INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Incorporating Circuit Level Information into the Retiming Process

by

Tolga Soyata

Submitted in Partial Fulfillment
of the
Requirements for the Degree
Doctor of Philosophy

Supervised by
Professor Eby G. Friedman

Department of Electrical and Computer Engineering
The College
School of Engineering and Applied Sciences

University of Rochester
Rochester, New York

1999

UMI Number: 9961367

Copyright 2000 by
Soyata, Tolga

All rights reserved.

UMI Microform 9961367
Copyright 2000 by Bell & Howell Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Dedication

This dissertation is dedicated to my country Turkey, where my soul belongs to. Currently, the sum of the market capitalizations of all Turkish companies in the Istanbul bourse is less than Intel's market cap. My goal is to see many Intel's in the Istanbul bourse before my last breath.

Curriculum Vitae

The author was born in Istanbul, Turkey on January 23, 1967. He attended Istanbul Technical University from 1984 to 1988 and graduated with a Bachelor of Science degree in Electrical Engineering. He received his Master of Science degree in Electrical and Computer Engineering from Johns Hopkins University, Baltimore, Maryland in 1992. He began further graduate studies in the Electrical and Computer Engineering field at the University of Rochester in 1992. His research interests include sequential circuit optimization by using pipelining, retiming, and clock scheduling techniques. He implemented a retiming algorithm that incorporates low-level circuit issues such as clock delays, variable register delays, and interconnect delays into the retiming process. He has published four conference papers and one journal paper. He pursued his research in the field of high performance VLSI design under the direction of Professor Eby G. Friedman.

Acknowledgements

I thank the Turkish government for sponsoring me throughout my Master's studies at Johns Hopkins University. I also thank the University of Rochester for sponsoring me during the early phase of my Ph.D. degree.

I am very thankful to the government of the United States for providing the infrastructure for my business. Without such an infrastructure, I would not be able to start my business and/or make such progress. Although I am a successful entrepreneur, I am very well aware that I would not be able to efficiently start and grow a business if the infrastructure was not there. It is clear to me that Soyata Computers is not only the success story of Tolga Soyata, but also just another success story of the United States.

Between 1981 and 1984, I was at Ankara Fen Lisesi (Ankara Scientific High School) — AFL. I owe a very strong math and science background to AFL and my teachers at AFL. The foundation I built up at AFL will be with me for the rest of my life. I cannot thank my teachers at AFL enough.

Between 1984 and 1988, I built up a very strong background at Istanbul Tech University (ITU). I cannot thank my professors enough since I owe the strong fundamental knowledge base in Electrical and Computer Engineering to them. I thank my advisor Prof. Hakan Kuntman, who always encouraged me. Prof. Ahmet Dervisoglu and Prof. Ahmet Kayran have always been great role models for me. I thank the Computer Science Department, especially Prof. Esref Adali, for encouraging me to get involved in

Computer Science.

Between 1990 and 1992, I further strengthened my Electrical and Computer Engineering background at the Johns Hopkins University (JHU). JHU is where I developed a strong practical knowledge base. Prof. Andrew Goldfinger taught me how to apply theoretical DSP knowledge to real life. Prof. Andreas Andreou interested his entire class in silicon retinas and analog VLSI. Professors at the Computer Science Department helped me build an invaluable computer science background.

The seven years I spent at the University of Rochester taught me both research and business. The company I started seven years ago became a success story in Rochester. I thank all of my employees for making it possible for me to continue to study for my Ph.D. while I was running the company. I also thank my lab mates, Brian Cherkauer, Victor Adler, and Ivan Kourtev, for sharing their valuable views with me.

I thank Prof. Kevin Parker for always believing in my business and supporting me. I also thank him for recommending me to the e-commerce committee and to the scientific editor of the UR newspaper, Tom Trickey. I thought I would fail my Ph.D. many times due to my decision of starting a business. I might not be leaving the University of Rochester with an exceptional Ph.D. that Prof. Friedman initially expected from me, however, it is an honor to see that both Prof. Friedman and Prof. Kevin Parker support my decision of choosing the business world over the academic world.

I thank Professor Eby G. Friedman for being a father, a friend, and a mentor for me.

I also thank him for being very patient with me. The lessons he taught me will help me for the rest of my life. I already started using his soft-but-hard approach towards my own employees. Although I disagreed with Prof. Friedman at times, I am amazed with the same response I am getting from my employees. I will always use his bottom-line oriented approach and zero tolerance for mistakes during my career. Prof. Friedman went far beyond teaching me academics, but changed my life once and forever.

Abstract

The advances in CMOS technology over the past decades have created the need for the design of extremely complex Very Large Scale Integration (VLSI) Integrated Circuits (IC)s. The simultaneous progress in Computer-Aided Design (CAD) tools enable large design teams to work independently on sophisticated VLSI projects. The pipelining process is widely utilized in VLSI ICs as a performance enhancement tool. Efficient automated pipelining algorithms have been developed to permit the application of pipelining to large integrated circuits. The emerging technique of retiming, on the other hand, has not found its way into the VLSI circuit design process due to complex and non-practical algorithms. Therefore, algorithms and/or methodologies to help achieve retiming with simpler and practical algorithms can significantly improve the acceptance of retiming techniques in standard VLSI design methodologies.

A retiming methodology is presented in this dissertation to model low-level circuit characteristics in VLSI ICs. This objective is achieved by modeling low level circuit parameters using the Register Electrical Characteristic (REC) model. The path delays in a VLSI IC are defined from register-to-register based on this REC model. The REC model forms the core of the retiming algorithms introduced in this dissertation. The REC model, for the first time, permits incorporating low-level circuit issues into the retiming process, thereby yielding significantly more accurate retiming results than the existing retiming algorithms currently described in the literature. Path monotonicity constraints

have been developed to permit the application of standard Linear Programming (LP) based techniques to the general retiming process. These monotonicity constraints permit circuits to be retimed with low-level characteristics with significantly less CPU time complexity.

Although the application of retiming to practical circuits has not as yet become common place, the research described in this dissertation is a significant improvement in making retiming into a practical and useful design methodology. The relationship between clock scheduling and retiming is also discussed in this dissertation where it is shown that the two processes are inextricably intertwined. The results of applying retiming to benchmark circuits have demonstrated performance improvements of up to 50%. When clock scheduling techniques are combined with retiming techniques while including low-level circuit characteristics, retiming can significantly improve the design efficiency and performance of the next generation VLSI circuits.

