variation in the number of FF's with the initial state.

states) shown in Figure 6.7(b), requiring only one FF. If the initial values for FF A through D are as shown, then the retimed circuit in Figure 6.7(b) does not have an equivalent initial state. Further, there is no possible retiming with one FF that has an equivalent initial state. However, if the initial value for each FF in the circuit is 0, then the retiming obtained in Figure 6.7(b) is feasible in terms of initial state equivalence. Therefore, depending on the initial state, it may or may not be possible to achieve the lower bound Γ.

Additionally, the optimal number of FF's with equivalent initial state depends on the initial state of the original circuit. As shown in Figure 6.8, if initial value of FF's are {A=0, B=0, C=1, D=0} then the minimum number of

FF's possible is 3. It can readily be verified that if the initial value of FF's are {A=0, B=0, C=1, D=1} then the minimum number of FF's is two, as shown in Figure 6.9.

Figure 6.10. An example circuit where lower bound Γ is achievable.

Figure 6.11. A minarea retiming without an equivalent initial state.

Figure 6.12. A minarea retiming with an equivalent initial state.

As an example, consider the circuit in Figure 6.10 and its retimed version in Figure 6.11. If the initial values of FF's are {A=0, B=1, C=1}, then there is no equivalent set of initial values for the retimed FF's D and E in Figure 6.11. Figure 6.12 presents an alternative retiming of the original circuit requiring the same number of FF's, but in this case an initial value of 0 at FF I and 1 at FF H is equivalent to the original initial state. Note that performing conventional minarea retiming without regard to initial states could result in either of these retimings. Therefore, even if the lower bound Γ is achievable, there is no guarantee that an algorithm that ignores the initial state will find it.

The fundamental reason for being unable to achieve an equivalent initial state after retiming is the presence of conflicting values at the fanouts of a gate. For example, in Figure 6.10 if we move the FF's in the original circuit backwards to obtain the retiming in Figure 6.11, then we get FF's at the output of gate G1 with values 1 and 0 that cannot be moved to the input of gate G1 while maintaining an equivalent initial state. This situation is referred to as a *conflict*, and it is the reason why the lower bound Γ is not always achievable.

In this approach, the minarea equivalent state retiming is found for a given circuit topology, and a given set of initial values. To ensure the existence of an equivalent initial state in the retimed circuit, only those retiming moves that have an equivalent initial state are allowed. This includes all forward retimings except across host node ($r(H) = 0$), and backward retiming moves with equivalent initial states. As in Section 6.2, retiming across the host vertex is forbidden because it requires a sequence of initial values for the primary inputs to be obtained from the state transition diagram extracted from the circuit and, in addition, could require modifications to the original circuit.

Bounds are placed on the retiming variables to allow backward motions of FF's only if an equivalent initial value exits. Therefore, any retiming thus obtained is guaranteed to have an equivalent initial state. There may be multiple sets of these bounds, and each of these must be explored to obtain an optimal minarea initial state retiming. The number of FF's obtained by standard minarea retiming can be used as a lower bound to prune this exploration.

To see how initial state retiming is performed in the presence of conflicts, consider the circuit of Figure 6.10. If no backward motion of FF's across gate G1 are allowed, then it is certain that every retiming has an equivalent initial state. Backward motion across G1 can be prohibited by forcing an upper bound of 0 on the r variable of gate G1. It can be seen that any retiming with $r(G1) \leq 0$ and $r(H) = 0$ has an initial state that is equivalent to the initial state of the original circuit.

Thus, one way to ensure that any obtained retiming has an equivalent initial state, is to update the upper bounds U_v in the LP of relation (5.13) so that conflicting FF's at the fanouts of a gate are never retimed to its inputs. This will ensure a valid equivalent state in the retimed circuit. This new upper bound on gate v that ensures a valid equivalent state is called B_v. For a retiming that has an equivalent state *and* satisfies the target clock period, it is necessary to enforce $r(v) \leq B_v$ and $r(v) \leq U_v$. Then the condition $r(v) \leq J_v$ must be satisfied, where $J_v = \min(B_v, U_v)$ is the *justification upper bound.* A set of such justification upper bounds denoted by $\Lambda^i = \{J_u^i | \ \forall u \in V\}$. Since forward retiming moves always have equivalent initial values, the lower bounds from relation (5.13) for conventional minarea retiming are still valid for minarea initial state retiming. This leads to the following modified LP

$$\begin{array}{lll} & \text{minimize } \sum_{v \in V} [(|FI(v)| - |FO(v)|) \cdot r(v)] & \qquad (6.1) \\ \text{subject to} & r(u) - r(v) \leq c_{uv} & \forall c(u,v) \in C \\ & L_u \leq r(u) \leq J_u & \forall u \in V \end{array}$$

Any solution to this LP will have an initial state that is equivalent to the initial state in the original circuit and will satisfy the target clock period. The techniques of Section 5.2 can be applied to further reduce the size of the LP in (6.1).

This section also provides a formulation that takes into account the initial value of the FF's while modeling the sharing of FF's at the output of a multi-fanout gate. The method presented here is applicable for retiming of any circuit which has more than one type of memory elements (e.g., FF's with load enables) such that memory elements of different types cannot be merged together.

6.3.2 Obtaining the Justification Bounds

The procedure for obtaining these new justification upper bounds J_v for a gate v consists of two steps: a justification step, where an equivalent initial state is found, and a bound computation step, where the bounds J_v on each gate under that equivalent initial state are calculated.

With each FF is associated a three valued (1,0,X) logic. The notion of *compatibility* is defined as follows: a logic value of 0 is compatible with both 0 and X, but logic values 0 and 1 are not compatible with each other[2]. A gate can only be retimed if it has FF's with compatible logic values at all of its fanouts. A gate is retimed in the backward direction by removing an FF from each of its fanouts, and adding one to each of its inputs. A gate is called output-ready if it has an FF on each of its fanouts and the logic value on each such FF is compatible with the values on the others. The procedure maintains a list of gates that can be retimed. A gate is taken from the list and retimed, and the list is updated. As the gates are retimed, a procedure similar to the one in Section 5.2 is used to compute he bounds. The upper bounds, J_v, are obtained by moving FF's as far backwards as possible without violating the period constraints. The count of the FF's moved across any gate gives its upper bound on the r variable of the gate.

Each time FF's are moved from the outputs of a gate to its inputs, logic values are to be assigned to the new FF's added at the inputs. These logic values must be equivalent to the original value at the output of the gate in order to obtain an initial state retiming. This assignment, in general, may not be unique and is similar to the phase of justification in the process of automatic test pattern generation [ABF90]. These output-ready gates are classified into the following two categories.

Unique Justification If there is only a unique mapping of the logic value at the output to the logic values at the inputs, then no choices need to be made. These are maintained in a unique justification queue, U. The following cases are examples of unique justifications:

- A single input gate such as inverter or buffer.
- A logic value of X at the output: in this case all inputs can be assigned a logic value of X.

- A logic value of 0 at the output of an OR (NAND) gate: in this case, all inputs are assigned to logic value 0 (1).
- A logic value of 1 at the output of an AND (NOR) gate: in this case, all inputs are set to logic value 1 (0).

Nonunique Justification If there are multiple mappings possible for the logic value at the output to the logic values at the inputs, then a decision must be made. These decisions are maintained in a decision queue, D. Since the solution to the LP in (6.1) depends on the set Λ^i which in turn depend on these decision made here, these decisions may have have to be revisited. The following cases are examples of nonunique justifications:

- A logic value of 1 at the output of an OR (NAND) gate: in this case, any one input is set to logic value 1 (0) and the rest to logic value X.
- A logic value of 0 at the output of an AND (NOR) gate: in this case, any one input is assigned the logic value 0 (1) and the rest to logic value X.

6.3.3 Searching for the Optimal Solution

Figure 6.13. Effect of justification choices on the number of FF's.

Different justification decisions may lead to a different number of FF's obtained after minarea retiming. As an example, consider the circuit shown in Figure 6.13(a), with an FF with value 0 at the output of an AND gate, leading to two possible choices shown in Figures 6.13(b) and 6.13(c). The corresponding decisions lead to retimed circuits with three and two FF's, respectively.

Under nonunique justifications, a number of different allowable justifications are possible. Let us define a set of one such possible justification as Δ^i. Each such Δ^i will give a set (one for each gate) of justification upper bounds $\Lambda^i = \{J_v^i | \forall v \in V\}$ that is used to solve the minarea LP. If the number of FF's so obtained is not equal to the minarea lower bound Γ, another set of justifications Δ^j must be obtained by backtracking, leading to a different Λ^j. This process is repeated until either the minarea lower bound Γ is achieved, or no more justifications exist. Since a complete exploration will be computationally expensive, one may halt the exploration of the search space at any time and take the best solution obtained so far.

Thus, the process of minarea initial state retiming can be given by the following pseudocode. The procedure returns the minarea retiming with an equivalent initial state.

```
Algorithm MINAREA_INITIAL_STATE_RETIMING
{
   Obtain minarea lower bound Γ
   j = 0;
   Best = ∞;
   while (true) {
      while (U ≠ ∅ OR D ≠ ∅) {
         if (U ≠ ∅) do_unique_justification
         if (D ≠ ∅) = do_decision_justification
      }
      /* This gives a justification set Δ^j, corresponding
         to a set of justification upper bounds Λ^j.  */
      Obj = lp_solve(Λ^j); /* solve LP in (6.1) */
      If(Obj == Γ) return(Obj); /* lower bound obtained */
      If(Best > Obj) Best = Obj; /* store best result */
      B = backtrack(Δ^j);
      If(B == Infeasible) return(Best);
      /* all justifications explored */
   }
}
```

The function `backtrack` changes the last decision that has a yet unexplored choice, and is similar to one used in automatic test pattern generation [ABF90]. The period constraints need be generated only once during the entire procedure since they do not depend on the justification process. This is helpful since the period constraint generation is a very computationally intensive process.

The theoretical upper bound on the number of possible justification sets Δ^i's in the worst case is $|FF| \cdot \prod_{\forall v \in V} |FI(v)|$, where $|FI(v)|$ is the number of fanins of the gate v, and $|FF|$ is the number of FF's in the original circuit. This upper bound is due to the fact that in the worst case, each FF in the circuit may move across every gate and every such move may require a decision. This bound is clearly exponential and thus the problem of finding all possible justification sets for a general circuit is NP-hard, as in the case of the justification phase of ATPG.

However, in practical circuits the number of feasible justifications will be much less than this theoretical upper bound due to the following reasons

- As shown in Section 5.2.6, the mobility of FF's is very limited in practice and hence, all FF's cannot move across <u>all</u> gates as assumed by the theoretical bound above.

- Due to conflicts at the gate fanouts the FF's may not be able to move towards the inputs of that gate, and this further restricts the mobility of the FF's.

- Some FF's moving across gates have unique justifications.
- Every time a decision is made in case of a nonunique justification, all fanins but one are assigned logic value X. This logical X moves backward through unique justification until it is forced to a 0 or 1.
- As soon as the lower bound of Γ is achieved, no more justification sets are necessary. In the experimental results for this work, it was found that in many circuits this lower bound is achieved in the first few iterations.
- Only backward moves need justification, while forward moves have a unique mapping of logic values and hence, do not add to the number of Δ^i's.

Figure 6.14. An example of a pruning technique.

The number of justification sets can be further reduced by pruning suboptimal Λ^i's. Consider the circuit in Figure 6.14 with the logic values of FF A and FF B equal to 0. Since the output of the AND gate G1 is at logic value 0 two possible mappings exist for the equivalent values at the inputs a and b. However, the choice of setting input a to X and input b to 0 is better than the choice of $a = 0$ and $b = X$. This is because in the presence of FF B with logic value 0, the X on input b will be forced to a 0, effectively setting the choice to $a = 0$ and $b = 0$. This is suboptimal to the choice of $a = X$ and $b = 0$, since X on input a can move further back than a 0.

Minarea Retiming by Reverse Retiming. Minarea initial state retiming may also be performed by extending the reverse retiming approach in Section 6.2. In this method, conventional minarea retiming is performed first. If a conflict occurs at a gate while moving the FF's to obtain this retiming, then an appropriate bound is placed on the retiming variable of this gate, and the minarea retiming problem is solved again. This procedure is repeated until no more conflicts are obtained. Thus, the final circuit has an equivalent initial state, although it may require more FF's than the conventional minarea, since the extra bounds placed on the retiming variables to ensure equivalent initial states can increase the number of FF's in the optimal solution. This alternative method can be seen as a "dual" of the approach presented in this section, since it starts from the lower bound and tries to achieve feasibility (equivalent initial state), as against the approach of starting with a feasible solution and trying to achieve optimality. However, in this approach, the initial state value on the FF's that can be possibly retimed to the fanouts of a gate is not known before

solving the minarea LP. Hence, it will not be able to model the conditional sharing of FF's, described in the next section, and this would cause its solution to be suboptimal.

Conditional FF sharing. The LP in (5.6) in Section 5.1.2 assumes that an FF can be combined with any other FF, and hence, is not applicable to minarea initial state retiming where FF's have logic values associated with them, and an FF with logic value 1 cannot be shared with one that has logic value 0. For example, consider the circuit in Figure 5.1(a) with initial state values as shown in Figure 6.15(a). With these initial values, the sharing given by the mirror vertex model of Section 5.1.2 is shown in Figure 5.1(b); this is not valid for the given initial values. Instead, the maximal sharing is as shown in Figure 6.15(b) and requires a total of six FF's. The reason is that only two FF's, shown in the dashed box in Figure 6.15(a), can be shared. The situation is further complicated by the fact that two FF's can be shared only if the FF's at their fanins (if any) are also shared. For example, consider the circuit in Figure 6.15(b), the FF's on output C and D cannot be shared, although both have an initial state value of 1, because their fanins are not shared.

Figure 6.15. Conditional register sharing at multiple fanouts.

This motivates the need for a technique to model the conditional sharing when FF's have initial values associated with them. Note that the idea of mirror vertices used by Leiserson and Saxe to model unconditional FF sharing cannot be used for the initial state retiming problem. This conditional sharing is also required for circuits having more than one type of FF's that cannot be shared with each other.

We will now present the modifications required to model the conditional sharing by a 0/1-MILP formulation. This modification is used for all gates with conflicts at their fanouts, and for all other gates the simpler model of Section 5.1.2 is used. This combination keeps the number of integer variables within a small fraction of the total number of variables. We will first present the model and then illustrate it through an example.

The justification process of Section 6.3.1 determines the logic values of all FF's that can possibly be retimed backwards to arrive at the fanout of a given gate. There is a sequence of these "possible" FF's that may arrive at every

Figure 6.16. An example of FF sharing.

fanout of every gate, and possibly be retimed across the gate, or remain at the gate output; the final retiming may contain only a subsequence of this possible sequence. The logic values of these possible FF's at the fanouts of a gate u are represented by a table T_u with $|FO(u)|$ rows as shown in Figure 6.16. Since a maximum of J_v FF's can be moved backwards across gate v to its fanins, and $w(e_{uv})$ FF's already exist between gate u and gate v, the maximum number of FF's possible between gate u and gate v is $J_v + w(e_{uv})$. Therefore, each row, $v \in FO(u)$, has $J_v + w(e_{uv})$ entries, each of which is either a 0 or a 1.

The value in the vth row and k^{th} column of the table is denoted by $T_u(v,k)$. A sharing class[3] S_i is defined as containing a set of values that can be shared, and represent the set of sharing classes for the fanouts of gate u by N_u. Two values $T_u(p,q)$ and $T_u(r,s)$ can be shared (i.e., belong to the same sharing class) only if $q = s$ and $T_u(p,i) = T_u(r,i)$ for $i = 0, \cdots, s-1$. A function $class(T_u(v,k))$ gives the index of the sharing class for entry (v,k) in table T_u, e.g., $S_{class(T_u(v,k))}$ is the sharing class containing the kth FF between gate u and its fanout v (counting from u). All the FF's in a sharing class can be shared with each other, and hence, require only one physical FF. Each sharing class S_i is represented in the MILP by a variable $\alpha_i \in \{0,1\}$. If $\alpha_i = 1$ in the optimal solution of the MILP, then the FF's of sharing class S_i share a physical FF and the sharing class S_i is said to be active. FF's moved forward across gate u to its fanouts can be shared unconditionally and will be handled later.

To ensure that the kth FF retimed across gate v activates its own sharing class variable $\alpha_{class(T_u(v,k))}$, the variable $\alpha_{class(T_u(v,k))}$ must be activated before the variable $\alpha_{class(T_u(v,k+1))}$. This is achieved by adding the following constraint

$$\alpha_{class(T_u(v,k))} \geq \alpha_{class(T_u(v,k+1))} \quad \forall v \in FO(u) \text{ and } 1 \leq k \leq J_v + w(e_{uv}) - 1$$

For every multi-fanout gate u, an integer variable $\rho_u \leq 0$ is defined to model forward retiming moves. This is required because unlike backward retiming, FF's introduced at the fanouts by forward retiming across gate u can be unconditionally shared since all of them have the same logic value. Thus, the α variables model the backward retiming and ρ models the forward retimings. Notice that this is different from the unconditional sharing model of Section 5.1.2

where the mirror variable $r(m_u)$ modeled FF's moved by both forward and backward retimings since all FF's could be unconditionally shared. Requiring the α variables to be nonnegative, $\alpha_i \geq 0$ ensures that they model only forward (positive) retiming moves, while the condition $\rho_u \leq 0$, ensures that ρ_u models only backward (negative) retiming moves.

6.3.4 Modifications in the Objective Function

To model the conditional sharing represented by the sharing classes, the objective function term for a gate u that has a conflict at its fanouts is modified to

$$(|FI'(u)| - 1) \cdot r(u) + \rho_u + \sum_{i \in N_u} \alpha_i \tag{6.2}$$

This expression counts the number of FF's that settle at the output of gate u after retiming and the significance of each term is as follows:

- The first term $(|FI'(u)| - 1) \cdot r(u)$ in (6.2) models the increase in the number of FF's when gate u is retimed by one unit, and is similar to the model in [LS91]. As earlier $FI'(u)$ is the set of fanins that have only a single output, i.e., $FI'(u) = \{v | v \in FI(u)\ AND\ |FO(v)| = 1\}$. It assumes a shared cost of one at the fanouts of gate u for any set of FF's retimed in either direction across gate u. In forward retiming, all FF's inserted at the outputs of a gate have the same logic values, and therefore, the shared cost at fanouts of gate u in forward retiming is one. Since a gate can be retimed backwards only if all FF's at its output have the same logic values, the shared cost at the outputs before retiming is also one, as modeled by this term. The bound $r(u) \leq J_u$, ensures that no set of FF's, with shared cost greater than one, is ever retimed backwards across gate u.

- The second term $\rho_u \leq 0$ is a correction factor applied to correctly model the situation in which a set of FF's moves forward across gate u and all its fanouts. It is active only during forward retiming steps, and models the number of FF's moved across the fanout junction of gate u by forward retiming. Since a negative value of ρ_u denotes forward retiming, it reflects a cost saving in the objective function.

- As mentioned earlier $\alpha_i = 1$ implies that the sharing class S_i is active, therefore, $\sum_{\forall\ i \in N_u} \alpha_i$ denotes the number of active sharing classes at the fanouts of gate u. Since each active sharing class requires one FF, the number of active sharing classes is also the number of physical FF's required at the fanouts of gate u. The minimization of the objective function will force the maximal sharing at the outputs of gate u. The first FF in a sharing class S_i that arrives at the fanout junction activates the sharing class variable α_i, incurring a cost of one in the objective function. The remaining FF's in that sharing class can then arrive without incurring any extra cost in the objective function.

6.3.5 Additional Constraints

The number of FF's between gate u and its fanout v is given by $w_r(e_{uv}) = w(e_{uv}) + r(v) - r(u)$. The cost of the FF's between u and v is given by $\sum_{k=1}^{J_v+w(e_{uv})} \alpha_{class(T_u(v,k))}$, out of which $r(u)$ FF's are removed by backward retiming across gate u and $-\rho_u$ FF's are removed by forward (negative) retiming across the fanouts. The conditional sharing of FF's is automatically modeled by the sharing of the α variables amongst the fanouts. Since the cost of FF's should be same as the number of actual FF's, we have

$$w(e_{uv}) + r(v) - r(u) = \left[\sum_{k=1}^{J_v+w(e_{uv})} \alpha_{class(T_u(v,k))} \right] - r(u) + \rho_u \;\; \forall\, v \in FO(u) \quad (6.3)$$

which can be rewritten as

$$w(e_{uv}) + r(v) = \rho_u + \sum_{k=1}^{J_v+w(e_{uv})} \alpha_{class(T_u(v,k))} \;\; \forall\, v \in FO(u) \quad (6.4)$$

Since the right hand side of equation (6.4) is being minimized in the objective function, the equality can be relaxed to the following inequality

$$w(e_{uv}) + r(v) \leq \rho_u + \sum_{k=1}^{J_v+w(e_{uv})} \alpha_{class(T_u(v,k))} \;\; \forall\, v \in FO(u) \quad (6.5)$$

An Example. Consider the circuit with the sharing classes in its table of logic values, as shown in Figure 6.16. The MILP for this circuit is

$$Minimize: -r(b) - r(c) - r(d) + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 + \rho_a$$

$$\begin{aligned} \text{subject to} \quad & r(b) \leq \alpha_1 + \alpha_2 + \alpha_3 + \rho_a \\ & r(c) \leq \alpha_1 + \alpha_4 + \rho_a \\ & r(d) \leq \alpha_5 + \alpha_6 + \rho_a \\ & \alpha_1 \geq \alpha_2 \geq \alpha_3 \\ & \alpha_1 \geq \alpha_4 \;;\; \alpha_5 \geq \alpha_6 \\ & \rho_a \leq 0 \;;\; \alpha_i \in \{0,1\} \;\; \forall i \end{aligned}$$

Backward Retiming: When the sharing for $r(a) = 0$, $r(b) = 3$, $r(c) = 1$ and $r(d) = 2$ is modeled, the optimal objective function value of the above LP is -1, which gives the correct increase in the number of FF's from the original circuit in Figure 6.17(a) to the retimed circuit in Figure 6.17(b).
Forward retiming: For $r(a) = -2$, $r(b) = -2$, $r(c) = -1$ and $r(d) = -1$, the optimal objective function value is 3, which is the increase in the number of (shared) FF's from the original circuit in Figure 6.18(a) to the retimed circuit in Figure 6.18(b). As can be seen one FF is shared for the edges e_{ac} and e_{ad}

Figure 6.17. An example of positive retiming.

even though they were not in the same sharing class. This is possible because the FF's moved forward to the outputs of gate a; hence, they all have same logic value without regard to the sharing class which are defined for backward movements. Thus, these FF's can be shared and this formulation correctly models the cost.

Figure 6.18. An example of negative retiming.

FF Sharing with Don't Cares. Since every nonunique justification decision generates an X, the actual problem of FF sharing is to find the optimal sharing between 0, 1 and X, FF's. The logic value X can be shared with either 0 or 1 and hence presents additional modeling problems. Particularly difficult are the cases when X's are followed by 0's and 1's in a sequence, since the choice of merging X with 0 or 1 influences sharing of the remaining sequence.

As an example consider the following values at the output of a gate.

0	1
1	X
1	0

The X can only be shared with 0 and, hence, should be converted to 0. However, it is not always possible to uniquely determine the possible values a X may take.

Consider another example that has the following values a gate output.

0	1
X	1
1	0

Here the X may be converted to either 0 or 1. If all 6 FF's arrive, then it is beneficial to merge X with 0; however, if only the first X and 1 arrive then X should be merged with 1. The values in the table show the FF's that can potentially arrive at this junction by reverse retiming, but not all of these FF's are required to arrive at the junction. In fact, due to the constraints and the objective function any combination of these FF's may arrive, and this makes it difficult to model the sharing in presence of X's. To avoid this problem, all X's are converted to either 0 or 1. If the X can be shared with only 0 (or a 1) then it is converted to a 0 (or a 1). To find an optimal solution both possibilities need to be explored. However, in the implementation of this method, a random assignment was used to convert X's to 0 or 1.

6.3.6 Experimental Results

An initial state minarea retiming algorithm based on the presentation in this chapter was implemented [MS97b]. Since a complete exploration of the search space is required to obtain an optimal solution, this implies that all of the possible justification sets, Δ^i's, must be generated and the corresponding LP's must be solved. Since generating the Δ^i's exhaustively is a problem of exponential complexity, it is not likely to be computationally feasible. Hence, the implemented justification algorithm makes random choices whenever there is a nonunique justification. The LP is then solved for the corresponding Λ_i. If the lower bound, Γ, is not achieved, then another random decision-based justification is performed. This process is repeated until either the lower bound is reached, or a user specified number of iterations are performed, and the best solution found is reported. Although it may seem arbitrary to use random decisions, the experimental results show that the algorithm gives good engineering solutions that are close to the (possibly unachievable) lower bound.

Other possible stopping criteria could be (a) having the best result obtained so far be within a given percentage of the lower bound, or (b) obtaining no improvements in the best solution for a given number of iterations, etc.

If there are no gates with conflicts, then the LP is the dual of a network flow mincost flow problem, and is solved using a network simplex algorithm of Section 5.2.5. If, however, the general MILP is to be solved, then the public domain MILP solver, *lp_solve* [Ber92], is used.

Table 6.1. Results of Minarea Initial State Retiming

Circuit	$\|G\|$	P	A		B		C		D		Γ
			$\|FF\|$	T_{exec}	$\|FF\|$	T_{exec}	#FF	T_{exec}	#FF	T_{exec}	
s27	11	6.0	3	0.01s	3	0.00s	3	0.01s	3	0.00s	3
s208.1	105	10.0	8	0.02s	8	0.02s	8	0.03s	8	0.03	8
s298	120	6.0	22	0.40s	22	0.44s	22	0.44s	22	0.44s	22
s382	159	7.0	23	2.59s	23	4.34s	23	4.35s	23	4.45s	23
s386	169	11.0	6	0.04s	6	0.03s	6	0.04s	6	0.03s	6
s344	161	14.0	19	1.77s	19	1.79s	19	1.77s	19	1.82s	19
s349	162	14.0	19	1.62s	19	1.62s	19	1.69s	19	1.62s	19
s526n	195	6.0	30	0.95s	30	0.95s	30	2.75s	30	0.97s	30
s510	212	11.0	7	0.12s	7	0.12s	7	012s	7	0.12s	7
s420.1	219	12.0	17	0.07s	17	0.06s	17	0.07s	17	0.06s	17
s641	380	74.0	19	0.11s	19	0.43s	19	0.44s	19	0.43s	19
s713	394	74	19	0.18s	19	0.68s	19	0.68s	19	0.69s	19
s967	395	12.0	35	28.52s	35	27.21s	35	28.05s	35	27.27s	35
s938	447	16.0	33	1.45s	33	1.53s	33	1.49s	33	1.53s	33
s1196	530	24.0	18	0.08s	18	0.07s	18	0.08s	18	0.17s	18
s1238	5.09	22.0	18	0.08s	18	0.07s	18	0.56s	18	0.08s	18
s1423	658	53.0	76	8.77s	76	9.23s	76	8.89s	76	9.31s	76
s1488	654	16.0	7	0.11s	7	0.11s	7	0.12s	7	0.11s	7
s1494	648	16.0	7	0.13s	7	0.12s	7	0.13s	7	0.12s	7
s3330	1790	14.0	110	0.58s	110	0.56s	110	0.59s	110	0.56s	110
s5378	2780	21.0	173	3:18s	173	3:19s	173	3:18s	173	3:17s	173
s9234.1	3271	38.0	134	21:18s	134	21:19s	134	23:47s	134	21:15s	134
s635	287	66.0	42	22.6s	42	22.38s	35	1.08s	39	22.5s	35
s953	396	13.0	32	32:02s	32	27:35s	32	31:30s	32	27:2s	27
s1269	570	19.0	84	0.26s	85	1:33s	85	1:31s	85	1:4s	84
s1512	781	23.0	71	1:51:19s	72	1:52s 1s	72	1:56:23s	70	1:38s	70
s3271	1573	15.0	169	16:46s	173	16:5s	170	16:40s	173	16:29s	168
prolog	1602	13.0	124	16:40s	125	16:42s	125	16:39s	125	16:29s	122
s3384	1686	27.0	168	55:42s	169	1h3:18s	169	1:2:56s	169	51:3s	167
s15850.1	9618	63.0	544	3:9:56s	540	4:2:36s	542	3:57:7s	544	3:59:5s	525

Results on the ISCAS89 [BBK89] benchmark suite are presented in Table 6.1. For each circuit, the number of gates $|G|$, the target clock period P, and the lower bound on the number of FF's obtained by Minaret, Γ, are shown. Also shown are the minimum number of FF's obtained with equivalent initial state and the execution time T_{exec} for all of the tasks, including solving the LP for all iterations, on an HP 9000/777 C110 workstation. In the absence of specified initial state values for the benchmark circuits, results for four different initial state assignments are presented. Case A has all FF's initialed to 0, Case B has all initialized to 1, and Cases C and D are for randomly generated state assignments. As can be seen from the results, for many circuits, the lower bound

is achieved in a small number of iterations for almost any initial state. In fact, in almost all of these cases the lower bound Γ is obtained in the first iteration itself. For some circuits the lower bound was not reached. This, however, does not imply that the solution obtained is not optimal since the lower bound is not always achievable with equivalent initial state. For these circuits, the best solution obtained in 50 (5 for s15850.1) iterations is reported. The execution time of this method is considerably higher than the run times for conventional minarea reported for Minaret, since this procedure may need to solve multiple general MILP's, unlike Minaret, which needs to solve only one mincost flow problem. However, note that in most cases where the lower bound is achieved, the execution times are comparable to those of Minaret. In the circuits where the lower bound, Γ, is not achieved, the solution reported by this algorithm is very close to Γ, and, therefore, corresponds to a good engineering solution. Since the optimal number of FF's in a circuit depend on the initial state of the original circuit, some variation in the number of FF's and execution time is obtained for different initial states. For `s635`, `s1269` and `s1512`, the lower bound was seen to be achieved for only some initial states.