Table of Contents

Chapter 1 Introduction 1

Chapter 2 Theoretical Background 6

2.1 Terms Related to Synchronous Circuits 7

2.2 Representing Synchronous Circuits Using Graphs 12

2.2.1 Background Information on Graph Theory 12

2.2.2 Synchronous Circuits Represented as Graphs 13

2.3 Clocking of the Synchronous Circuits 15

2.4 Formulation of the Synchronous Circuit Optimization Problem 17

2.4.1 Timing Constraints 18

2.4.1.1 Edge Weight Constraints 19

2.4.1.2 Long Path Constraints 20

2.4.1.3 Short Path Constraints 21

2.4.1.4 Vertex Lags 21

2.4.2 Linear Programming 23

2.5 Algorithms for Synchronous Circuit Optimization 24

2.5.1 Notation for Algorithmic Complexity 24

2.5.2 Topological Sort 25

2.5.3 Shortest Paths 28

2.5.3.1 Single Source Shortest Paths 29

2.5.3.2 All Pairs Shortest Paths 29

2.5.3.3 Floyd-Warshall Method 30

2.5.3.4 The Bellman-Ford Algorithm 33

2.5.3.4.1 Relaxation 34

2.5.3.4.2 Using Bellman-Ford to Solve the Single-Source Shortest Paths Problem 36

2.5.3.4.3 Using Bellman-Ford to Solve Linear Programs 39

2.5.4 Branch and Bound Algorithms 40

Chapter 3 Synchronous VLSI Circuit Optimization Techniques 43
3.1 Pipelining of Synchronous Digital Systems 44
3.1.1 Pipelining of Combinatorial Circuits 44
3.1.2 Pipelining of Microprocessors 47
3.1.3 Pipelining of DSPs 49
3.1.4 Wave-Pipelining 50
3.2 Retiming Techniques for Sequential Circuit Optimization . . 51
3.2.1 Overview of the Retiming Process 52
3.2.2 More Recent Work in the Field of Retiming 57
Chapter 4 Register Electrical Characteristics (RECs) 60
4.1 The Importance of an Effective Timing Model 61
4.2 Register Delays 63
4.2.1 T_{Set-up}, T_{Hold}, $T_{C \rightarrow Q}$, and T_{CD} 63
4.2.2 Estimating T_{Set-up}, T_{Hold}, and $T_{C \rightarrow Q}$ 66
4.2.3 Estimating T_{CD} from the Layout 69
4.3 Interconnect Delays 72
4.3.1 T_{Int1} and T_{Int2} 72
4.3.2 T_{Int} Calculated from the Layout 73
4.3.3 Load-Dependant Model for Interconnect Delays . . . 76
4.4 Non-Ideal Logic and Register Delays 77
4.4.1 Model of Logic Delay 79
4.4.2 Model of Non-Uniform Logic Element Delays 80
4.5 Modeling the RECs 83
4.5.1 Basic REC Model (Level 1) 83
4.5.2 The Enhanced REC Model (Level 2) 86
4.6 Path Delays 89
4.6.1 Path Delays using REC Level 1 Model 91
4.6.2 Path Delays using REC Level 2 Model 94
4.6.3 Short Paths 97
4.6.4 Internal Short and Long Paths 97

Chapter 5 Retiming with RECs 99

5.1 Incorporating REC's into the Retiming Process 103

5.2 Sequential Adjacency Matrix (SAM) 107

5.3 Timing Constraints 109

5.3.1 Negative edge weight constraints 110

5.3.2 Long path constraints 110

5.3.3 Short path constraints 112

5.3.4 Internal path constraints 112

5.3.5 Constraints due to vertex lags 115

5.4 Retiming Algorithm 116

5.4.1 RETSAM: Retiming Algorithm for Synchronous Circuits with Attached Electrical Information 117

5.4.2 CHECKCP: Clock Period Feasibility Check 118

5.4.3 SOLVELAGS: Determination of the Vertex Lags Using a Branch and Bound Approach 119

5.5 Path Delay Monotonicity Constraints 125

5.5.1 Intuition for the Monotonicity Constraints 126

5.5.2 Designing the Clock Distribution Network 129

5.5.3 Feasibility Check for the Clock Distribution Network 132

5.5.4 Derivation of the Monotonicity Constraints 134

5.6 Experimental Results 140

5.6.1 Application of RETSAM to Modified MCNC Benchmark Circuits 140

5.6.2 The Impact of Latency on the Retimed Clock Period 144

5.7 Conclusions 147

Chapter 6 Conclusions 150
Chapter 7 Future Work 154
  7.1 Studying the Impact of Pipelining Depth on Retimed Minimum Clock Period 156
  7.2 Pipelining Recursive DSP Circuit Architectures 159
    7.2.1 Theoretical Background on IIR Filter Characterization 160
    7.2.2 Clustered Look-Ahead Pipelining 161
    7.2.3 Scattered Look-Ahead Pipelining 162
    7.2.4 Minimum Denominator Multiplier (MDM) 164
Bibliography 166
Appendix A Publications 175

LIST OF TABLES

Table **Title** **Page**

1 Delay of a 1 bit adder for different input and output pairs using a 0.8 μm 5 V CMOS technology. Note the non-uniformity due to asymmetry in the logic paths within the adder. 81

2 Delay of a 4–bit adder for different input and output pairs using a 0.8 μm, 5 V CMOS technology. "x" denotes the delays that cannot be measured since a change at the specified input is not propagated to the specified output. 82

3 The SAM for the graph of Figure 33. Light shaded entries represent short paths, whereas dark shaded entries represent long paths for $c = 37$ tu. Unshaded entries denote permissible paths. 109

4 Example solution for $c = 24$. A single value is shown for equal lower and upper bounds. 124

5 Results of the application of the retiming algorithm to MCNC benchmark circuits 145

LIST OF FIGURES

Figure	Title	Page
1	A synchronous circuit with three registers. Data signals are moved from the input to the output at a constant rate (the clock period). Registers are indicated as "R," and the logic elements are indicated as "L."	8
2	Two sequentially adjacent registers (initial register R_i and final register R_f) together with the logic elements between the two registers. This path forms a local data path.	10
3	A synchronous circuit with two inputs and a single output. Some local data paths (LDP) and global data paths (GDP) are indicated on the circuit. Registers are indicated with a vertical bar along the edges.	11
4	A synchronous circuit with four logic gates (g1 through g4) and five synchronizing registers (r0 through r4).	11
5	A simple graph with an edge set E and a vertex set V.	13
6	The graph representation of Figure 4. Vertices are used to represent the logic elements and edges are used to represent the connection between the logic elements. Note the use of zero delay vertices to model the inputs and outputs of the circuit.	15
7	Due to the difference between the arrival time of the clock signal at the initial register (C_i) and at the final register (C_f), negative and positive clock skew is created between the registers depending on the lead/lag relationship between C_i and C_f.	17
8	Pseudocode of the Algorithm TS for the topological sort of a graph G.	26
9	Pseudocode of the algorithm VISIT for coloring every vertex in G recursively.	27
10	Pseudocode of the Floyd-Warshall algorithm.	32
11	The relaxation procedure. (a) a tighter bound is found after relaxation, (b) relaxation has no effect.	35

12 Pseudocode of $RELAX(e)$. $d[v]$ is updated according to the aforementioned procedure. 35

13 An example graph which does not contain a negative cycle. 37

14 Graph of Figure 13 with a negative cycle along the path $v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_0$. Note that the negative edge weight of e_8 causes the negative cycle. . 37

15 Pseudocode of the Bellman-Ford algorithm. 38

16 Pipelining breaks global data paths into local data paths with smaller delay so as to increase the data flow rate. 45

17 Pipelining of microprocessors: five primary operations of the microprocessor are pipelined to increase computational speed. 48

18 In wave-pipelining, successive data waves propagate through the logic elements, forming an effective pipeline. 51

19 a) The original graph introduced in and b) its retimed version. Note that the sequential latency is four clock periods in both cases. $r(v_n)$ denotes the lag of vertex n after retiming. 54

20 The lag function changes the edge weights while preserving the circuit function. Increasing the lag of a vertex by one has the effect of increasing the weights of all edges by one connected in front of this vertex and decreasing the weights of all edges by one connected behind this vertex. . 56

21 A typical edge-triggered master-slave flip flop is composed of two latches. The set-up time is the time required for the data to successfully latch into the first latch. The hold time is the duration during which the data at the input of the register must be stable after the arrival of the clock signal. The clock-to-Q time is the time required for the data to appear at the output of the flip flop upon arrival of the incoming clock signal. . . . 64

22 Timing diagrams of the flip-flop shown in Figure 21. The data signal must be stable for T_{Setup} before and T_{Hold} after the arrival of the clock signal. The valid data signal appears at the output of the flip flop $T_{Clock-to-Q}$ after the clock signal arrives. 65

23 A block diagram of an example integrated circuit layout. The chip area is assumed to be partitioned into regions of similar clock delay. 71

24 The pre-register and post-register interconnect delays are caused by the distributed RC impedance of the connections between the register and the logic elements. 73

25 A logic stage (or a register) connected to another logic stage or register through an interconnect. The 50% interconnect delay T_{Int} caused by the RC impedance of the interconnect and the load capacitance is calculated based on physical parameters of the interconnect 74

26 A CMOS NAND gate. Note that the delay of this element is dependent on the output load. The element has an output slope m characterizing this load dependence. 78

27 A path containing two edges and a vertex between the two edges. The REC values are attached to each edge to characterize the registers, interconnect, and the clock delays located along the edges. 85

28 The graph of Figure 27 with attached load dependent parameters. The values in the brackets are the enhanced REC values m and c. 87