6.4 MAINTAINING INITIAL STATES WITH EXPLICIT RESET CIRCUITRY

Figure 6.19. The reset mapping [SMB96] ©1996 IEEE.

Figure 6.20. Explicit reset circuitry.

In [SMB96], it was shown that backward retimings with equivalent initial states, such as the example in Figure 6.1, can always be obtained if the reset signal is expressed explicitly. To do so, the transformation shown in Figure 6.19 is used, which replaces a reset-FF by a multiplexer and a no-reset FF. The select input of the multiplexer is controlled by the reset line; when the reset input is 1, the initial state value B is passed through, and when it is 0, then the standard data input D is passed to the output of the no-reset FF. For a given initial state value B, this multiplexer can be simplified as shown in Figure 6.20. For an initial value of 1, the multiplexer is replaced by an OR gate. This OR gate is fed by the reset line and the data line D. When the reset line is 1, the output of the OR gate is 1, and when it is 0, the output of the OR gate is D, and is stored in the FF. Similarly, for an initial state value of 0, the multiplexer is replaced by an AND gate fed by inverted reset signal and the data signal D. When reset is high, the output of the AND gate is 0, and the FF is reset to 0; when it is low, the output of the and gate is D and the correct value is stored in the FF.

Figure 6.21. Retiming using an explicit reset [SMB96] ©1996 IEEE.

Using explicit reset circuitry provides an alternative means of retiming the circuit in Figure 6.3, as shown in Figure 6.21. The original circuit is first

transformed by applying explicit reset to all reset-FF's. Since the reset input has now been removed from each of the FF's in the circuit, they may be retimed using the standard retiming techniques of Chapter 4 without considering initial state issues. After the retiming is performed, the FF's in the final circuit may again be replaced by reset FF's. In case of forward retimings, the initial value can be found using the procedure outlined in Section 6.1, and in case of backward retimings, a justification process may be used to obtain the new initial states and remove the reset logic where possible. In the example, this step provides two choices, resulting in the two circuits in the last row of Figure 6.21. The circuit on the right is identical to that obtained by the method of [TB93] in Figure 6.6, by choosing the 00 state as the new initial state, as in step 2 of the example reproduced in Section 6.1. The circuit on the left corresponds to the choice of 11 as the new initial state.

This method requires the addition of a multiplexer before the FF, and thus changes the path delays in the circuit in situations where the multiplexer cannot be removed. This may cause the clock period of the circuit to increase. However, explicit resets can still be applied on non-critical paths to reduce the number of FF's in the circuit. A minarea initial state retiming method that allows explicit reset logic must model the area cost of the FF's, the reset logic and the routing cost of the reset lines.

Notes

1. This number may be less than k if some FF's are moved off the reset line by forward retiming moves

2. For circuits with multiple types of memory elements that cannot be combined, compatibility can be defined similarly.

3. Sharing classes for circuits with different types of FF's can be defined similarly.

7 MISCELLANEOUS ISSUES IN RETIMING

In previous chapters, we have described techniques for minimum area and minimum period retiming, and have reviewed extensions of the idea to retiming control logic while maintaining the initial states. In this chapter, we will overview the application of gate-level retiming under other scenarios and other objective functions.

While retiming can be used to improve the timing behavior of a circuit, it is important to also consider its effects on other phases on the design process. One of the more expensive parts of the design process is in testing, and in Section 7.1, we list results that describe the effects of retiming on test. Various circuit properties that are affected by retiming are pointed out, and methods that use retiming to speed up test generation time are overviewed. Next, the verification issues that are brought up by retiming are studied in Section 7.2. The use of retiming in reducing the circuit power dissipation and in enhancing the quality of logic synthesis is reviewed in Sections 7.3 and 7.4, respectively. The role of retiming in FPGA optimization is briefly described in Section 7.5, and finally, real-world issues in retiming, such as the incorporation of complex logic styles and the use of realistic delay models, are overviewed in Section 7.6.

N. Maheshwari et al., *Timing Analysis and Optimization of Sequential Circuits*
© Kluwer Academic Publishers 1999

7.1 RETIMING AND TESTING

7.1.1 Effect of Retiming on Testability and Test Generation Time

The problem of automatic test pattern generation (ATPG) is that of developing algorithms that will automatically determine the stimuli necessary to test a digital circuit according to a fault model to determine whether the circuit is faulty or fault-free. An example of such a fault model is the commonly-used single stuck-at model. The objective of ATPG is to generate a minimal number of test vectors that provide a maximal fault coverage, and this has been the subject of a tremendously large body of research. An excellent introductory text on the topic is [ABF90].

Since the retiming transformation alters the structure of the circuit, it is likely that it would have an effect on the generation of test patterns. An investigation of the effect of retiming on the test generation time was reported in [MEMR96]. In this work, the ATPG run time of a circuit was compared with that obtained for a temporally equivalent version of the circuit that was retimed for high performance. It was reported that the retimed circuits could take up to two order of magnitudes longer for ATPG than the original circuits, and in many cases, were seen to have lower fault coverage.

An explanation of this phenomenon was sought using conventional wisdom. Two metrics that have traditionally been considered to affect the complexity of test generation are the sequential depth of the circuit (defined as the maximum number of sequential elements encountered in any traversal between a primary input and a primary output), and the number and length of cycles in the circuit. Since retiming does not affect either of these, they cannot account for the observed increase in ATPG runtime. This discrepancy in the CPU times was explained using the concept of *encoding density.*

The total number of possible states in a circuit are $2^{|FF|}$, where $|FF|$ is the number of flip-flops in the circuit. However, not all of these states are valid (i.e., not all states can be traversed). The term "density of encoding" is defined as the fraction of the total number of states that are valid. The experimental results in [MEMR96] indicate that the lower the density of encoding of a circuit, the higher the test generation time. Since structural test generators do not process state transition information, the smaller the percentage of states that are valid, the greater the chance that the test generator will spend time attempting to traverse an invalid state. Thus, the smaller the fraction of the total number of states that are valid, the greater the difficulty of state traversal, and the higher the complexity of sequential automatic test pattern generation.

Experiments on circuits with a large number of FF's and a high density of encoding resulted in faster test generation than for circuits with less FF's and lower density of encoding. It was also observed that the circuits that were retimed for high performance had a significantly smaller density of encoding that the original circuit, leading to the conclusion that the density of encoding is correlated with the test generation time.

The same authors carried out further research on the effect of retiming on the testability of a circuit in [EMRM95, EMRM97]. It was shown that functional synchronizing sequences are preserved under retiming by adding a prefix sequence of a predetermined number of arbitrary input vectors.

Forward retiming moves across a fanout node introduce distinct individual FF's on different fanouts of the same signal. Consider the case of a fanout node with connections to two gates. If a single FF is moved from a fanout stem to the two different fanout points, as in Figure 7.2, then at power-on, each FF takes a random logic value. Therefore, it is possible that different FF's connected to fanouts of the same signal in the retimed circuit may have different logic values on power-on. Thus, the retimed circuit can have a power-on state that is not equivalent to any power-on state in the original circuit. However, the FF's connected to different fanouts of the same signal in the retimed circuit will have the same logic value after one clock cycle. Hence, every state in the retimed circuit, after one clock cycle, is equivalent to some power-on state in the original circuit.

Extending this argument, if the retimed circuit is obtained by a maximum of k forward retiming moves across any fanout node, then k cycles after power-on (with arbitrary input vectors) every state in the retimed circuit is equivalent to some power-on state in the original circuit. Thus, the application of the synchronizing sequence of the original circuit to the retimed circuit after k cycles will synchronize it. In other words, if a synchronization sequence for the original circuit is I with the corresponding synchronization state q, and P is any arbitrary sequence of length k, then the sequence $< P, I >$ synchronizes the retimed circuit to a state that is equivalent to q. Similarly, if a maximum of k' backward retiming moves are applied, then the synchronization sequence I may be appended with an arbitrary vector P' of length k' to get a sequence, $< I, P' >$ for the retimed circuit that synchronizes it to an equivalent state as that in the original circuit. The above statements are also shown to hold for the faulty circuit.

For the detection of a fault, it is necessary to apply a synchronizing sequence followed by a distinguishing sequence. Let P be a sequence of arbitrary vectors of length equal to the maximum number of forward retiming moves across any node in the original circuit. It is shown that if T is a test for a fault f in the original circuit, then the sequence $< P, T >$ is a test for the fault f' in the retimed circuit that corresponds to fault f in the original circuit. Moreover, it is shown through examples that without adding the prefix sequence P, neither a structural nor a functional test set is guaranteed to be preserved under retiming. The result is also shown to be true under the multiple observation time testing strategy.

This idea is applied to retime a circuit with the objective of reducing the run time of the ATPG tool. A retimed circuit is generated with a set of characteristics that increase the likelihood that the ATPG run time for the circuit will be low. Unfortunately it is not known what objective function to use in retiming for faster ATPG. However, performing minarea retiming on a circuit

could be a good heuristic to increase the density of encoding, and thus improve ATPG run time. Since the retimed circuit is only used by the test pattern generator, it need not satisfy any performance, power or area constraints. The test vectors generated on this retimed circuit are translated by prefixing the required sequence of arbitrary inputs, and is fault simulated on the original circuit. Experimental results showed that the CPU time for ATPG differed by about two orders of magnitudes on many circuits, between the original circuit and a retimed version.

7.1.2 Retiming and Redundancies

A stuck-at-fault that does not change the input-output behavior of a circuit is referred to as a redundant fault. In case of a combinational circuit, such a redundant fault can easily be identified as an undetectable fault, and the its presence denotes a combinational redundancy. In sequential circuits, a redundant fault is one that does not prevent the correct output sequence from being produced, for all input sequences starting from any initial state. Any combinational redundancy in a combinational block of the circuit is also redundant in the sequential circuit. On the other hand, a redundant fault in a sequential circuit that is not combinationally redundant is called a sequentially redundant fault. Examples of a sequentially redundant faults are (1) a stuck-at-fault that changes the next state of the faulty machine to a different, yet equivalent, state, and (2) a fault that is detectable only by an unreachable state. The identification of sequential redundancies is a difficult problem with no known solutions for large circuits.

The procedure of retiming changes the state assignment of the circuit. While the retimed circuit always has at least one state corresponding to each reachable state in the original circuit, the same is not necessarily true of unreachable states; there may be no state in the retimed circuit that corresponds to an unreachable state in the original circuit. As a result, some sequentially redundant faults may be converted into combinational redundancies by retiming, as pointed out in [DC94].

The example [DC94] in Figure 7.1 shows how a sequentially redundant fault may be converted to a combinational redundancy using retiming. The original circuit in Figure 7.1(a) has no combinational redundancies, but has 27 sequentially redundant faults. An analysis of the structure shows that two disjoint paths from stem $s1$ reconverge at gate $g11$, and the nodes $\{s1, g10, Z, Y, g8, g15, g16, g9, g11\}$ form a reconvergent region. If the circuit is retimed by moving the registers out of the reconvergent region, as shown in Figure 7.1(b), removing registers Y and Z and creating registers R and S in their stead, four sequentially redundant faults, $\{g14 \rightarrow g8s/0, g14 \rightarrow g8s/1, g8 \rightarrow g16s/0, g8 \rightarrow g15s/0\}$, are converted to combinational redundancies.

A detailed study of the effect of retiming on the different kinds of sequential redundancies can be found in [DB96]. Since combinational redundancies are much easier to detect and remove than sequential redundancies, retiming is used

Figure 7.1. Effect of retiming on sequentially redundant faults: (a) the initial circuit, and (b) the retimed circuit [DC94] ©1994 IEEE.

as a vehicle to identify them by converting them to combinational redundancies as in [DC94, YKK95b].

A retiming procedure to reduce the number of unreachable states in an attempt to convert sequential redundancies into combinational redundancies is presented in [YKK95b]. This procedure attempts to move FF's to the fanout stem from the fanout branches. For circuits that have a specified initial state, a similar procedure to convert sequential redundancies into combinational redundancies is presented in [YKK95a]. A 5-valued logic (0,1,X,1x,0x) is used when moving FF's, where 0, 1 and X have their usual meanings. The symbol, 1x, means that an FF may take a value of 0 or X (don't care), depending on values of other FF's to be retimed. For example, when an FF with value 0 is moved from the output of a two-input AND gate, then the value of FF's at the inputs is 0x, so that if one input is assigned a value 0, then the other can take a value of X. FF's are permitted to move towards a fanout stem only if their values do not conflict with each other.

In [DC94], a minimum cost flow formulation is presented for retiming a circuit to convert sequential redundancies into combinational ones. The method repositions FF's in an attempt to minimize the number of registers in any strongly-connected components in a circuit. This results in a tendency to move the FF's to the peripheries of reconvergent regions, exposing reconvergent com-

binational logic. Since reconvergent regions may contain redundancies, the effect is that the sequential redundancies enclosed in the region are likely to become combinational redundancies after retiming.

7.1.3 Scan-based Testing

The complexity of test generation for sequential circuits is well known to be more than that for combinational circuits. Scan-based testing is a method that is used to reduce the expense of test generation of sequential circuits. This method uses a scan chain of registers to make internal registers controllable and observable [ABF90]. The length of this scan chain affects the test time, and hence, small chains are desirable. If all of the FF's in a design belong to the scan chain, then the design is said to be a full-scan design; if only a subset of the FF's belong to the scan chain, then the design is called a partial-scan design.

For full-scan design, the method of minarea retiming can be used to reduce the length of the scan chain since this would reduce the total number of FF's in the design. In [HKK95], retiming is used with reduced shift scan to compact test sequences for full-scan designs. Since the repositioning of FF's by retiming changes the control and observation requirements of FF's for some test vectors, in some cases retiming could reduce the test length even when the number of FF's did not decrease. In [HKK96], a technique called partially parallel scan chain is presented to reduce the shift clocks required for testing a design. This method arranges the scan chain in a partially serial and partially parallel fashion, and uses retiming techniques to select the FF's to be placed in parallel.