29 C_{Int2} and C_L are the loads at the output of the initial register. The circle is the logic element (or vertex) with a delay of 4 tu, and load dependent m and c values of 0.5 and 1, respectively. 88

30 The path shown in Figure 27. The minimum and maximum path delays, $T_{PD_{min}}$, and $T_{PD_{max}}$ between the two edges are calculated using (4.27). . 93

31 The path shown in Figure 28. The path delay T_{PD} between the two edges is calculated from (4.30). 95

32 Graph of the digital correlator in . The edges that are not labeled are assumed to have zero weight. 104

33 Graph of the digital correlator with added REC values. The edges that do not have $[m, c]$ values assigned are assumed to have values $m = 0$ and $c = 0$. 104

34 The internal path delay between registers located on the same edge, $T_{PD_{Internalmax}}$, is equal due to the definition of the RECs. This example demonstrates the case where $w(e_3) = 3$. 114

35 An example graph in which the internal path delay on edge e_2 exceeds the path delay between e_2 and e_3. This graph exemplifies the importance of considering internal long paths before the retiming process is initiated. . 114

36 Pseudo-code for RETSAM 118

37 Pseudo-code for clock period feasibility test, CHECKCP 119

38 Pseudo-code for the branch and bound algorithm SOLVELAGS that calculates the vertex lags. 123

39 An example graph in which the path delays do not monotonically increase. The sub-path p_1 has a delay greater than its original path p. The cause of this non-monotonic behavior is due to the negative clock skew between edges e_1 and e_2. Note that only the maximum path delay, $T_{PD_{max}}$, is considered and the load-dependence of the delays is assumed to be negligible for simplicity. 127

40 Pseudo-code version of algorithm for designing the clock distribution network 131

41 A path p with 3 registers and 2 vertices. 134

42 A path containing two registers and a vertex between the two registers. The path delay T_{PD} contains components related to the register delays. If all registers in the circuit are similar, the register delay components would be equal. 144

43 Effect of the pipelining depth on the clock period on C17. The initial clock period is 112 tu which is shown as a solid line. 147

44 The theoretical limit from Figure 43. Note that the effect of negative clock skew is ignored. 159

Chapter 1. Introduction

Digital circuits perform tasks by logical functions. The speed at which these logical operations can be completed is defined by the delays of the logic elements used in the digital circuits such as inverters, OR gates, and AND gates. A digital circuit containing one or multiple levels of logic elements is called a **combinational** circuit. Alternatively, in a **synchronous** circuit, synchronizing elements called **flip flops** or **registers** are used to ensure that data signals reach specified destinations at specified times. These synchronizing elements permit the realization of complex digital functions which are sequentially ordered.

The concept of inserting flip flops into a circuit in order to divide the main circuit function into isolated sub-functions was first proposed by Cotten in [1]. The sub-functions are performed independently of each other by placing the logic elements between the flip flops. The result of one sub-function is held at the input of the corresponding register and upon arrival of the next clock signal, transferred to the output of that register. Therefore, provided that a certain function can be divided into n independent sub-functions, the overall processing speed of the system can be increased by a factor of n by using $n - 1$ flip flops. This technique is called **pipelining**.

A revised version of the same technique called **maximum-rate pipelining** was first proposed in [2], also by Cotten. In this technique, multiple data signals are fed into the input of a circuit during the same clock cycle such that these signals travel simultaneously from one register to the next without overriding each other. This process of transmitting multiple *waves* of data signals permits operating at a higher data rate than is possible with conventional pipelining. This technique requires careful consideration of the data skew, specifically the difference between the *minimum* delays and the *maximum* delays between the registers, since data-overwriting could occur. This technique is called **wave pipelining** [3, 4]. In wave pipelining, the capacitance in the logic elements is used to temporarily store the values of the data waves. Typically, wave-pipelined systems can operate at three to seven times the frequency of a similar pipelined system. The limiting factor for the speed of a wave pipelined system is the *differences* between the different path delays, since the number of waves that are possible is limited by the uncertainities in the path delays [3] caused by process parameter variations [4].

Pipelining of recursive structures such as digital IIR filters has been studied in [5–9]. The inherent difficulty in pipelining recursive digital filters is due to the delay of the feedback loop built into these architectures [10]. Three different compensation techniques are introduced to handle the feedback [9–11]. The primary goal of pipelining a synchronous system is to increase the operation speed of the system. Similarly, the purpose of pipelining a recursive digital filter is to increase the sampling frequency of the filter. The application of algorithms to increase the sampling frequency of a digital filter is discussed in this dissertation.

Although pipelining offers significant performance advantages, this technique increases the latency of the synchronous circuit (the number of clock cycles required for a data signal to reach the output of the circuit). A technique to increase the speed of a synchronous circuit without changing the latency was first proposed by Leiserson and Saxe in [12]. This technique, called **retiming**, relocates the flip flops in a synchronous circuit so as to increase the speed of the synchronous circuit by changing the relative location of the registers. The number of clock periods required to traverse every path between the input and the output is preserved, *i.e.*, the overall input-to-output latency of the synchronous circuit remains the same once this operation is completed [13, 14]. This technique therefore improves the synchronous speed without changing the latency while preserving the system function. Advanced techniques to increase the accuracy of retiming by incorporating low-level circuit issues are proposed in [15–17].

The aforementioned techniques exploit *parallelism* both in *time* and in *space*. In pipelining, parallelism in time is exploited by inserting the pipelining registers so as to permit parallel execution of multiple functions at the same time. Since each register is responsible for completing an independent subtask of a larger task, the results can be accomplished in parallel. This concept is similar to using multiple functional elements to perform multiple operations at the same time, thereby exploiting parallelism in space. Although early work in both pipelining and retiming permits automating the pipelining of VLSI circuits, accurate application of these techniques is not possible without considering low-level circuit details.

Incorporating low-level circuit parameters such as clock skew, interconnect delay, and variable register delays has been the focus of recent research. Early researchers [12] have assumed equal register delays to model the effects of the registers. Equal register delays, however, have no impact on the optimization process since simple biasing of the clock period by the register delay value is sufficient to calculate the new clock period of the synchronous circuit [18]. Incorporating *variable* register delays, however, requires the redesign of existing algorithms and was first introduced in [19] along with variable interconnect delays. This approach was also extended by adding the effects of load-dependent delay [20, 21]. Incorporating clock skew is the first attempt to make retiming into a more practical optimization technique [19, 22–25]. Proposed research in this field includes optimizing multi-phase synchronous circuits [17, 26–29] as well as combining retiming with clock scheduling [30, 31]. Research in this field later has evolved to include the effects of precharged gate delays [32], multiplexers [33, 34], and relocating the registers so as to reduce the switching activity, thereby resulting in lower power consumption [35].

Automating the pipelining and retiming of VLSI circuits by considering the aforementioned low-level circuit details forms the basis of this dissertation. Without considering these low-level circuit characteristics, a useful optimization of a synchronous circuit is not feasible. The results obtained by omitting these effects may make the circuit implementation completely invalid. Therefore, this dissertation focuses on accurately automating the VLSI synchronous circuit optimization process without neglecting these crucially important circuit parameters.

This dissertation is organized as follows: In Chapter 2, background information pertaining to the later chapters is provided. Terms that are used throughout this dissertation are introduced, and some common algorithms that form the basis of the algorithms that are developed in the following chapters are described. In Chapter 3, the aforementioned synchronous optimization techniques are introduced. A review of pipelining and retiming techniques is provided in this chapter. The model developed to include low-level circuit issues in synchronous optimization algorithms is introduced and described in Chapter 4. Based on the algorithms introduced in Chapter 2 and the timing model introduced in Chapter 4, new algorithms are developed that utilize this improved timing model. These retiming algorithms based on the enhanced timing model in Chapter 4 are described in Chapter 5. Some conclusions pertaining to the algorithms and techniques introduced in this research are drawn in Chapter 6. A discusion related to open-ended future research is provided in Chapter 7.

Chapter 2. Theoretical Background

The primary objective of this dissertation is to introduce algorithms and design techniques that both consider and exploit low-level circuit information, such as the delays of the circuit building blocks including the effects of interconnect impedances. A synchronous circuit can be represented by a graph. The primary building blocks of a synchronous circuit are therefore represented by the vertices and edges of the graph. Values are assigned to the vertices and edges of the graph to characterize the components of the synchronous circuit being modeled by the graph. Using this modeling, existing graph-theoretical algorithms can be used for synchronous circuit optimization. Low-level circuit characteristics are integrated into a circuit model and previously developed algorithms to increase the accuracy of these existing algorithms.

In this chapter, a theoretical background is provided to introduce the key elements of this research. In Section 2.1, the terms related to synchronous circuits and used throughout this dissertation are introduced. The process of representing a synchronous circuit by a graph and some background information related to graph theory is described in Section 2.2. Due to its importance to this research, Section 2.3 is devoted to issues associated with the clocking of synchronous circuits. In Section 2.4, key issues related to converting a graph representation to a set of mathematical inequalities is described. Finally, the algorithms that are used to optimize the synchronous circuit are reviewed in Section 2.5.

2.1. Terms Related to Synchronous Circuits

A synchronous digital circuit consists of logic elements and storage elements (also called **registers** or memory elements) between these logic elements as shown in Figure 1. Synchronous circuits typically operate by clocking all registers in a circuit simultaneously. The logic elements located between the registers perform subtasks of the main system function. At the beginning of each clock interval, a different subtask is initiated by the logic subcircuit located at the output of each register. Each suboperation must be completed before the next clock signal is applied [1]. The temporal distance between the adjacent clock signals (called the **clock period** of the synchronous circuit) defines the rate at which new operations are performed. At each clock signal, the results of the completed subtasks are transferred to the next logic operation, and the following subtask is initiated. A synchronous circuit containing three registers is shown in Figure 1. Data signals are moved from the input of the circuit to its output at a rate defined by the clock frequency of the circuit. With this strategy, logic elements located at the output of different registers can work on different subtasks at the same time, thereby permitting a significant enhancement of the amount of information that can be processed concurrently by a synchronous circuit [1, 36].

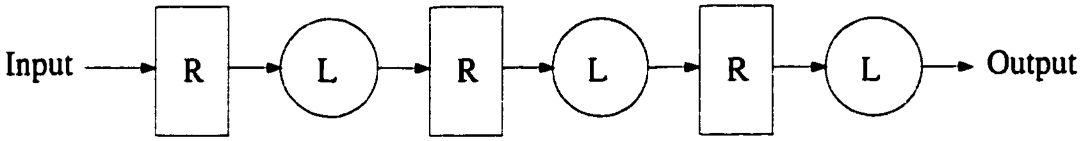


Figure 1. A synchronous circuit with three registers. Data signals are moved from the input to the output at a constant rate (the clock period). Registers are indicated as "R," and the logic elements are indicated as "L."

The **latency** of a circuit is broadly defined within the literature. For a combinatorial circuit, the latency of the circuit is defined as the *time* required for a signal to reach the system output after arriving at the system input. For synchronous systems, however, this definition may be extended. For a synchronous system, two different latencies may be defined: **temporal latency**, the *time* required for a signal to reach the system output after arriving at the system input, and **sequential latency**, the *number of clock periods* required for a data signal to reach the system output upon the data signal arriving at the system input. The difference between these two definitions can be significant, since sequential optimization operations often do not change the sequential latency, but do change the temporal latency. These definitions of latency are used herein in order to provide insight into different effects of sequential optimization on system latency [37].

The **clock frequency** of a sequential circuit is the rate at which new data flow into the system and appear at the output. The primary goal of sequential optimization is to increase the clock frequency or equivalently decrease the clock period, the reciprocal of the clock frequency. The relationship between clock period and latency depends upon the degree of pipelining as described in [38, 39].

In a synchronous digital circuit, clock signals are distributed across an integrated circuit over conducting wires, typically composed of metal. These metal wires have a distributed *RC* impedance which degrade the shape of the clock signals. Depending on the thickness and width of the wires and the distance of the connections distributing the clock signal, clock delays may vary throughout the circuit. Since the *RC* impedance cannot be made zero in practical circuits, differences among the delays from the global clock source to different points in the circuit cannot be eliminated, unless specific design techniques are utilized. Due to this attribute of clock distribution networks in sequential circuits [40], clock signals typically reach different points in the circuit at different times. Therefore, differences in delay exist between the arrival times of the clock signals at different registers.

The absolute delay of the clock signal from the global clock source to a specific register (or memory element) is the **clock delay** and is denoted as T_{CD}. The difference between the clock delay of any two registers is the **clock skew** between these registers, denoted as T_{Skew}. The notion of **localized clock skew** and its application to increasing the clock frequency within pipelined systems was first introduced by Friedman and Mulligan in [38]. They show that only clock skew between **sequentially adjacent registers** (registers that receive information at successive clock intervals and are either directly connected or connected by logic elements) is significant in pipelined systems.

Sequentially adjacent registers form a **local data path** (LDP). A local data path consists of two sequentially adjacent registers and logic elements between these registers,

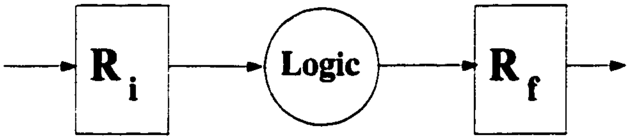
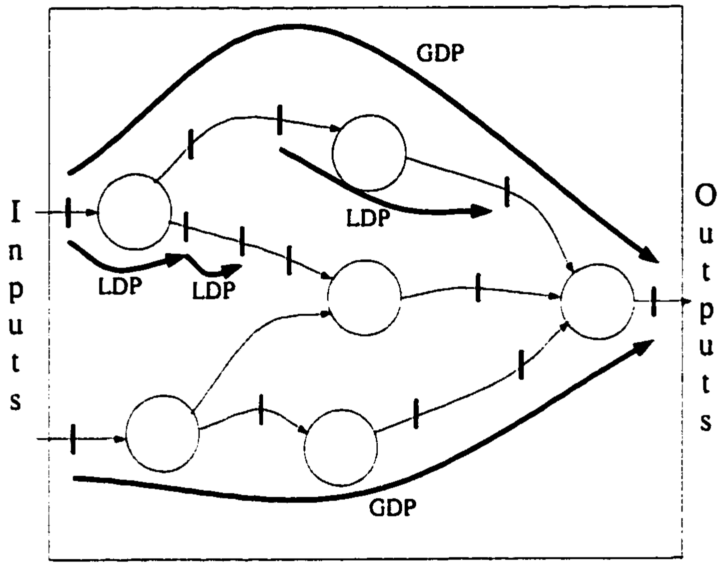


Figure 2. Two sequentially adjacent registers (initial register R_i and final register R_f) together with the logic elements between the two registers. This path forms a local data path.

as shown in Figure 2. R_i and R_f are the initial and final registers, respectively. The local data paths with the greatest delay are the **critical data paths**, whose delay defines the minimum clock period (and therefore the maximum clock frequency) of the circuit. To permit a sequential circuit to operate faster, the amount of logic placed between these registers must be decreased. However, the cost (in area) of the sequential circuit increases in this case due to the added overhead caused by the additional registers in the circuit. This trade-off between circuit area and circuit speed has been previously investigated in [41]. **Global data paths** (GDP) are those paths between any input and any output of a pipelined circuit. A synchronous circuit with two inputs and a single output is shown in Figure 3.

Note the aforementioned concept of *adjacency* between any two registers. The term that is defined here and plays a crucially important role in the following chapters is the sequential adjacency of a pair of registers. These two registers receive successive clock signals. Therefore, if a register receives a data signal at clock signal n, a sequentially adjacent register receives the processed data signal at a clock signal $n+1$. For example, as shown in Figure 4, register pairs (r_0, r_1), (r_1, r_2), (r_4, r_1), and (r_2, r_3) are sequentially



Local and Global Data Paths

Figure 3. A synchronous circuit with two inputs and a single output. Some local data paths (LDP) and global data paths (GDP) are indicated on the circuit. Registers are indicated with a vertical bar along the edges.

adjacent, whereas register pairs (r_0, r_2), (r_4, r_2), (r_0, r_3), (r_4, r_3), and (r_1, r_3) are not sequentially-adjacent.