Retiming can also be used for reducing test lengths in partial-scan based designs [KT93, KT94, KT96]. A generally effective criterion for selecting FF's in the partial-scan problem is to choose the FF's that break the cyclic structure of the circuit and reduce its sequential depth. Unfortunately, finding the minimum number of FF's that break all of the cycles in the design is equivalent to the Minimum Vertex Feedback Set (MVFS) problem, which is known to be NP-hard [GJ79]. It is shown in [KT96] that the average number of cycles that each FF participates in is maximized in the minarea configuration. This measure is a good heuristic since the minarea circuit is likely to have a small MVFS set; however, minarea retiming does not guarantee the optimal MVFS, as illustrated in a counterexample in [KT96]. The essential reason for this is that two or more FF's may participate in exactly the same cycles, and hence, the cycles may be counted incorrectly in a minarea objective function. To offset this problem, a binary configuration for minarea was used in the work, allowing at most one FF on any edge. The logic behind this is that having more than one FF on the same edge does not at least increase the pool of "good" candidates for breaking cycles. Further, the presence of more than one FF on a chain path (i.e., fanout free path) could result in having all of the FF's on the chain path to belong to exactly the same cycles, and was be discouraged. In addition, the work presents a retiming-based algorithm to reduce the sequential depth of the circuit and a technique for timing-driven partial scan.

The method in [DC94], described earlier, sets up a minimum cost flow formulation to retime a circuit with the objective of reducing the total number of FF's in the strongly connected components of the circuit. This has the effect of reducing the number of FF's required in the partial-scan chain.

In [CD94], retiming is used with resynthesis to find the minimal set of FF's to be inserted in the partial-scan chain. The basic idea is motivated by the fact that the size of the MVFS of the circuit graph (called the CMVFS, or the minimum number of gates whose removal makes the circuit acyclic) is a lower bound on the MVFS of the FF dependency graph. In this approach, the given circuit is retimed so that an FF exists at the output of every gate in the CMFVS. The FF's on the outputs of gates in CMFVS are then selected to give the minimal partial-scan chain. The conditions that a circuit must satisfy for retiming to position an FF at the output of every gate in CMFVS are also presented, along with resynthesis techniques to modify a circuit to satisfy these conditions. A network flow formulation is used to reposition FF's at the output of the gates in CMVFS. Experimental results on ISCAS89 benchmark suite show significant reductions in the number of FF's required in the partial-scan chain over previous techniques, although the retiming and resynthesis steps result in some increase in the area of the circuit.

In [BC96] an ILP formulation is presented for performing retiming with logic duplication, a procedure referred to as the RLD transform. The objective here, again, is to minimize the number of FF's in the partial-scan. The RLD transform achieves the same number of FF's in the partial-scan chain as in [CD94], but with a smaller area penalty.

7.1.4 Pseudoexhaustive Testing

Pseudeoexhaustive built-in self test is another method for reducing the testing complexity of sequential circuits. This process consists of breaking all cycles in the circuit, partitioning it into subcircuits by placing some segmentation cells, and balancing the partitioned circuit by inserting additional FF's as bypass delays in order to apply combinational ATPG methods [KTB93]. To reduce the test application cost for subcircuits with large number of inputs, the circuit is partitioned such that the number of inputs to each subcircuit is less than a given number. Thus, the objective of the partitioning procedure is to minimize the number of segmentation cells required.

In [LKA95a], peripheral retiming [MSBS91] is used to find the FF's that cause the unbalanced structure in the acyclic circuit, and these FF's are used for segmentation, as far as possible, in the partitioning stage. An iterative way to minimize the number of segmentation cells required for partitioning the circuit using peripheral retiming and resynthesis is presented in [LKA95b, LKA95c].

The procedure in [KT94, KTB95] presents a retiming heuristic for bounding the sequential depth, satisfying the limit on number of inputs for each subcircuit, and balancing the circuit while preserving the clock period. A retiming-based method is also presented to balance a circuit by inserting minimum number of delays. The method uses the binary configuration presented in [KT96]

while adding the additional constraints to ensure the preservation of the clock period. The method to balance a circuit first greedily inserts the bypass delay cells to make the circuit balanced. The circuit is then retimed to remove redundant delay cells.

7.2 VERIFICATION ISSUES

A sufficient condition that allows a design to be replaced, independently of its environment, is the classical notion of equivalent FSM's. Two FSM's are equivalent if, for every state in the first machine, there is an equivalent state in the second machine, and vice-versa. A weaker condition, which is both necessary and sufficient for replacement, is called safe-replacement and is described in [PSAB94]. Safe replacement guarantees that if an old design is replaced by a new one, it is impossible for any environment to detect that the replacement has been made. In the absence of reset lines, safe-replacement requires that no matter which state the new design powers up in, the subsequent input-output behavior of the design must be acceptable to the environment. This requires that the environment be able to drive a replacement machine into its steady-state behavior (i.e., reset the machine) with the same sequence that is used to reset the original machine. It has been shown in [PSAB94] through an example that retiming does not always satisfy the safe-replacability condition.

The reason why retiming fails to satisfy the safe-replacement condition is similar to the discussion of [EMRM97] in Section 7.1.1: due to the effect of a forward motion of a register across a multi-fanout junction. Forward retiming moves across a fanout-node result in possible power-on states in the retimed machine that were not possible in the original design. For example, consider a fanout stem with two branches, with one FF on the stem in the original circuit, as shown in Figure 7.2(a). No matter which state (0 or 1) the FF powers up in, both the fanout branches have the same logic value. Now if the FF is retimed forward across this fanout junction, as shown in Figure 7.2(b), we have one FF on each of the fanout branches and it is possible for them to power up in different state (say 0,1). This can result in a power-on state that has no equivalent state in the original circuit. This intuition leads to the result in [SPRB95] that states that if forward moves across a fanout node are not allowed, then retiming satisfies the condition of safe replacement.

Figure 7.2. Example illustrating the effect of forward retiming on replaceability.

If, however, the environment can wait for a certain specified number of cycles (dependent upon the retiming) before applying the inputs, then retiming (including forward retiming moves across a fanout node) becomes a safe transform. In the above example, if we wait for one cycle after power on, both of the FF's will have the same logic value since they are driven by the same node, and thus, this delayed design will be same as the original design before retiming. It is shown in [SPB95] that the environment must wait k cycles before applying inputs to the retimed design in order for the design to be a safe replacement of the original design, where k is the maximum number of forward moves made across a non-justifiable element; an element F with m outputs and n inputs is a non-justifiable element if there exists $y \in 2^m$ such that for all $x \in 2^n$, $y \neq F(x)$. A fanout node is an example of a non-justifiable element.

It is also shown in [SPRB95] that while an accurate logic simulation may distinguish a retimed circuit from the original circuit, a conservative three-valued simulator (CLS) cannot do so. The intuition behind the result is that a CLS does not have the information about two don't care values being identical. Therefore, a CLS cannot recognize that two X's actually originated from the same fanout stem. In the example in Figure 7.2, in the original circuit, the CLS will assign X's to both the fanout branches at power on, as it will do for the retimed circuit, but a more accurate simulator would recognize the fact that in the original circuit both fanout branches have the same logic value. Although three valued simulators are most commonly used in practice, since most environment wait some cycles after power-on before processing the outputs from a design, retiming can be considered a safe replacement in practice.

In the real world, a more important question that must be answered is whether a retimed design is temporally equivalent to the original one. This is important because even though retiming is a safe replacement for a conservative simulator, the retiming tool used may be faulty. Thus, there is a need to verify retimed circuits. A first step is to ensure that the number of latches on each cycle is the same before and after retiming. The approach in [SSBS92] shows that a set of fundamental cycles may be identified, and it is enough to check for these temporal constraints on the fundamental cycles. For verification at the logic level, a verification method based on sequential ATPG techniques is presented in [HCC96, HCCG96], checking the three-valued equivalence of the retimed circuit. In this approach, the original circuit M_1 and the retimed circuit M_2 share the same primary inputs but have their own separate registers. The primary outputs are connected to an XOR gate whose output is denoted by g. The comparison is now performed by attempting to find a test vector that will excite a 1 at g. If such a test vector is found, then the retimed circuit is different from the original circuit. A preprocessing step is used to find equivalent internal signal pairs that reduce the search space, thus speeding up the whole verification process. Experimental results on circuits hundreds of FF's are also presented in this work. The same authors present a faster verification tool in [HCC97] by taking advantage of local binary decision diagrams (BDD) based approaches. The use of local BDD's makes this new approach less sensitive to the degree

of structural similarity as compared to the pure ATPG-based approaches of the pervious tool, and enables it to handle much larger sequential circuits. Experimental results on circuits with thousands of FF's are presented in this work.

In [Ran97, RSSB98], a method for verifying feedback-free sequential circuits generated by retiming and resynthesis is presented. General sequential circuits with feedback are handled by fixing the location of some registers in the design. The sequential verification problem is then reduced to an equivalent combinational verification problem to leverage powerful combinational verification techniques. This method trades off some of the power of sequential optimization to reduce the complexity of verifying the optimized circuit.

7.3 RETIMING FOR LOW POWER

The location of FF's in a circuit affects the capacitance and the switching activity on a net, particularly since the output of an FF switches only once every clock cycle. Since the power dissipated in a circuit depends on the product of the switching activity and the load capacitance on every node, the power dissipation of a circuit can be reduced by placing FF's on nets with high switching activity.

In [MDG93], a heuristic for minimizing power dissipation is presented, using retiming (in fact, by pipelining a combinational circuit). First, an estimate of the average switching activity of the combinational network, ignoring the FF's, is obtained. The power is computed under both zero delay and actual delay models, with the difference between power calculated under these two models giving the glitching power, E_{glitch}, at each gate. Next, the sensitivity $s_{j,i}$ of gate j relative to gate i is obtained, which is the probability of a transition at gate i causing a transition at the output of gate j. This sensitivity is then weighted by the capacitive load of the corresponding node, and the power reduction obtained by placing an FF at the output of node i is now given by

$$\text{power_red}(i) = E_{glitch}(i) \cdot \left(C_i + \sum_{j}^{fanout_i} (S_{j,i} \cdot C_j) \right).$$

Since the transitive fanout of a node may contain a large number of nodes, the number of levels traversed is restricted in order to control the computation time. Moreover, while it does not accurately consider the effect of large glitches, this captures the fact that small glitches are filtered out in a few levels by the inertial delay of combinational logic.

The number of FF's in a circuit can significantly affect the power dissipation in a circuit and it is desirable to contain the number of FF's added when retiming for low power. The method in [MDG93] gives a higher preference to nodes with higher number of fanins and fanouts, thus potentially placing the FF on a larger number of paths. Thus, the final cost function is:

$$weight(i) = \text{power_red}(i) \cdot (|FI(i)| + |FO(i)|).$$

The algorithm now places one stage of FF's at a time. This helps to reduce the error due to correlation between the switching activity at different nodes. A set of nodes, with no more than one node per input-output path and with the highest sum of weights is found by a binary search. An FF is added to each of the primary inputs and retimed until all of the nodes selected in the set above have an FF on their output. This process may leave some extra FF's that are not on these selected nodes, and are free to move around. Some simple heuristics, such as locally minimizing the number of FF's, are used to place them. This whole process is repeated until the required number of stages have been added. Clock period constraints can also be added in this process to ensure that the retimed circuit satisfies the required clock period. The results presented in [MDG93] show a 10% to 15% improvement in power dissipation using this method.

A second approach to power minimization uses a procedure called fixed-phase retiming [LP96], which transforms a given edge-triggered circuit into a two-phase level-clocked circuit by replacing each edge-triggered FF by two latches. Next, the latches of one phase are kept fixed, while those belonging to the other phase are relocated onto wires with high glitching activity and capacitive loads. A $O(V^6 \log V)$-time algorithm for fixed-phase retiming using boolean monotonic linear program is also presented in this work. This method, however, does not take into account the effects of cycle stealing in level-clocked circuits.

7.4 RETIMING WITH LOGIC SYNTHESIS

7.4.1 Retiming with Combinational Resynthesis

A significant amount of work has been performed on combining retiming with logic synthesis. An early approach [De 91] is to incorporate FF motion as yet another transform in the logic synthesis process. In this work, timing optimization of the entire sequential circuit is considered, as against the traditional approach of optimizing combinational blocks separately. The basic graph model here uses the concept of a synchronous Boolean network, where the registers are considered together with the combinational components. Commonly used combinational logic synthesis transforms, such as elimination, resubstitution, extraction, and decomposition, are extended to operate within and across the register boundaries by exploring the possibility of moving the register positions. The advantage of this approach is that small motions of FF's (local retiming) can be performed in conjunction with combinational synthesis. However, the global view taken by the traditional retiming algorithms is not utilized in this approach.

In [BBR91], combinational logic optimization tools are extended to support multiphase circuits and allow movement of logic across latches. The logic is manipulated heuristically after technology mapping so that accurate delay models are used to determine which logic to move. Negative moves up to some threshold are permitted, and are later undone if they are not resolved by other

moves. Experimental results presented indicate an extra 10-20% improvement when compared to pure combinational optimization.

The idea of integrating retiming with resynthesis was also used in [MSBS90, MSBS91]. A technique called peripheral retiming is used to temporally move latches to the boundaries of the circuit. Next, the logic between latches is optimized for speed, using existing combinational techniques, after which the circuit is retimed again to reposition the latches. The idea of negative registers at peripheral edges (edges associated with primary inputs and outputs) is used to temporarily borrow registers from the environment, with the requirement that these must be returned in the final retiming step after synthesis. Conditions on the given circuit are also presented that guarantee that all of the borrowed FF's can be returned to the environment; if these conditions do not hold, it is possible that all of the borrowed FF's cannot be returned after resynthesis and retiming, thereby invalidating the procedure. In [MSBS93], peripheral retiming is used to optimally resynthesize a pipelined circuit to meet performance specification by using the equivalence, proved in this work, between pipelined circuit optimization and combinational speedup.

An improved technique for technology-dependent local optimization for tree structures, incorporating retiming, is reported in [GLH+94]. Tree mapping involves the selection of cells from a given library to implement a tree-structured circuit or subcircuit. While traditional tree mapping algorithms allow latches only at tree roots, this work permits latches to be located at interior points as well, and determines optimal latch positions by retiming. In this work two cell libraries are maintained, one each for the combinational and the sequential cells. Complex latches that contain some logic at their inputs are allowed by considering only cells from the sequential library while matching a node on whos output a latch is desired. At each node in the tree, a list of best solutions is stored, one for each retiming weight (i.e., the number of latches between the node and the tree root), and a procedure similar to tree mapping is performed, maintaining the constraint that the number of latches between any leaf and the root of the tree must be preserved during the mapping procedure.