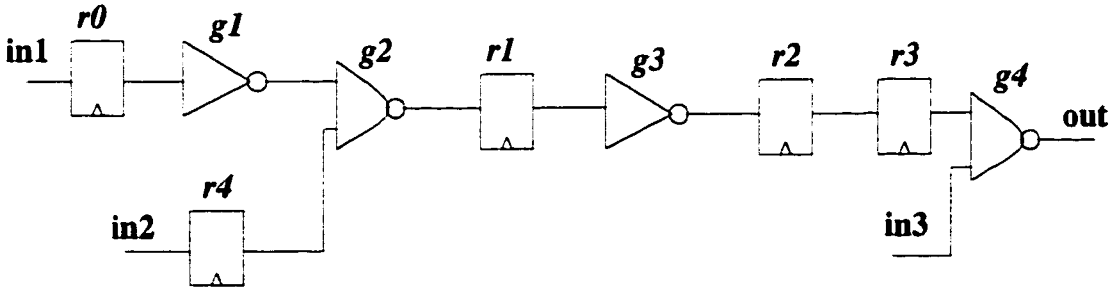


Figure 4. A synchronous circuit with four logic gates (g1 through g4) and five synchronizing registers (r0 through r4).

2.2. Representing Synchronous Circuits Using Graphs

Synchronous circuits can be represented by graphs to permit the utilization of graph-theoretical approaches for synchronous optimization. In subsection 2.2.1, keywords that are borrowed from graph theory and used in this dissertation are introduced. The process of representing synchronous circuits using graphs is described in subsection 2.2.2. The material presented in this section permits using algorithms from graph theory to solve synchronous circuit optimization problems.

2.2.1. Background Information on Graph Theory

A graph is a representation of a circuit, machine, or an event chain and is composed of an edge set E and a vertex set V. Cardinalities of these sets, $|V|$ and $|E|$, denote the number of edges and vertices in the graph, respectively. A simple graph with seven vertices and seven edges is shown in Figure 5. The numbers inside the vertices and above the edges represent the vertex or edge **weight**, respectively. Depending on the application, this weight can be used to represent time, physical weight, length, etc. If, for example, the vertices are used to represent cities and the edges are used to represent highways between the cities, the edge weights can be used to represent the travel times along the highways and the cities can be used to represent the layover times inside the city terminals. A classical application of this type of representation is determining the shortest transition times between any pair of cities.

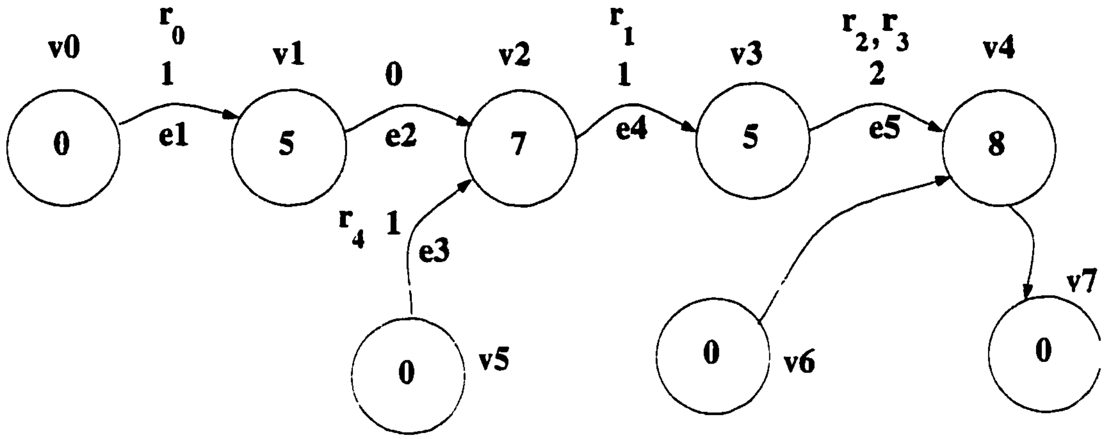


Figure 5. A simple graph with an edge set E and a vertex set V.

Every edge e has an initiating vertex and a terminating vertex. An edge that is initiated at vertex u and terminated at vertex v is represented as $e : u \rightarrow v$. To refer to these vertices without having to use different letters, the notations $e.start$ and $e.end$ are used in place of u and v.

A **path** is a route between any pair of vertices or edges. A **vertex-to-vertex path** is a path that starts at a vertex u and ends at a vertex v and includes edges e_0 through e_k. Such a path p is represented as $p : u \rightarrow e_0 \rightarrow v_0 \rightarrow e_1 \rightarrow \ \ldots \ \rightarrow e_{k-1} \rightarrow v_{k-1} \rightarrow e_k$, or simply $p : u \rightsquigarrow v$. An **edge-to-edge path** is represented in the same manner. For example, a path that starts at edge e_0 and ends at edge e_k is represented as $p : e_0 \rightsquigarrow e_k$.

2.2.2. Synchronous Circuits Represented as Graphs

Synchronous circuits can be represented as graphs by using the vertices to represent the logic elements and the edges to represent the connection between a pair of logic elements. The edge weight is used to denote the number of registers between the logic elements and the numbers assigned to the vertices are used to denote the delay of the logic elements. The weight function $w : E \rightarrow Z$ is defined to denote the weight of the edges. In the same manner, the delay function, $d : V \rightarrow Z$ denotes the delay of the logic elements. For simplicity, it is assumed that the delay function is an integer, although the algorithms developed herein can be used with non-integer delays. The **path weight**, $w(p)$, is defined as the sum of the weights of the edges along the path p. The **path delay**, $d(p)$, is similarly defined as the sum of the delays of the logic elements along a path p.

Let p be a path with weight $w(p)$ and delay $d(p)$. The clock period of a synchronous circuit T_{CP} can be calculated as the greatest delay of any zero weight path in the circuit, as follows:

$$T_{CP} = \max\{d(p) : w(p) = 0\}. \tag{2.1}$$

The path that has the greatest delay is called the **critical path** of the circuit. The rationale behind (2.1) and synchronous data flow in general is as follows: In a synchronous circuit, a data signal departing from a register must be provided sufficient time to arrive at the temporally farthest sequentially adjacent register. Otherwise the following clock signal will be applied before the data signal reaches the temporally farthest sequentially adjacent destination and is successfully latched, possibly overwriting an earlier data signal. Therefore, the temporal distance between the clock signals must be greater than

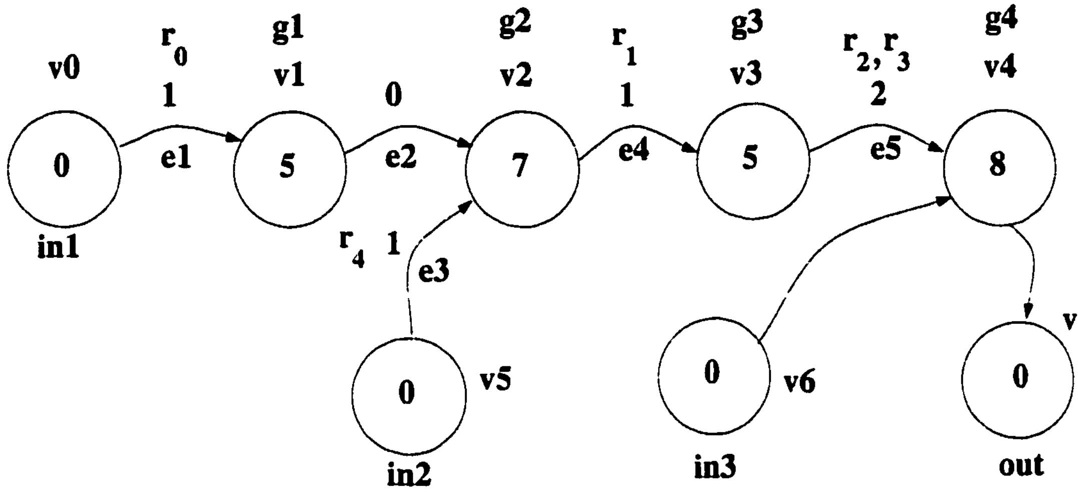


Figure 6. The graph representation of Figure 4. Vertices are used to represent the logic elements and edges are used to represent the connection between the logic elements. Note the use of zero delay vertices to model the inputs and outputs of the circuit.

the longest delay between any sequentially adjacent register pair. Therefore, *balancing* the delays in a synchronous circuit yields a very efficient circuit since the number of logic blocks that are idle waiting for the next clock signal to arrive is minimized.