In [HE97a], experimental results are reported on the iterative application of retiming and synthesis. It was observed that the successive use of retiming and synthesis produces as much as twice the gain in clock period over using a single retiming/synthesis step. It was seen that most of the improvements occur earlier in the iterative process, and that although the sequence in which retiming and synthesis were performed affected the result, no one sequence was consistently better than the other. It must be noted, however, that the results reported are on small circuits only, and do not use minarea retiming.

The problem of enabling retiming by transforming the combinational part of the circuit is addressed in [DPR92]. This method makes combinational synthesis aware of the possibilities that retiming offers in relaxing the critical paths. To guarantee the existence of an equivalent initial state in the final circuit, only forward movement of FF's is considered. The bottlenecks on retiming are identified, and this information conveyed to a combinational delay optimizer in

the form of arrival times at the inputs and the required times at the outputs. Instead of using 0 as the arrival time for all inputs, and the clock period (P) as the required time for each output, this work provides a more realistic set of arrival and required times by using the slack information at the FF's. The input slack, $s(x)$, on an FF, x, is defined as the difference between the required time and the arrival time at its input. The FF's are partitioned into two sets, the first of which is called the forward slack latch (FSL) set, and contains all FF's that have a positive input slack, i.e., can be moved forward by some amount, equal to the slack at the input, without making any path to this FF critical. All other FF's are placed in a second set, called the nonforward slack latch (NFSL) set. Timing constraints on all combinational paths in the circuit are now derived, assuming that all of the FF's in the FSL can move forward, e.g., a path p from FF u to FF v is required to have a delay less than $P + s(u) - s(v)$. Once the combinational delay optimizer is able to satisfy all of the timing constraints, the circuit is retimed to move all FF's in the FSL set by an amount determined by their respective slacks.

Since this slack information is only local, the above delay constraints may be unduly restrictive. In [CDPR93] an improved method is presented, which simultaneously exploits the delays on all paths in the circuit. This method recognizes that a combinational delay optimizer can satisfy some delay constraints more easily than others. A measure of difficulty for delay optimization is proposed, and is used to determine a delay constraint set that is optimal in that it is the "easiest" constraint set for the optimizer. The problem is formulated as a minimum cost flow problem. After delay optimization is successful, it is possible for some paths in the resynthesized circuit to have delays that are larger than the clock period; these paths are to be fixed by retiming the resynthesized circuit.

An alternative way to make retiming aware of the capabilities of combinational synthesis in shortening the critical paths is utilized in [AB98]. The Shannon transformation is used to predict the speedup, and speedup edges are marked out. These edges indicate the potential of speed-up obtainable by resynthesis and are used during retiming.

7.4.2 Architectural Retiming

Architectural retiming [HE95, HE96, Has97, HE97b] is a technique for optimizing latency-constrained circuits. Formally, it can be defined as follows: Given a synchronous circuit whose clock period is limited by a latency-constrained path, increase the number of registers on the path, thereby decreasing the clock period, while preserving the circuit's functionality and latency.

Latency constraints occur often during the design of high-performance systems because of either performance requirements or cyclic dependencies. An example of a latency constraint is a critical cycle in a design. The computation along a latency-constrained path (or cycle) is required to complete in a fixed number of cycles. Since simply adding a register to the cycle will change the behavior of the circuit, which is not permissible, architectural retiming adds a

negative register and a normal register in series to replace a selected wire in the circuit. A negative register, unlike a normal register performs a shift backward in the time, i.e., the output of a negative register is defined as $z^t = y^{t+1}$.

The problem is now reduced to: (a) finding the location in the latency-constrained path to add this pair, and (b) finding a way to implement the negative register in hardware. A heuristic approach for finding potential locations to add the register pair is presented in [Has97]. Critical single-register cycle are always selected for adding the negative register/normal register pair. The heuristic gives priority to a node which affects more critical paths, and to nodes that are farther forward in a combinational stage.

Two implementations of the negative register are presented in [HE96]: precomputation and prediction. The idea behind precomputation is as follows. The negative register is implemented as a function f' that computes the output of the negative register one cycle before the input actually arrives. The implementation relies on signals available from the previous pipelined stage to compute the output. The function f' is derived by recursive collapsing (or flattening) of the input to the negative register across one register boundary. The function f' synthesized by the precomputation includes more than one clock cycle's worth of computation.

At logic level, precomputation exposes a larger combinational block in the hope that resynthesis will reduce the delay, and can, therefore, be viewed as a forward retiming move combined with logic duplication. Architectural retiming using precomputation at the logic level is compared with iterative application of retiming and resynthesis in [Has97]. It is seen that architectural retiming results in better clock periods (at a higher area penalty) than two consecutive resynthesis-retiming steps; however, if up to 15 successive steps of resynthesis-retiming are performed, the obtained clock period is significantly better than that obtained by architectural retiming and at a much lower area penalty. Further efforts in optimizing the precomputation algorithm are necessary before it can be shown to be better than iterative resynthesis-retiming steps.

Precomputation is not always possible, particularly if there is a purely combinational path from a primary input to the negative register. In such cases, the alternative to precomputation is prediction. In this situation, an oracle predicts the value produced by the negative register one cycle before the value is actually computed. The predicted value is compared with the actual value computed one clock cycle after prediction. If the prediction is found to be correct, the circuit functions normally; however, if there was a misprediction, then the effects of this incorrect prediction must be nullified.

A systematic way of synthesizing the oracle by an FSM is presented in [HE97b]. Don't care conditions are also utilized to use an incorrectly predicted value if it does not change the behavior of the design. The design interface is modified to inform the design environment of a misprediction. When a misprediction is detected, one cycle after it occurs, the design environment is instructed to ignore the invalid output and to provide the primary inputs again for the correct computation to proceed. This can be done by adding a

"data-valid" and a "data-not-taken" signal. Two strategies for recovering from a misprediction are presented in [HE97b]: (a) the ASAP restoration which views the misprediction as an urgent matter to be immediately attended, and (b) ALAP restoration which views it as an accident that need not be handled until necessary.

7.5 RETIMING FOR FPGA'S

Field programmable gate arrays contain a significant number of FF's that can be used for retiming and pipelining purposes. Since these FF's are already present on the chip, their use does not necessarily add to the cost of the design. In an early paper [TSS92], a method is presented to retime a placed and routed FPGA design using the accurate delay information available at that stage. In this work, FF's are only allowed at physically realizable locations. Additional constraints on the maximum number of FF's that an edge can have are added to model the limited availability of FF's at any location. The MILP formulation from [LS91] is used for retiming in the presence of these additional constraints. However, the experimental results presented in this work indicate that significant reductions in the clock period are not obtained by retiming the placed and routed design.

Various attempts have been made to integrate retiming with technology mapping for FPGA's, e.g., [WR93, CW94, PL96a, PL96b, CW96, CW97, CW98]. The process of technology mapping for a k-input lookup table (LUT) based FPGA's first partitions the circuit into a cluster of subcircuits, such that each subcircuit has less than k inputs, and can, therefore, be fitted in one LUT. Conventional technology mapping algorithms assume the location of registers to be fixed, and perform technology mapping on only the combinational subcircuits.

In [WR93], acyclic sequential circuits are technology-mapped using combinational technology mapping algorithms by treating FF's as combinational nodes. This allows interactions between logic separated by FF's, improving the quality of technology mapping. After mapping, the FF's that remain within the logic blocks are moved to the borders of LUT's by retiming. This retiming may require some logic to be duplicated, causing an increase in the number of LUT's needed to map the design. General sequential circuits are converted to acyclic circuits by temporarily converting an FF to a primary input-primary output pair that is converted back to an FF after mapping.

In [PL96a, PL96b], the FF locations are assumed to be fully dynamic in that they may be arbitrarily repositioned by retiming. This results in a mapping algorithm that produces the same result, irrespective of the initial FF locations. To fully exploit the potential of retiming, logic duplication is permitted. This method results in the minimum clock period for the unit delay model. The idea of expanded circuits is introduced to represent all possible k-LUT's under retiming and node replication. An expanded circuit is constructed by replication, with the property that all paths from any given node to the only output node have the same number of FF's. An LUT for a node can be derived from a cut in the expanded circuit for that node. To find the optimal mapping,

an iterative labeling scheme is used. The run-time of this method is significantly improved in TurboMap [CW96] by improving the labeling process and reducing the size of the expanded circuits. TurboSYN [CW97] combines the combinational functional decomposition technique with retiming and pipelining to perform sequential functional decomposition, and incorporates it in the label computation of TurboMap [CW96] to eliminate all critical loops. In TurboMap-frt [CW98], this method is further extended to take into account the initial state computation while retiming. This is done by first retiming the circuit so that all of the FF's are moved as far as possible towards the primary inputs, without regard to clock period, but with the restriction of guaranteed equivalent initial states. The mapping phase next moves FF's only in the forward direction, thus simplifying the equivalent initial state computation.

7.6 PRACTICAL ISSUES

7.6.1 Retiming DAG's

The work in [CMP+97] presents a simple and efficient algorithm for retiming circuits whose topology may be represented by a directed acyclic graphs (DAGs), i.e., circuits without cycles or feedback paths. The utility of this approach is that several commonly-used arithmetic circuits have a DAG structure. The algorithm, inspired by an earlier method in [MH91], requires only two passes, as opposed to the $|V|$ passes of the more general retiming algorithms. In the first pass, all registers are moved as close as possible to the primary inputs, while ensuring that the edge weight on all edges is non-negative. In the second pass, all registers are moved as far as possible from the primary inputs, while satisfying the clock period constraints in addition to the non-negativity constraints. The first pass operates in the reverse-topological order, while the second pass operates in topological order. If at the end of of the second pass, if some purely combinational paths still have delays that are larger than the clock period, then the given clock period is not feasible.

7.6.2 Circuit Issues in Real Designs

One of the main obstacles to applying the retiming theory to real design is the incapability of the traditional retiming algorithm of taking into account the various peculiarities of real designs, such as registers with logic or load enables, multiple clocks and multi-cycle paths, existence of timing insensitive paths, gated clocks, precharged circuits, etc.

In [LVW97], a method for handling multiple clocks and load-enable registers, called mc-retiming, is presented. This work defines classes of registers, where each class consists of all FF's connected to the same clock and load enable signal. Only FF's belonging to the same class can be retimed across a gate at the same time.

The Leiserson-Saxe graph representation is modified to include an ordered list $l(e_{uv})$ with every edge. This ordered list contains one entry for each class,

which gives the number of FF's of that particular class on the edge. As a preprocessing step, all possible forward retiming moves are performed while ensuring that only FF's belonging to the same class are moved across a gate at any given time. The number of FF's so moved across a gate v is called $MVR(v)$, and the retiming so obtained is called *maximal forward mc-retiming.* Maximal backward mc-retiming is similarly defined to be the retiming in which all FF's have been moved as far back as possible, and $MBR(V)$ is the maximum number of FF that can move back across a gate v. No FF is allowed to move across the primary inputs or primary outputs.

Using $MBR(v)$ and $MFR(v)$ as the upper and lower bounds on $r(v)$, mc-retiming can be defined as follows. A retiming $r : V \leftarrow Z$ is a valid mc-retiming if and only if

$$\forall e_{uv} \in E \ : \ r(u) \leq r(v) + |l(e_{uv})|$$

$$\forall v \in V \ : \ -MFR(v) \leq r(v) \leq MBR(v)$$

$$\forall v \in I \cup O \ : \ r(v) = 0$$

Notice that the concept of bounds is similar to that used in Minaret in Section 5.2; however, the bounds in Minaret are calculated after taking into account the target clock period and are derived using the equivalence between retiming and skew.

The upper and lower bound constraints are represented in the graph as additional edges, and traditional retiming algorithms are used to solve both the minperiod and minarea retiming problems.

For circuits with multi-cycle paths, the minperiod retiming problem can be solved if the multiple clocks are derived from one fundamental clock, i.e., the period of all clocks is some integer multiple of a fundamental clock period P. This is achieved by using P as the target clock period and inserting dummy FF's. Let $\alpha_{ij} \cdot P$ represent the time available to a signal to proceed from an FF of clock i to one with clock j, where α_{ij} is a positive integer. Then $\alpha_{ij} - 1$ dummy FF's are inserted on every path from an FF of clock i to one of clock j. The problem is now transformed into the traditional minperiod retiming problem. However, due to the granularity of gate delays, it is not possible to place these dummy FF's somewhere halfway a gate, although this might be necessary to find certain solutions. because of this reason the method cannot guarantee optimal solution. The ASTRA algorithm presented earlier also suffers from this granularity problem for real registers, but can naturally handle multicycle paths by using $\alpha_{ij} \cdot P$ instead of just P, while constructing the phase A constraint graph, eliminating the need for dummy registers. Therefore, it does not suffer from the deleterious effects of granularity on the dummy registers.

7.6.3 Retiming Gated Clocks and Precharged Structures

A technique for retiming gated clocks and precharged structures is presented in [Ish93]. For retiming gates that have clock signals as inputs, an alternative interpretation of retiming is used, instead of interpreting retiming as moving registers across a gate, since such an interpretation may not be obvious in

this situation. The interpretation is based on the observation that retiming corresponds to the application of a lag at the inputs to a gate. Thus, while retiming a gated clock, the phase of the clock is changed. This change in the phase of the clock input to the gate will also change phase of the output clock. For example, consider the two-phase circuit in Figure 7.3(a) and its retimed version in Figure 7.3(b).

Figure 7.3. Retiming structures with gated clocks [Ish93] ©1993 IEEE.

To retime (lag) the input to gate D, a lag must be applied to gate A. Since gate A has the clock signal ϕ_0 as one of its other inputs, it must be lagged by changing it to phase ϕ_1. The output of gate A, a clock signal, is now also lagged by changing its phase to ϕ_1, and the retiming step is complete.

For retiming a precharged structure consider the circuit in Figure 7.4(a). The precharged structure is retimed (or lagged) by removing the latch at the output, adding a latch of the same phase to all of the inputs, and changing the phases of the precharge and evaluate clocks.

Figure 7.4. Retiming precharged logic structures [Ish93] ©1993 IEEE.

Timing constraints for these circuits are then generated, and retiming performed using the technique in [ILP92], while following the above procedure for retiming (lagging) gated clocks and precharged structures.

7.6.4 Improved Delay Models

The traditional retiming algorithms assume a simple delay model in which the delay of a gate does not change during retiming. However, since retiming relocates registers, the fanout loading on gates and clock skew on FF's can change during the retiming process, thus changing the delays.

A more general delay model using register electrical characteristics (REC's) is presented in [SFM93, SF94, SFM97]. This model includes the effects of variable register, clock distribution, and interconnect delays. The REC's are assigned to each edge in the graph representation of the circuit, and the critical paths are now defined from an edge to another edge, unlike the traditional approaches where they are defined from a gate to another gate. Timing constraints are derived from a sequential adjacency matrix that contains all path delays. The lag values, $r(v)$, are now a range, rather than an integer as in [LS91]. These ranges are initialized with unbounded values, and localized timing constraints are used by a branch and bound algorithm to iteratively tighten these ranges.

The work in [LP95a] presents a MILP formulation of retiming under the REC delay model. In this work, a delay value for each gate is listed for the case when it fans out to another gate. Each edge has associated with it a two tuple (t_e, t_b) that lists the extra delay incurred by signals whose propagation begins or ends at an FF on this edge. If an FF exists on this edge in the retimed circuit, then the delay of the fanin gate is increased by t_e because of the fanout loading on that gate. The parameter, t_b, corresponds to the delay associated with an FF. Thus, in this model the delay of a combinational register-to-register path from FF i to FF j via a path p is given by

$$D_{total}(i,j) = t_b(i) + \sum_{x \in p} d(x) + t_e(j).$$

The difficulty of the retiming problem under this more general delay model depends on the values of these edge weights t_b and t_e. If these parameters are permitted to take any real value, then it may be possible to satisfy a timing constraint by pulling two FF's farther apart. This model does not preserve the path delay monotonicity, i.e., the property that delay of a path increases as the number of logic gates on it is increased, and therefore, traditional algorithms are not applicable for this delay model. A clever integer-linear programming (ILP) formulation is presented in [LP95a] for minperiod retiming under this more general delay model.

The formulation modifies the original circuit graph, $G =< V, E, d, w, t_b, t_e >$, to a new graph, $G' =< V', E', d, w', t_b, t_e >$, by adding a dummy vertex of zero delay on every edge, i.e., $V' = \{V \cup \{x_{uv} : u \xrightarrow{e} v \in E\}\}$, and $E' = \{u \xrightarrow{e_1} x_{uv}, x_{uv} \xrightarrow{e_2} v : u \xrightarrow{e} v \in E\}$. This dummy vertex breaks every edge, $u \rightarrow v$, into $u \rightarrow x_{uv} \rightarrow v$. The first edge, $u \rightarrow x_{uv}$, is called e_1, while the second is called e_2. It is desired that the first FF on the edge $u \rightarrow u$ should go on the first edge e_1 and the remaining FF's if any go on the second edge e_2, i.e., $w'(e_1) = \min\{1, w(e)\}$ and $w'(e_2) = w(e) - w'(e_1)$. For this purpose, additional

constraints of the form $w_r(e_1) \leq 1$ and $w_r(e2) \leq F \cdot w_r(e_2)$ are added, where F is a sufficiently large number. Restricting the r variables to integers ensures that $w_r's$ are also integers. For a clock period of P, the clock period constraint implies that any combinational path $e \rightsquigarrow e'$ with $D_{total}(e, e') > P$ should not have FF's at both its ends, i.e., $W'_{r'}(u, v) = 0 \Rightarrow w'_{r'}(e_1) = 0$ or $w'_{r'}(e'_1) = 0$. This can be rewritten as $w'_{r'}(e_1) + w'_{r'}(e'_1) \leq W'_{r'}(u, v) + 1$. Thus, the retiming problem can be formulated and solved as an ILP.

The work in [LP95a] also presents a more efficient method of retiming a special class of circuits, called one-way extendable circuits, using monotonic integer linear programming. One-way extendible circuits are those in which a clock period violation can be fixed by extending the path in one direction only. These circuits can be either one-way end-extendable or one-way begin-extendable, depending on whether the path is extended at its end or its beginning.

Figure 7.5. Effect of FF sharing on delay.

Even though the REC model is more general that other delay models used in retiming, it has its shortcomings [Mah98]. When retiming relocates FF's to the output of a multi-fanout gate, the actual delay of the gate depends on how the FF's are shared on the fanout junction. As shown in Figure 7.5, the fanout loading can be split in different ways between the gate and the FF's at its output, depending on the way in which FF's are shared. Since the fanout-dependent delay of the gate is included only in paths that end at these FF's, while the fanout-dependent delay of an FF is included only in a path starting at that FF, the delay may be split over adjacent clock cycles. This complicates the retiming problem, since in addition to deciding the location of FF's, the algorithm must also decide the sharing of FF's at multi-fanout gates.

In [IP96], a method for pipelining an acyclic, two-phase level-clocked circuit under a bounded delay model is presented. It is observed that pipelining amounts to computing a retiming with all latches initially placed at the primary inputs (or outputs) of the circuit. In the bounded delay model, each gate v is assigned a minimum delay $\delta(v)$ and a maximum delay $d(v)$. The approach taken in this work operates by iteratively solving the short path (hold) constraints and adjusting the latches to ensure that the long path (setup) constraints are satisfied. The latches always move in only one direction, the forward direction. In the first phase, called the hold phase, latches are moved only as far forward as required to satisfy the short path constraints, using d, the minimum delay values. In the second phase, called setup phase, any long path violations are fixed by moving the latch forwards at the beginning of the path, this time using δ, the maximum delay values. The hold phase is then invoked again to fix any short path violations, and the algorithm alternates between the hold and the setup phase until all constraints have been satisfied, or until one of the phases fails. It is shown that the algorithm must terminate in a polynominal number of steps.

The work in [KO95] performs retiming under imprecise delays information. The essential motivation behind the work is that the precise gate delays are not known at the time of retiming a circuit since a number of effects that could cause delay variability exist, ranging from incomplete information at a given design stage, to the presence of manufacturing variations in the fabricated circuit. The gate delay model, instead of using a precise delay, $d(v)$, for each gate, v, uses an imprecise delay, $\hat{d}(v) = (d^p(v), d^p(v), d^o(v))$, where $d^m(v)$ is the the most probable gate delay, $d^o(v)$ is the minimum possible gate delay, and $d^p(v)$ is the maximum possible gate delay. The imprecise delay of a path $\hat{d}(p)$ is defined through the definition of the addition operation on fuzzy numbers, i.e.,

$$\hat{d}(p) = \sum_{v_i \in p} \hat{d}(v_i) = (\sum_{v_i \in p} d^m(v_i), \sum_{v_i \in p} d^p(v_i), \sum_{v_i \in p} d^o(v_i)).$$

The maximum imprecise delay on any critical path from u to v is given by

$$\hat{D}(u,v) = \max_{p \in C} \hat{d}(p) = (\max_{p \in C} d^m, \max_{p \in C} d^p, \max_{p \in C} d^o),$$

where C is the set of all paths p between u and v with the minimum number of registers, $w(p)$.

The imprecise clock period of the circuit also has the most probable, the minimum and the maximum components, and is defined as

$$\hat{\phi} = \max_{p \in W^0} \hat{d}(p) = (\max_{p \in W^0} d^m, \max_{p \in W^0} d^p, \max_{p \in W^0} d^o),$$

where W^0 is the set of all paths with $w(p) = 0$. A linear function can be applied to convert this imprecise clock period to a precise number by using a linear mapping with a positive coefficient $f : X \to R$, where X is an imprecise fuzzy number such as $\hat{d}(v)$, and R is a precise number such as $d(v)$.

It is shown in [KO95] that the optimal clock period obtained by performing a retiming on a circuit with precise delays given by $f(\hat{d}(v))$ is that same as that obtained by optimally retiming a circuit with fuzzy delays, $\hat{d}(v)$, and then converting the clock period to a precise number using f. Thus, it is only necessary to convert the imprecise delays, $\hat{d}(v)$, to precise delays using f, and run a standard retiming algorithm [LS91] to find the optimal solution under the imprecise delay model.

7.7 CONCLUSION

In this chapter, an overview of issues and applications of retiming in the design flow have been presented. The objective of this presentation has been to paint a broad picture, and the reader is invited to look into the references provided herein for greater details.

8 CONCLUSION

This book has attempted to provide an overview of algorithms for analysis and optimization of the timing behavior of sequential circuits. The main focus has been on efficient techniques for timing analysis, with an illustration of two major methods that are true sequential timing optimization methods: clock skew optimization and retiming.

Both of these methods are techniques with a somewhat socialistic ideology: they perform cycle borrowing and lend time from a timing-"rich" cycle that has enough to spare, and provide it to a timing-"poor" cycle that has many hungry gates to feed (the gates themselves, being incapable of speech or action, rely on the designer to carry out the class struggle on their behalf). The eventual goal of this Robin Hood scheme is to achieve an overall benefit to the circuit, taxing the timing-"rich" segments enough to arrive at an optimal solution. An equivalence between retiming and clock skew optimization has been shown and algorithmically exploited. An interesting point to note is that the use of level-clocked latches permits a limited amount of cycle borrowing across combinational boundaries, and this has been extensively used in the past. The method of retiming is also not an entirely new method, since designers have been moving latches to balance delays long before the term was coined. However, with the availability of efficient CAD algorithms, the ad hoc nature of a designer's optimization can be replaced by more efficient means that are more likely to provide the optimal circuit.

N. Maheshwari et al., *Timing Analysis and Optimization of Sequential Circuits*
© Kluwer Academic Publishers 1999

The optimization of a sequential circuit, of course, involves not only these steps, but also methods for optimizing the combinational segments. Within these segments, the application of methods such as logic synthesis, sizing, technology mapping, and the choice of circuit style, can provide further improvements. However, many conventional CAD techniques have focused purely on optimizing the combinational part, without adequate recognition that optimizing the sequential circuit as a whole can pay large dividends. This book is an effort to convince the researcher and CAD professional that sequential optimization methods are useful, and can be executed in a computationally cheap fashion. With timing constraints becoming more and more tight in the coming decade, there will be a strong need for true sequential optimizations that optimize not only within one cycle, but also across cycles, and the methods outlined in this book are likely to be extremely useful in this regard.

[AB98] H. Arts and M. Berkelaar. Combining logic synthesis and retiming. In *Workshop Notes, International Workshop on Logic Synthesis*, pages 136–139, 1998.

[ABF90] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design.* W. H. Freeman and Company, New York, NY, 1990.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows Theory, Algorithms and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1929–1934, 1989.

[BBR91] K. Bartlett, G. Borriello, and S. Raju. Timing optimization of multiphase sequential logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):51–62, January 1991.

[BC96] A. Balakrishnan and S. Chakradhar. Retiming with logic duplication transformation: Theory and an application to partial scan. In *Proceedings of the International Conference on VLSI Design*, pages 296–302, 1996.

[Ber92] M. R. C. M. Berkelaar. *LP_SOLVE User's Manual.* Eindhoven University of Technology, Eindhoven, The Netherlands, June 1992.

[BJS77] M. S. Bazaraa, J. J. Javis, and H.D. Sherali. *Linear Programming and Network Flows.* John Wiley, New York, NY, 1977.

[BK92] K. D. Boese and A. B. Kahng. Zero-skew clock routing trees with minimum wirelength. In *Proceedings of the IEEE International ASIC Conference*, pages 17–21, 1992.

[BSM94] T. M. Burks, K. A. Sakallah, and T. N. Mudge. Optimization of critical paths in circuits with level-sensitive latches. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 468–473, 1994.

[BSM95] T. M. Burks, K. A. Sakallah, and T. N. Mudge. Critical paths in circuits with level-sensitive latches. *IEEE Transactions on VLSI Systems*, 3(2):273–291, June 1995.

[BVBD97] L. Benini, P. Vuillod, A. Bogliolo, and G. De Micheli. Clock skew optimization for peak current reduction. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 16(2/3):117–130, June/July 1997.

[CD94] S. Chakradhar and S. Dey. Resynthesis and retiming for optimum partial scan. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 87–94, 1994.

[CDPR93] S. Chakradhar, S. Dey, M. Potkonjak, and S. G. Rothweiler. Sequential circuit delay optimization using global path delays. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 483–489, 1993.

[Cha93] L. F. Chao. *Scheduling and Behavioral Transformations for Parallel Systems.* PhD thesis, Department of Computer Science, Princeton University, Princeton, NJ, 1993.

[CHH92] T. H. Chao, Y. C. Hsu, and J. M. Ho. Zero-skew clock net routing. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 518–523, 1992.

[CHKM96] J. Cong, L. He, C. K. Koh, and P. H. Madden. Performance optimization of VLSI interconnect layout. *Integration: The VLSI Journal*, 21(1-2):1–94, November 1996.

[CL75] L. O. Chua and P.-M. Lin. *Computer-aided analysis of electronic circuits: Algorithms and computational techniques.* Prentice-Hall, 1975.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill, New York, NY, 1990.

[CMP+97] P. Y. Calland, A. Mignotte, O. Peyran, Y. Robert, and F. Vivien. Retiming DAGs. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 123–128, 1997.

[CN94] E. Chiprout and M. Nakhla. *Asymptotic Waveform Evaluation and Moment-Matching for Interconnect Analysis.* Kluwer Academic Publishers, Boston, MA, 1994.

[CS92] L. F. Chao and E. H.-M. Sha. Unfolding and retiming data-flow DSP programs for RISC multiprocessor scheduling. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 565–568, 1992.

[CS93a] L. F. Chao and E. H.-M. Sha. Efficient retiming and unfolding. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 421–424, 1993.

[CS93b] J. D. Cho and M. Sarrafzadeh. A buffer distribution algorithm for high-speed clock routing. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 537–543, 1993.

[CSH93] W. Chuang, S. S. Sapatnekar, and I. N. Hajj. A unified algorithm for gate sizing and clock skew optimization to minimize sequential circuit area. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 220–223, 1993.

[CSH95] W. Chuang, S. S. Sapatnekar, and I. N. Hajj. Timing and area optimization for standard cell VLSI circuit design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 308–320, March 1995.

[CW94] Y. P. Chen and D. F. Wong. On retiming for FPGA logic module minimization. In *Proceedings of the IEEE International Conference on Computer Design*, pages 394 –397, 1994.

[CW96] J. Cong and C. Wu. An improved algorithm for performance optimal technology mapping with retiming in LUT-based FPGA design. In *Proceedings of the IEEE International Conference on Computer Design*, pages 572–578, 1996.

[CW97] J. Cong and C. Wu. FPGA synthesis with retiming and pipelining for clock period minimization of sequential circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 644–649, 1997.

[CW98] J. Cong and C. Wu. Optimal FPGA mapping and retiming with efficient initial state computation. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 330–335, 1998.

[DB96] D. K. Das and B. B. Bhattachaya. Does retiming affect redundancy in sequential circuits? In *Proceedings of the International Conference on VLSI Design*, pages 260–263, 1996.

[DC94] S. Dey and S. Chakradhar. Retiming sequential circuits to enhance testability. In *Proceedings of the IEEE VLSI Test Symposium*, pages 28–33, 1994.