The graph representation of the synchronous circuit shown in Figure 4 is depicted in Figure 6. In Figure 6, the connections between the logic elements with no registers are modeled as zero weight edges and connections between the logic elements with registers are modeled as weighted edges. Zero delay vertices are used to represent the input and output nodes of the circuit.

2.3. Clocking of the Synchronous Circuits

The notion of sequential adjacency can be extended to edges. **Sequentially adjacent**

edges are those edges that receive data signals at successive clock signals. Only the last register of the initial edge and the first register of the terminating edge are sequentially adjacent. In Figure 6, the edge pairs (e_1, e_2) and (e_3, e_4) are sequentially adjacent, whereas the edge pairs (e_1, e_5) and (e_2, e_3) are not sequentially adjacent.

The clock skew T_{Skew} between two sequentially adjacent edges i and j is defined as

$$T_{Skew}(i, j) = T_{CD}(i) - T_{CD}(j), \tag{2.2}$$

where $T_{CD}(i)$ and $T_{CD}(j)$ are the clock delays from the global clock source to nodes i and j. If $T_{CD}(j) > T_{CD}(i)$, the clock skew between registers i and j is defined as being negative. **Negative clock skew** occurs if the initial clock signal leads the final clock signal of a local data path. If $T_{CD}(j) < T_{CD}(i)$, the clock skew between registers i and j is positive. **Positive clock skew** occurs if the initial clock signal lags the final clock signal of a local data path. In the case that $T_{CD}(j)$ equals $T_{CD}(i)$, *i.e.*, the clock signal reaches the clock input of the two registers at precisely the same time, the clock skew is zero [42]. In Figure 7, it is shown how negative and positive clock skew can be created depending on the lead/lag relationship between the clock signals arriving at the initial and final registers of a local data path.

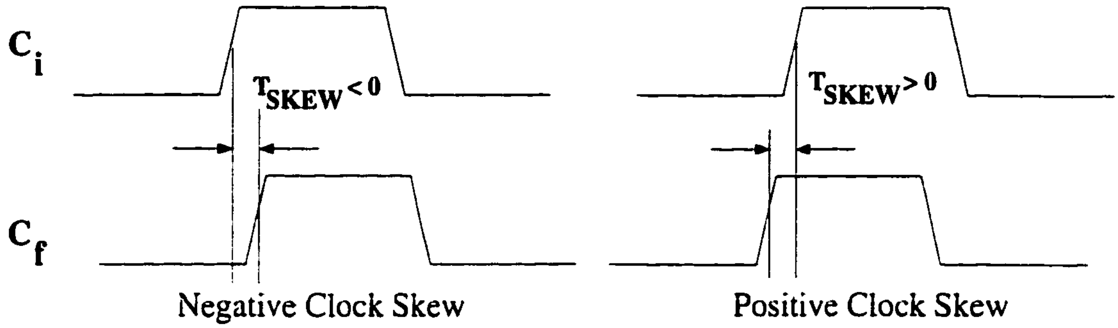


Figure 7. Due to the difference between the arrival time of the clock signal at the initial register (C_i) and at the final register (C_f), negative and positive clock skew is created between the registers depending on the lead/lag relationship between C_i and C_f.

Positive clock skew increases the path delay of a local data path, potentially making its local data path a critical path, whereas negative clock skew may improve circuit speed in critical paths [42, 43]. However, negative clock skew may also create negative path delays, resulting in **race conditions**. Race conditions are caused by *early-clocking*, *i.e.*, clocking of registers before the relevant data is successfully latched. A race condition occurs if the skew is negative and greater in magnitude than the total local data path delay [38, 42, 43]. Those paths with negative delay are called **short paths** [44]. Similarly, a **long path** designates those paths with a delay greater than the desired clock period of the circuit.

2.4. Formulation of the Synchronous Circuit Optimization Problem

For a synchronous circuit to function properly, the clock period of the synchronous circuit must be greater than the delay of the critical data path of the circuit. Synchronous optimization techniques permit the relocation or insertion of registers in the synchronous circuit so as to reduce the effective clock period. The critical path delay of the circuit may be reduced by placing a register within the critical path of the circuit. However, after inserting a register, one additional clock period is required to traverse the critical path, *i.e.*, the critical path is broken into two separate local data paths, each with smaller delay. Due to this added register, other registers may have to be removed from a related path so as not to increase the latency of the overall circuit. Removing registers from related paths may further change the latency and delay of these paths. Therefore, changing the location of one register may require a change in the entire circuit structure so as to preserve the original circuit function.

This complicated synchronous circuit optimization problem may be solved by applying linear programming methods. Local timing constraints are derived to ensure proper circuit operation. These constraints are converted to a set of linear inequalities permitting the application of standard linear programming techniques. These timing constraints are written to achieve a certain clock period by either keeping the latency constant or by increasing the latency. In the event that no satisfactory solution for the constraints is possible, the specified clock period cannot be achieved [14, 45].

2.4.1. Timing Constraints

Utilizing linear programming techniques for solving the synchronous circuit optimization problem is accomplished by formulating the synchronous optimization problem as a linear program. The linear program consists of a set of timing constraints derived from each local data path of the synchronous circuit while achieving a specific clock period. The timing constraints are represented as the weight of the edges and the delay of the paths. The first set of constraints, the **edge weight constraints** [13, 23], is used to ensure that the weight of the edges are nonnegative after the optimization process has been completed. A second set of constraints, the **long path constraints** [14] and the **short path constraints** [23, 37], is used to ensure that the delay of each path in the circuit remains within a specified range. Vertex lags are introduced and used in this section to improve computational efficiency.

2.4.1.1. Edge Weight Constraints

In the algorithms introduced in this dissertation, *negative edge weights* are not permitted. Negative edge weights are permitted temporarily for peripheral edges in [46] in order to shift the registers to the periphery of a synchronous circuit. This approach permits combinatorial optimization to be performed on the circuitry placed between the peripheral edges. However, since the algorithms described in this paper do not utilize this logic optimization feature, negative edge weights are disallowed. The negative edge weight constraint can be written as

$$w(e) \geq 0, \ \forall e \in E . \tag{2.3}$$

According to (2.3), all edge weights must have a zero or higher integer value. Edge weights are not permitted to be negative, even temporarily. This constraint ensures that the edge weights are non-negative after completion of the synchronous optimization process without requiring additional optimization steps.

2.4.1.2. Long Path Constraints

If a clock period C is desired, none of the zero weight paths in the synchronous circuit are permitted to have a delay that is more than C time units. Therefore, to ensure that a synchronous circuit has a clock period $T_{CP} \leq C$, paths with a path delay $d(p) > C$ are disallowed. Assuming that the registers in the circuit have zero delay, the method used to eliminate an undesired path is to place a register along that path in order to make the path non-zero weight. Therefore, since the resulting path will have a higher weight, the path is divided into paths with less delay. For example, in Figure 6, the clock period is 12 **time units (tu)**, since the critical path $v1 \rightarrow v2$ has a delay of 12 tu. If a clock period of $C = 10$ tu is desired for this circuit, a register can be placed on edge e_2, thereby breaking this path into two smaller zero weight paths. This operation of inserting a register, however, changes the relative temporal nature of the circuit since the two inputs of v_2 receive the data signals at different time intervals. To maintain the original circuit timing characteristics, a register may be placed on e_3, thereby delaying both inputs of v_2 by the same number of clock periods. Altough this process decreases the clock period of the circuit, it increases the sequential latency of the entire circuit since the global data paths $v0 \rightarrow v7$ and $v5 \rightarrow v7$ have a higher weight due to the added registers. The sequential optimization techniques presented in this dissertation fall into the two separate categories of pipelining and retiming. Although both techniques are used to increase the overall system-wide clock frequency, pipelining increases the latency, whereas retiming preserves the original sequential latency.