[DCJ96] M. P. Desai, R. Cvijetic, and J. Jensen. Sizing of clock distribution networks for high performance CPU chips. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 389–394, 1996.

[De 91] G. De Micheli. Synchronous logic synthesis: Algorithms for cycle time minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):63–73, January 1991.

[De 94] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, New York, NY, 1994.

[Deo94] R. B. Deokar. Clock period minimization using skew optimization and retiming. Master's thesis, Department of Electrical and Computer Engineering, Iowa State University, Ames IA, May 1994.

[DMQP94] F. Dartu, N. Menezes, J. Qian, and L. T. Pillage. A gate-delay model for high-speed CMOS circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 576–580, 1994.

[DPP95] Y. G. DeCastelo-Vide-e-Souza, M. Potkonjak, and A. Parker. Optimal ILP-based approach for throughput optimization using simultaneous algorithm/architecure matching and retiming. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 113–118, 1995.

[DPR92] S. Dey, M. Potkonjak, and S. G. Rothweiler. Performance optimization of sequential circuits by eliminating retiming bottlenecks. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 504–509, 1992.

[DS94] R. B. Deokar and S. S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1.407–1.410, 1994.

[DS95] R. B. Deokar and S. S. Sapatnekar. A fresh look at retiming via clock skew optimization. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 310–315, 1995.

[Dun92] P. Duncan *et al.* Hi-PASS: A computer-aided synthesis system for maximally parallel digital signal processing ASICS. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages V–605–608, 1992.

[Eck80] J. G. Ecker. Geometric programming: methods, computations and applications. *SIAM Review*, 22(3):338–362, July 1980.

[Eda91] M. Edahiro. Minimum skew and minimum path length routing in VLSI layout design. *NEC Research and Development*, 32:569–575, 1991.

[Eda93] M. Edahiro. Delay minimization for zero skew routing. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 563–566, 1993.

[EMRM95] A. El-Maleh, T. Marchok, J. Rajski, and W. Maly. On test set preservation of retimed circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 176–182, 1995.

[EMRM97] A. El-Maleh, T. Marchok, J. Rajski, and W. Maly. Behavior and testability preservation under the retiming transformation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(5):528–543, May 1997.

[ESS96] G. Even, I. Y. Spillinger, and L. Stok. Retiming revisited and reversed. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(3):348–357, March 1996.

[Eve94] G. Even. *Design of VLSI circuits using retiming.* PhD thesis, Israel Insistitute of Technology, Haifia, Israel, November 94.

[FD85] J.P. Fishburn and A.E. Dunlop. TILOS: A posynomial programming approach to transistor sizing. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 326–328, 1985.

[FF95] P. Feldmann and R. W. Freund. Efficient linear circuit analysis by Padé approximation via the Lanczos process. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(5):639–649, May 1995.

[Fis90] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, July 1990.

[FK82] A. L. Fisher and H. T. Kung. Synchronizing large systolic arrays. In *Proceedings of the SPIE*, volume 341, pages 44–52, May 1982.

[Fri95] E. G. Friedman, editor. *Clock distribution networks in VLSI circuits and systems.* IEEE Press, New York, NY, 1995.

[Geb93] C. H. Gebotys. Throughput optimized architectural synthesis. *IEEE Transactions on VLSI Systems*, 1(3):254–261, 1993.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, San Francisco, California, 1979.

[GLC94] C. T. Gray, W. Liu, and R. K. Cavin, III. *Wave Pipelining: Theory and CMOS Implementation.* Kluwer Academic Publishers, Boston, MA, 1994.

[GLH+94] J. Grodstein, E. Lehman, H. Harkness, H. Touati, and B. Grundmann. Optimal latch mapping and retiming within a tree. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 242–245, 1994.

[Has97] S. Hassoun. *Architectural Retiming: A Technique for Optimizing Latency-Constrained Circuits.* PhD thesis, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1997.

[HB84] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, NY, 1984.

[HCC96] S. Y. Huang, K. T. Cheng, and K. C. Chen. On verifying the correctness of retimed circuits. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 277–280, 1996.

[HCC97] S. Y. Huang, K. T. Cheng, and K. C. Chen. AQUILA: An equivalence verifier for large sequential circuits. In *Proceedings of the Asian and South Pacific Design Automation Conference*, pages 455–460, 1997.

[HCCG96] S. Y. Huang, K. T. Cheng, K. C. Chen, and U. Glaeser. An ATPG-based framework for verifying sequential equivalence. In *Proceedings of the IEEE International Test Conference*, pages 865–874, 1996.

[HE95] S. Hassoun and C. Ebeling. Architectural retiming: An overview. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 27–38, 1995.

[HE96] S. Hassoun and C. Ebeling. Architectural retiming: Pipelining latency-constrained circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 708–713, 1996.

[HE97a] S. Hassoun and C. Ebeling. Experiments in the iterative application of resynthesis and retiming. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 164–169, 1997.

[HE97b] S. Hassoun and C. Ebeling. An overview of prediction-based architectural retiming. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 39–48, 1997.

[Hit82] R. B. Hitchcock, Sr. Timing verification and the timing analysis program. In *Proceedings of the 19th Design Automation Conference*, pages 594–604, 1982.

[HKK95] Y. Higami, S. Kajihara, and K. Kinoshita. Test sequence compaction by reduced scan shift and retiming. In *Proceedings of the IEEE Asian Test Symposium*, pages 169–175, 1995.

[HKK96] Y. Higami, S. Kajihara, and K. Kinoshita. Partially parallel scan chain for test length reduction by using retiming technique. In *Proceedings of the IEEE Asian Test Symposium*, pages 94–99, 1996.

[HLFC97] H.-Y. Hsieh, W. Liu, P. Franzon, and R. Cavin III. Clocking optimization and distribution in digital systems with scheduled skews. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 16(2/3):131–148, June/July 1997.

[HW96] M. Heshami and B. A. Wooley. A 250-MHz skewed-clock pipelined data buffer. *IEEE Journal of Solid-State Circuits*, 31(3):376–383, March 1996.

[ILP92] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou. Optimizing two-phase, level-clocked circuitry. In *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, pages 246–264, 1992.

[IP96] A. T. Ishii and M. C. Papaefthymiou. Efficient pipelining of level-clocked circuits with min-max propagation delays. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 39–51, 1996.

[Ish93] A. T. Ishii. Retiming gated-clocks and precharged circuit structures. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 300–307, 1993.

[JC93] D. Joy and M. Ciesielski. Clock period minimization with wave pipelining. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(4):461–472, April 1993.

[JSK90] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh. Clock routing for high-performance IC's. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 573–579, 1990.

[KC66] T. I. Kirkpatrick and N. R. Clark. PERT as an aid to logic design. *IBM Journal of Research and Development*, 10(2):135–141, March 1966.

[KCR91] A. Kahng, J. Cong, and G. Robins. High-performance clock routing based on recursive geometric matching. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 322–327, 1991.

[KGV83] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[KO95] I. Karkowski and R. H. J. M. Otten. Retiming synchronous circuitry with imprecise delays. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 322–326, 1995.

[KR95] A. B. Kahng and G Robins. *On Optimal Interconnections for VLSI.* Kluwer Academic Publishers, Boston, MA, 1995.

[KT93] D. Kagaris and S. Tragoudas. Partial scan with retiming. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 249–254, 1993.

[KT94] D. Kagaris and S. Tragoudas. Retiming algorithm with application to VLSI testability. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 216–221, 1994.

[KT96] D. Kagaris and S. Tragoudas. Retiming-based partial scan. *IEEE Transactions on Computers*, pages 74–87, January 1996.

[KTB93] D. Kagaris, S. Tragoudas, and D. Bhatia. Pseudo-exhaustive BIST for sequential circuits. In *Proceedings of the IEEE International Conference on Computer Design*, pages 523–527, 1993.

[KTB95] D. Kagaris, S. Tragoudas, and D. Bhatia. Pseudo-exhaustive BIST for sequential circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(9):1160–1171, September 1995.

[Lal96] K. N. Lalgudi. *Architecural-Level Design of High-Performance, Energy-Efficient VLSI Systems.* PhD thesis, Yale University, New Haven, CT, December 1996.

[LE91] B. Lockyear and C. Ebeling. Optimal retiming of level-clocked circuits using symmetric clock schedules. Technical Report UW-CSE-91-10-01, Department of Computer Science and Engineering, University of Washington, Seattle, 1991.

[LE92] B. Lockyear and C. Ebeling. Optimal retiming of multi-phase level-clocked circuits. In *Advanced Research in VLSI and Parallel Systems: Procedings of the 1992 Brown/MIT Conference*, pages 265–280, 1992.

[LE94] B. Lockyear and C. Ebeling. Optimal retiming of level-clocked circuits using symmetric clock schedules. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1097–1109, September 1994.

[Lin93] B. Lin. Restructuring of synchronous logic circuits. In *Proceedings of the European Design Automation Conference*, pages 205–209, 1993.

[LKA95a] S. Lejmi, B. Kaminska, and B. Ayari. Retiming for BIST-sequential circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1740–1742, 1995.

[LKA95b] S. Lejmi, B. Kaminska, and B. Ayari. Retiming, resynthesis, and partitioning for the pseudo-exhaustive testing of sequential circuits. In *Proceedings of the IEEE VLSI Test Symposium*, pages 434–439, 1995.

[LKA95c] S. Lejmi, B. Kaminska, and B. Ayari. Synthesis and retiming for the pseudo-exhaustive BIST of synchronous sequential circuits. In *Proceedings of the IEEE International Test Conference*, pages 683–692, 1995.

[LKW93] S. Lejmi, B. Kaminska, and E. Wagneur. Resynthesis and retiming of synchronous sequential circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1674–1677, 1993.

[LP95a] K. N. Lalgudi and M. Papaefthymiou. DELAY: An efficient tool for retiming with realistic delay modeling. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 304–309, 1995.

[LP95b] K. N. Lalgudi and M. C. Papaefthymiou. Efficient retiming under a general delay model. In *Advanced Research in VLSI : the 1995 MIT/UNC-Chapel Hill Conference*, 1995.

[LP96] K. N. Lalgudi and M. Papaefthymiou. Fixed-phase retiming for low power. In *Proceedings of the International Symposium of Low Power Electronics and Design*, pages 259–264, 1996.

[LRS83] C. Leiserson, F. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. In *Proceedings of the 3rd Caltech Conference on VLSI*, pages 87–116, 1983.

[LS91] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[LS96] D. Lehther and S. S. Sapatnekar. Clock tree synthesis for multichip modules. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 53–56, 1996.

[LS98] D. Lehther and S. S. Sapatnekar. Moment-based techniques for rlc clock tree construction. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(1):69–79, January 1998.

[LSC+93] L.-T. Liu, M. Shih, N.-C. Chou, C.-K. Cheng, and W. Ku. Performance-driven partitioning using retiming and replication. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 296–299, 1993.

[LTW94] J.-f. Lee, D. T. Tang, and C. K. Wong. A timing analysis algorithm for circuits with level-sensitive latches. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 743–748, 1994.

[LVW97] C. Legl, P. Vanbekbergen, and A. Wang. Retiming of edge-triggered circuits with multiple clocks and load enables. In *Workshop Notes, International Workshop on Logic Synthesis*, 1997.

[LW83] Y. Z. Liao and C. K. Wong. An algorithm to compact a VLSI symbolic layout with mixed constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 3(1):130–133, April 1983.

[Mah98] N. Maheshwari. *Fast Algorithms for Retiming Large Digital Circuits.* PhD thesis, Department of Electrical and Computer Engineering, Iowa State University, Ames IA, 1998.

[Mar96] H.-G. Martin. Retiming for circuits with enable registers. In *Proceedings of EUROMICRO-22*, pages 275–280, 1996.

[MB91] P. C. McGeer and R. K. Brayton. *Integrating Functional and Temporal Domains in Logic Design.* Kluwer Academic Publishers, Boston, MA, 1991.

[MDG93] J. Monteiro, S. Devadas, and A. Ghosh. Retiming sequential circuits for low power. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 398–402, 1993.

[ME87] D. Marple and A. El Gamal. Optimal selection of transistor sizes in digital VLSI circuits. In *Stanford Conference on VLSI*, pages 151–172, 1987.

[MEMR96] T. Marchok, A. El-Maleh, W. Maly, and J. Rajski. A complexity analysis of sequential ATPG. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(11):1409–1423, November 1996.

[MH91] A. Münzner and G Hemme. Converting combinational circuits into pipelined data paths. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 368–371, 1991.

[MS96] N. Maheshwari and S. S. Sapatnekar. A practical algorithm for retiming level-clocked circuits. In *Proceedings of the IEEE International Conference on Computer Design*, pages 440–445, 1996.

[MS97a] N. Maheshwari and S. S. Sapatnekar. An improved algorithm for minimum-area retiming. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 2–7, 1997.

[MS97b] N. Maheshwari and S. S. Sapatnekar. Minimum area retiming with equivalent initial states. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 216–219, 1997.

[MS98a] N. Maheshwari and S. S. Sapatnekar. Efficient minarea retiming for large level-clocked circuits. In *Proceedings of the Conference on Design Automation and Test in Europe*, pages 840–845, 1998.

[MS98b] N. Maheshwari and S. S. Sapatnekar. Efficient retiming of large circuits. *IEEE Transactions on VLSI Systems*, pages 74–83, March 1998.

[MSBS90] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. In *Proceedings of the 23rd Anual Hawaii International Conference on System Sciences*, pages 397–406, 1990.

[MSBS91] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):74–84, January 1991.

[MSBS93] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli. Performance optimization of pipelined logic circuits using peripheral retiming and resynthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(5):568–578, May 1993.

[NF96] J. L. Neves and E. G. Friedman. Design methodology for synthesizing clock distribution networks exploiting nonzero localized clock skew. *IEEE Transactions on VLSI Systems*, 4(2):286–291, June 1996.

[NF97] J. L. Neves and E. G. Friedman. Buffered clock tree synthesis with non-zero clock skew scheduling for increased tolerance to process parameter variation. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 16(2/3):149–162, June/July 1997.

[OP97] A. Odabasioglu and L. T. Pileggi. PRIMA: Passive reduced-order interconnect macromodeling algorithm. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 58–65, 1997.

[Pan97] P. Pan. Continuous retiming: Algorithms and applications. In *Proceedings of the IEEE International Conference on Computer Design*, pages 116–121, 1997.

[Pap90] M. C. Papaefthymiou. On retiming synchronous circuitry and mixed-integer optimization. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, September 1990.

[Pap93] M. C. Papaefthymiou. *A Timing Analysis and Optimzation System for Level-Clocked Circuitry.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1993.

[PL95] P. Pan and C. L. Liu. Optimal clock period technology mapping for FPGA circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 720–725, 1995.