For a path p consisting of multiple edges, the long path constraint is in the form of

$$w(p) > 0, \ \ \forall p : \ d(p) > c. \tag{2.4}$$

An interpretation of (2.4) is that a critical path p can be eliminated by making it non-zero weight, thereby dividing the path into multiple paths with smaller delay. It is important to note that (2.4) can be used to eliminate any *undesired* path, *i.e.*, not necessarily a path with a delay *greater* than desired. A path with a delay *lower* than desired can also be eliminated using the same process as described in the following subsection.

2.4.1.3. Short Path Constraints

Short paths are created when the total delay of a path including the clock skew and register hold times is a zero or negative, thereby causing race conditions. Short path constraints can be modeled similarly to long path constraints. The notion of short paths plays a crucially important role in this dissertation and is investigated in great detail.

2.4.1.4. Vertex Lags

Observe that as the length of a path grows, the family of inequalities represented by (2.4) takes the form

$$w(e_0) + w(e_1) + \cdots + w(e_{k-2}) + w(e_{k-1}) > 0, \tag{2.5}$$

for a path consisting of k edges. However, Leiserson and Saxe show that by assigning an integer to each vertex, all of the inequalities that are represented by (2.4) can be transformed into a much simpler form [12–14]. The vertex lag function $r : V \rightarrow Z$ is an integer vertex label which is defined as follows:

$$w'(e) = w(e) + r(v) - r(u), \tag{2.6}$$

where $e : u \rightarrow v$ is an edge in the circuit with an initial weight of $w(e)$ and a weight of $w'(e)$ after completion of the synchronous circuit optimization. $r(u)$ and $r(v)$ are the vertex lags assigned to the vertices u and v. Using vertex lags for any path $p : v_a \rightsquigarrow v_b$ in the circuit, the following inequality holds:

$$w'(p) = w(p) + r(v_b) - r(v_a), \tag{2.7}$$

where $w(p)$ and $w'(p)$ are the initial and optimized path lengths and $r(v_b)$ and $r(v_a)$ are the vertex lags for the initial and the final vertices of path p.

Assume that path $p : v_a \rightsquigarrow v_b$ has an initial weight of $w(p) = 0$ and p has a delay greater than a desired value c, *i.e.*, $d(p) > c$. Therefore, path p can be made non-zero weight to eliminate that path. To make p non-zero weight, the following inequality is used,

$$w'(p) \geq 1. \tag{2.8}$$

or using vertex lags according to (2.7),

$$w(p) + r(v_b) - r(v_a) \geq 1 \;\Rightarrow\; r(v_a) - r(v_b) \leq w(p) - 1. \tag{2.9}$$

The right hand side of this equation $(w(p) - 1)$ consists of constants that are derived from the circuit, whereas the left hand side $(r(v_a) - r(v_b))$ consists of unknowns that are solved to achieve synchronous circuit optimization. Writing a set of inequalities in the form of (2.9) and solving for these inequalities using linear programming techniques [47] forms the basis of most synchronous circuit optimization techniques described in this dissertation.

2.4.2. Linear Programming

The most common linear programming (LP) problem consists of solving the following set of equations [48]:

$$\begin{aligned} x_1 - x_2 &\leq a_{12} \\ x_3 - x_4 &\leq a_{34} \\ &\vdots \\ x_{n-1} - x_n &\leq a_{n-1n}, \end{aligned} \tag{2.10}$$

where x_1 through x_n are unknowns and a_{12} through a_{n-1n} are constants.

The family of linear inequalities shown in (2.10) can be solved by a **Linear Program**. Standard synchronous optimization techniques consist of converting the aforementioned timing constraints into a linear program similar to (2.10), and applying a linear programming method such as the Bellman-Ford method. The Bellman-Ford method described in [47] is described in detail in the following subsection.

2.5. Algorithms for Synchronous Circuit Optimization

The building blocks for the algorithms introduced in this dissertation are topological sort, the Floyd-Warshall method, and the Bellman-Ford method. In subsection 2.5.1, the notation for describing the complexity of the algorithms used throughout the dissertation is introduced. The topological sort algorithm is introduced in subsection 2.5.2. The Floyd-Warshall and Bellman-Ford algorithm for determining shortest paths in a graph are introduced in subsection 2.5.3. These algorithms are used both in standard synchronous optimization algorithms [12–14] as well as the algorithms described in this dissertation.

In Section 2.5.4, the *branch and bound algorithms*, used to solve the generalized retiming problem, are introduced. These algorithms are used in those problems where the use of linear programming methods are not feasible. Branch-and-bound algorithms require excessive time, thereby making the application of these algorithms to large problem sets impractical. However, the properties of branch-and-bound algorithms is introduced for completeness. Readers with algorithms background can skip over this section.

2.5.1. Notation for Algorithmic Complexity

One of the more important attributes of an algorithm is its time-complexity, *i.e.*, the time requirement in relationship to its input size. The time-complexity of an algorithm is defined in terms of its asymptotic behavior. To clarify this concept, a simple algorithm that finds the maximum of n numbers can be considered as an example. In this example,

n numbers are processed, *i.e.*, the input size is n. If the input size doubles, the amount of time required for the algorithm to complete its processing is doubled. Thus, the operation time depends *linearly* on the input size. In this case the time-complexity of the algorithm is $O(n)$.

Similarly, an algorithm that has a square time-dependence has a time-complexity of $O(n^2)$. An example is the bubble sort algorithm. Such an algorithm takes four times as long to complete when the input size doubles.

Although Θ and Ω notations of time-complexity exist for asymptotically tight and asymptotically lower bounds, respectively [49], only the asymptotically upper bound notation $O()$ is used to denote time complexity in this dissertation.

2.5.2. Topological Sort

Let G be a directed acyclic graph (DAG) whose vertices v_1 through v_n represent the sub-events of a main event, and whose edges e_1 through e_k define the order of occurence of these events. If event v_i occurs before event v_j, this behavior is represented by an edge from v_i to v_j. Topological sort is the process of ordering the vertices in such a way that if there is an edge from v_i to v_j, v_i appears before v_j [49]. For a topological sort, G is assumed to be directed and acyclic. In this dissertation, only directed acyclic graphs are considered since DAGs are the only type of graphs that are encountered for the specific problems discussed in this dissertation.

```
1. Let $S[]$ be the array of sorted vertices in $G$.
2. Let $c[]$ be the array that stores the colors of vertices of $G$, $V(G)$.
3. for $\forall\ u \in V(G)$ do {
4.     $c[u]$ = WHITE
5.     $S[u] = nil$
6. }
7. $i = |V|$
8. for $\forall\ u \in V(G)$ do
9.     if $color[u]$ = WHITE then VISIT (u)
```

Figure 8. Pseudocode of the Algorithm *TS* for the topological sort of a graph G.

Topological sort utilizes a technique called a depth first search (DFS). In this subsection, the DFS algorithm is not separated from the topological sort. The pseudocode of the topological sort algorithm *TS* is shown in Figure 8. Algorithm *TS* plays an important role in the algorithms discussed later in this dissertation since the candidate paths are sorted using this algorithm, *TS*.