[PL96a] P. Pan and C. L. Liu. Optimal clock period FPGA technology mapping for sequential circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 720–725, 1996.

[PL96b] P. Pan and C. L. Liu. Technology mapping for sequential circuit for LUT-based FPGAs for performance. In *ACM/SIGDA International Conference on Field Programmable Gate Arrays*, pages 58–64, 1996.

[PMOP93] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage. Skew and delay optimization for reliable buffered clock trees. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 556–562, 1993.

[PMP93] S. Pullela, N. Menezes, and L. T. Pillage. Reliable nonzero clock skew trees using wire width optimization. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 165–170, 1993.

[PR90] L. T. Pillage and R. A. Rohrer. Asymptotic waveform evaluation for timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(4):352–366, April 1990.

[PR91] M. Potkonjak and J. Rabaey. Optimizing resource utilization using transformations. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 88–91, 1991.

[PR92] M. Potkonjak and J. Rabaey. Fast implementation of recursive programs using transformations. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 304–308, 1992.

[PR93a] M. C. Papaefthymiou and K. H. Randall. Edge-triggered vs. two-phase level-clocking. In *Research on Integrated Systems: Proceedings of the 1993 Symposium*, March 1993.

[PR93b] M. C. Papaefthymiou and K. H. Randall. TIM: A timing package for two-phase, level-clocked circuitry. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 497–502, 1993.

[PS96] N. L. Passos and E. H.-M. Sha. Achieving full parallelism using multidimensional retiming. *IEEE Transactions on Parallel and Dsitributed Systems*, 7(11):1150–1163, November 1996.

[PSAB94] C. Pixley, V. Singhal, A. Aziz, and R. K. Brayton. Multi-level synthesis for safe replaceability. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 442–449, 1994.

[PSB96] N. L. Passos, E. H.-M. Sha, and S. C. Bass. Optimizing DSP flow graphs via schedule-based multidimensional retiming. *IEEE Transactions on Signal Processing*, 44(1):150–155, January 1996.

[QPP94] J. Qian, S. Pullela, and L. T. Pillage. Modeling the effective capacitance of RC interconnect of CMOS gates. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(12):1526–1535, December 1994.

[Ran93] K. H. Randall. Edge-triggered vs. level-clocking. Bachelor's thesis, Massachusetts Institute of Technology, 1993.

[Ran97] R. Ranjan. *Design and Implementation Verification of Finite State Systems.* PhD thesis, Department of Computer Science and Engineering, University of California, Berkeley, December 1997.

[RPH83] J. Rubenstein, P. Penfield, and M. A. Horowitz. Signal delay in RC tree networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-2(3):202–211, July 1983.

[RSSB98] R. K. Ranjan, V. Singhal, F. Somenzi, and R. K. Brayton. Using combinational verification for sequential circuits. In *Workshop Notes, International Workshop on Logic Synthesis*, pages 290–298, 1998.

[San89] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, January 1989.

[Sap96] S. S. Sapatnekar. Efficient calculation of all-pair input-to-output delays in synchronous sequential circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages IV.520–IV.523, 1996.

[Sax85] J. B. Saxe. *Decomposable Searching Problems and Circuit Optimization by Retiming: Two Studies in General Transformations of Computational Structures.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1985.

[SBS91] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming of circuits with single phase transparent latches. In *Proceedings of the IEEE International Conference on Computer Design*, pages 86–89, 1991.

[SBS92] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Graph algorithms for clock schedule optimization. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 132–136, 1992.

[SBS93] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Minimum padding to satisfy short path constraints. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 156–161, 1993.

[SD96] S. S. Sapatnekar and R. B. Deokar. Utilizing the retiming skew equivalence in a practical algorithm for retiming large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(10):1237–1248, October 1996.

[SDK83] M. M. Syslo, N. Deo, and J. S. Kowalik. *Discrete Optimization Algorithms.* Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[Sem97] Semiconductor Industry Association. The national technology roadmap for semiconductors, 1997.

[SF94] T. Soyata and E. G. Friedman. Retiming with non-zero clock skew, variable register and interconnect delay. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 234–241, 1994.

[SFM93] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr. Integration of clock skew and register delays into a retiming algorithm. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1483–1486, 1993.

[SFM97] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr. Incorporating internconnect, register and clock distribution delays into the retiming process. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(1):165–120, January 1997.

[She93] N. Shenoy. *Timing Issues in sequential circuits.* PhD thesis, Department of Computer Science and Engineering, University of California, Berkeley, November 1993.

[She95] N. Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer Academic Publishers, Boston, MA, 2 edition, 1995.

[She97] N. Shenoy. Retiming: Theory and practice. *Integration, the VLSI Journal*, 22(1):1–21, January 1997.

[Sin96] V. Singhal. *Design Replacements for Sequential Circuits.* PhD thesis, Department of Computer Science and Engineering, University of California, Berkeley, March 1996.

[SK93] S. S. Sapatnekar and S. M. Kang. *Design Automation for Timing-Driven Layout Synthesis.* Kluwer Academic Publishers, Boston, MA, 1993.

[SMB96] V. Singhal, Sharad Malik, and R. K. Brayton. The case for retiming with explicit reset circuitry. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 618–625, 1996.

[SMO90] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. checkTc and minTc: Timing verification and optimal clocking of synchronous digital circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 552–555, 1990.

[SMO92] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(3):322–333, March 1992.

[SPB95] V. Singhal, C. Pixley, and R. K. Brayton. Power-up delay for retiming digital circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 566–569, 1995.

[SPRB95] V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton. The validity of retiming sequential circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 316–321, 1995.

[SR94] N. Shenoy and R. Rudell. Efficient implementation of retiming. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 226–233, 1994.

[SRVK93] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S. M. Kang. An exact solution to the transistor sizing problem for CMOS circuits using convex optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(11):1621–1634, November 1993.

[SS92] T. G. Szymanski and N. Shenoy. Verifying clock schedules. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 124–131, 1992.

[SSBS92] N. Shenoy, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli. On the temporal equivalence of sequential circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 405–409, 1992.

[SSE95] I. Y. Spillinger, L. Stok, and G. Even. Improving initialization through reversed retiming. In *Proceedings of the European Design and Test Conference*, pages 150–154, 1995.

[SSF95] H. Sathyamurthy, S. S. Sapatnekar, and J. P. Fishburn. Speeding up pipelined circuits through a combination of gate sizing and clock skew optimization. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 467–470, 1995.

[SSF98] H. Sathyamurthy, S. S. Sapatnekar, and J. P. Fishburn. Speeding up pipelined circuits through a combination of gate sizing and clock skew optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(2):173–182, February 1998.

[SSFD88] J.-M. Shyu, A. L. Sangiovanni-Vincentelli, J. P. Fishburn, and A. E. Dunlop. Optimization-based transistor sizing. *IEEE Journal of Solid-State Circuits*, 23(2):400–409, April 1988.

[SSWN97] S. Simon, C. V. Schimpfle, M. Wroblewski, and J. A. Nossek. Retiming of latches for power reduction of DSP design. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2168–2171, 1997.

[Szy92] T. G. Szymanski. Computing optimal clock schedules. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 399–404, 1992.

[TB93] H. J. Touati and R. K. Brayton. Computing the initial states of retimed circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(1):157–162, January 1993.

[TS97] G. E. Tellez and M. Sarrafzadeh. Minimal buffer insertion in clock trees with skew and slew rate constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(4):332–342, April 1997.

[Tsa91] R.-S. Tsay. Exact zero skew. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 336–339, 1991.

[Tsa93] R.-S. Tsay. An exact zero-skew clock routing algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(2):242–249, February 1993.

[TSS92] H. Touati, N. V. Shenoy, and A. L. Sangiovanni-Vincentelli. Retiming for table-lookup field programmable gate arrays. In *FPGA*, pages 89–94, 1992.

[VBBD96] P. Vuillod, L. Benini, A. Bogliolo, and G. De Micheli. Clock skew optimization for peak current reduction. In *Proceedings of the International Symposium of Low Power Electronic Devices*, pages 265–270, 1996.

[VCL+96] K. Venkat, L. Chen, I. Lin, P. Mistry, and P. Madhani. Timing verification of dynamic circuits. *IEEE Journal of Solid-State Circuits*, 31(3):452–455, March 1996.

[VHBM96] A. Vittal, H. Ha, F. Brewer, and M. Marek-Sadowska. Clock skew optimization for ground bounce control. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 395–399, 1996.

[VMS95] A. Vittal and M. Marek-Sadowska. Power optimal buffered clock tree design. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 497–502, 1995.

[VMS96] D. Van Campenhout, T. Mudge, and K. A. Sakallah. Timing verification of sequential domino circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 127–132, 1996.

[vVA+94] A. van der Werf, J. L. Van Meerbergen, E. H. L. Aarts, W. F. J. Verhaegh, and P. E. R. Lippens. Efficient timing constraint derivation for optimally retiming high speed processing units. In *International Symposium on High Level Synthesis*, pages 48–53, 1994.

[Wag88] K. D. Wagner. Clock system design. *IEEE Design and Test of Computers*, pages 9–27, October 1988.

[WBL+94] N. Wehn, J. Biesenack, T. Langmaier, M. Munch, M. Pilsl, S. Rumler, and P. Duzy. Scheduling of behavioral VHDL by retiming techniques. In *Proceedings of the European Design Automation Conference*, pages 546–551, 1994.

[WE93] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design.* Addison-Wesley, Reading, MA, 2nd edition, 1993.

[WR93] U. Weinmann and W. Rosenstiel. Techology mapping for sequential circuits based on retiming techniques. In *Proceedings of the European Design Automation Conference*, pages 318–323, 1993.

[Wya87] J. L. Wyatt, Jr. Signal propagation delay in RC models for interconnect. In A. E. Ruehli, editor, *Circuit Analysis, Simulation and Design*, pages 254–291. North-Holland, 1987.

[XD97] J. G. Xi and W. W.-M. Dai. Useful-skew clock routing with gate sizing for low power design. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 16(2/3):163–180, June/July 1997.

[XK95] T. Xue and E. S. Kuh. Post routing performance optimization via multi-link insertion and non-uniform wiresizing. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 575–579, 1995.

[YC91] C.-W. Yeh and C.-K. Cheng. A general purpose multiple way partitioning algorithm. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 421–426, 1991.

[YH97] H. Yalcin and J. P. Hayes. Event propagation conditions in circuit delay computation. *ACM Transactions on Design Automation of Electronic Systems*, 2(3):249–280, July 1997.

[YKK95a] H. Yotsuyanagi, S. Kajihara, and K. Kinoshita. Resynthesis for sequential circuits designed with a specified initial state. In *Proceedings of the IEEE VLSI Test Symposium*, pages 152–157, 1995.

[YKK95b] H. Yotsuyanagi, S. Kajihara, and K. Kinoshita. Synthesis for testability by sequential redundancy removal using retiming. In *International Symposium on Fault Tolerent Computing*, pages 33–40, 1995.

[ZS98] M. Zhao and S. S. Sapatnekar. Timing optimization of mixed static and domino logic. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1998.

Index

ASAP and ALAP retiming, 85

Bellman-Ford algorithm, 25, 43, 44
    Predecessor heuristic, 27, 45

checkTc, 25
Clock period, 9
Clock schedule optimization, 26
Clock skew optimization, 3, 33, 40, 46
    relation with retiming, 77, 91
Clock tree construction, 54, 59
    fixed skew routing, 39
    geometric methods, 36
    zero skew routing, 36
Clocking disciplines, 21
Constraint graph, 27, 42, 45
Control logic, 68, 70, 123
Convexity, 55
CPM, *see* Critical path method
Critical path, 14, 24
Critical path method, 13–15, 30, 56, 63
Cycle borrowing, *see* Cycle stealing
Cycle stealing, 20

Deferred merge embedding, 38, 59
Delay modeling, 10, 165
DME, *see* Deferred merge embedding
Domino logic, 27
Double-clocking, *see* Short path constraint

Edge-triggered circuits, 68, 69
Edge-triggered flip-flop's, *see* Flip-flops
Elmore delay, 11, 37, 55

False paths, 15
Field programmable gate arrays, *see* FPGA's
Finite state machines, 68, 70, 123
Flip-flops, 8, 17

FPGA's, 69, 161

Gate delay models, 10

Hold time, 9, 17, 41
Hold time constraint, *see* Short path constraint

Interconnect effects, 3, 165

Latch shift operator, 91
Latches, 8, 20
Level-clocked circuits, 68, 70, 88, 113
    multiphase, 91
    single-phase, 89
    two-phase, 90
Level-sensitive latches, *see* Latches
Linear program, 26
    ILP, 62
    MILP, 73
    minimum cost network flow, 100, 109, 112
Linear programming problem, 40, 47
Logic signal separation constraints, 61
Logic synthesis, 4, 67, 156, 157, 161
Long path constraint, 17, 41, 55, 56

MILP, 137
Minarea retiming, 66, 99
    edge-triggered circuits, 105
    level-clocked circuits, 113
Minaret, 105
Minimum padding, 18
Minperiod retiming, 66
    edge-triggered circuits, 74
    level-clocked circuits, 94
minTc, 26
Mirror vertices, 101

Network flow, *see* Linear program

Partitioning, 50
Peak current reduction, 62
PERT, *see* Critical path method, 14
Phase shift operator, 22, 91
Pipelines, 7
Posynomials, 55, 63
Power optimization, 67, 71, 156

RC modeling, 10, 58
Reduced order modeling, 11
Redundancies
    sequential and combinational, 150
Reset circuitry, 144
Resynthesis, *see* Logic synthesis
Retiming, 3, 65
    architectural, 159
    constraint reduction, 106, 115, 133
    continuous, 87
    DAG's, 162
    flip-flop sharing, 101, 137, 141
    for a target period, 84
    for power, 67, 70
    gated clocks and precharged logic, 163
    Leiserson-Saxe approach, 69, 100
    minarea initial state, 124
    minperiod initial state, 124
    peripheral, 158
    relation to test, 71, 128, 133, 148
    relation with clock skew, 77, 91
    with load enables, 129, 162
Reverse retiming, 127, 136

Setup time, 9, 17, 41
Setup time constraint, *see* Long path constraint
Short path constraint, 18, 35, 41, 55
Simulated annealing, 59
Skew tolerance, 42
Slack stealing, *see* Cycle stealing
SMO clock model, 21, 91
SMO formulation, 91
Symbolic constraint propagation, 16, 48

Tightness ratio, 52
Time borrowing, *see* Cycle stealing
Timing verification, 24
Transistor sizing, 2, 46

Verification
    safe replacement, 154
    temporal equivalence, 155

Wave pipelining, 60

Zero-clocking, *see* Long path constraint