Algorithm *TS* uses two arrays $S[]$ (Step 1), and c[] (Step 2) to store the sorted vertex indices, and the colors of the vertices, respectively. The status of the vertices are indicated using colors: WHITE indicates an unprocessed vertex, GRAY indicates a process that is being processed, and BLACK indicates a vertex whose processing is completed. The initialization phase of the algorithm (the **for** loop between steps 3 and 6) colors all vertices WHITE (Step 4), and clears the sorted index array S (Step

5). The variable i is an index pointer to array $S[]$ which initially points to the end of the array (Step 7). Every vertex is scanned using a simple **for** loop (Step 8) by using the recursive function *VISIT* (Step 9). The **for** loop in Step 8 starts by passing the first vertex in the vertex list $V(G)$ to the function *VISIT()*. *VISIT()* scans every vertex attached to this first vertex recursively. If there are no disjoint vertices, at the first call to *VISIT()*, all vertices will be BLACK and the function will return with the entire $S[]$ array filled with the sorted vertex indices. The **for** loop in Step 8 processes more than one vertex if and only if there are disjoint vertices. This step is due to the fact that the disjoint vertices will remain WHITE after the first call to *VISIT*, since they have no neighboring vertices close to the first vertex. The pseudocode of Algorithm *VISIT()* to scan and color the adjacent vertices of a given vertex u is shown in Figure 9.

```
1. $c[u] = $ GRAY
2. for $\forall e \in E(G) \; : \; e : u \rightarrow v$ do
3.    if $c[v] = $ WHITE then VISIT(v)
4. $c[u] = $ BLACK
5. $s[i] = u$
6. $i = i - 1$
```

Figure 9. Pseudocode of the algorithm *VISIT* for coloring every vertex in G recursively.

The algorithm initially assigns the color GRAY to the vertex being processed (Step 1) in order to prevent processing the vertex indefinitely. The vertices that are WHITE (not visited already) are recursively scanned (steps 2 and 3). When a vertex and its neighbors are entirely visited, the vertex is colored BLACK (Step 4). After a vertex is colored BLACK, the vertex is placed at the end of the sorted list (Step 5). Using this methodology, the vertex that is terminated last is placed at the end of the list. The index variable i is decreased in Step 6 so as to store the next vertex before the vertex is terminated. Since the vertex that is visited first is stored at the beginning of the array $S[]$, $S[]$ contains the vertex indices sorted by depth at the end of algorithm *VISIT()*. Therefore, the array $S[]$ stores a topological sort of vertices of G at the end of algorithm *TS*.

2.5.3. Shortest Paths

An important family of algorithms that determine the shortest paths in a given graph G are studied in this section. An intuitive example for the application of shortest paths in synchronous circuit optimization is determining the shortest time along any logic path between a given pair of registers in a synchronous circuit. The two important variants of the shortest path problem, single source shortest paths and all pairs shortest paths, are reviewed below.

On the other hand, all pairs shortest path algorithms determine the shortest paths between any pair of vertices. The Floyd-Warshall algorithm for the determination of all pairs shortest paths and the Bellman-Ford method for the determination of single source shortest paths are discussed in the following subsections.

2.5.3.1. Single Source Shortest Paths

Single source shortest path algorithms determine the shortest paths among a source vertex s and all other possible destination vertices. Such information is applied in synchronous circuit optimization to determine the shortest path among a given vertex and all of the other vertices. An example of applying the single source shortest paths is determining the shortest transit time between a given point in the synchronous circuit and any destination point. Among other existing algorithms, the Bellman-Ford method for determining the single source shortest paths is applied in the research presented in this dissertation and is explained later in this section.

2.5.3.2. All Pairs Shortest Paths

The algorithms applied in the research described in this dissertation utilize the all-pairs shortest paths in a given graph G. The result of the all pairs shortest path algorithms is a matrix of shortest paths. Since there are $|V|$ vertices in G, there are $|V|^2$ vertex pairs, and the all pairs shortest path algorithms determine the $|V|^2$ shortest paths between any vertex pairs. If there is no path between any given vertex pair (u, v), the resulting shortest path is ∞. $d(u, v) = \infty$ designates a non-existent path $p : u \rightsquigarrow v$, *i.e.*, a path having infinite delay between its initiating and terminating vertices. Although many algorithms exist for determining all pairs shortest paths [47, 49], only the Floyd-Warshall method is used in the algorithms described herein, and this method is explained in the following subsection.

All pairs shortest path algorithms are used to obtain global information about a graph. By observing the matrix of the shortest paths, a global decision can be made about a graph. Using the city and highway analogy, the matrix of $|V|^2$ numbers can be thought of as being a *road map*, where each number represents the minimum possible transit time between a pair of cities.

2.5.3.3. Floyd-Warshall Method

The Floyd-Warshall algorithm for determining the all-pairs shortest paths is explained in this subsection. This algorithm determines the vertex-to-vertex shortest path between every vertex pair in a graph consisting of $|V|$ vertices. To explain the internal operation of the algorithm, a matrix D that contains the shortest vertex-to-vertex path is first created. The D matrix can be thought of as being a *roadmap* which contains the shortest distances between any two cities. In terms of synchronous circuit optimization, each individual element of matrix D, $d(i,j)$, is assumed to contain the shortest path delay between vertices v_i and v_j.

The Floyd-Warshall algorithm recursively determines the elements of the D matrix by using the following definition [49]:

Let $p : v_i \rightarrow v_{k1} \rightarrow v_{k2} \rightarrow \cdots \rightarrow v_{kn} \rightarrow v_j$ be this shortest path from vertex v_i to vertex v_j going through intermediate vertices v_{k1}, v_{k2}, . . . , v_{kn}. Let $d^{(k)}(i,j)$ be the delay of a shortest path with all intermediate vertices in the set $\{1, 2, \ldots k\}$. The objective of the Floyd-Warshall algorithm is to begin with an infinitely large set and to decrease the cardinality of this set at every iteration step, thereby gradually approaching the shortest path. Simply, at iteration step 0 (*i.e.*, $k = 0$), a path from vertex v_i to vertex v_j with no intermediate vertex numbered higher than 0 contains no intermediate vertices. Therefore, $d^{(0)}$ for any vertex pair is equal to $w(i,j)$, the weight of the edge between vertices v_i and v_j. To form the recursive iteration, consider the following:

- If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set {1,2, . . . k-1}. Thus, a shortest path from v_i to v_j with all intermediate vertices in the set {1,2, . . . k-1} is also a shortest path from v_i to v_j with all intermediate vertices in the set {1,2, . . . k}.
- If k is an intermediate vertex of path p, then p can be broken into subpaths p_1 and p_2, where $p_1 : i \rightsquigarrow k$, and $p_2 : k \rightsquigarrow j$. Since k is a terminating vertex for p_1 and the initiating vertex for p_2, the intermediate vertices of both p_1 and p_2 are in the set {1,2, . . . k-1}.

Based on this strategy, the following recursive definition can be developed:

$$d^{(k)}(i,j) = \begin{cases} w(i,j) & k = 0, \\ \min\left(d^{(k-1)}(i,j),\ d^{(k-1)}(i,k) + d^{(k-1)}(k,j)\right) & k \geq 1. \end{cases} \tag{2.11}$$

Note that the two operands of the *min* operation are the $d^{(k-1)}(i,j)$, the shortest path from the previous operation if k is not an intermediate vertex at iteration k for $d^{(k-1)}(i,k) + d^{(k-1)}(k,j)$, if k is an intermediate vertex at iteration k, then the new shortest path is the sum of the shortest path weights of the paths p_1 and p_2. Therefore, (2.11) suggests that at the first iteration ($k = 0$), $d(i, j)$ must be initialized to the weight of the edge $e : i \rightarrow j$ (no intermediate vertex). A test must be made to determine whether, at iteration k, adding an intermediate vertex k along the path $p : i \rightarrow j$ decreases or increases the weight of the shortest path p. To perform this test, the current shortest path weight of path p ($d^{k-1}(i, j)$) is compared to the sum of the shortest path weight of paths p_1 and p_2 ($d^{k-1}(i, k)$ and $d^{k-1}(k, j)$, respectively). The minimum of the two values is used as the next shortest path value ($d^k(i, j)$).

Pseudocode of the Floyd-Warshall algorithm based on the recursive relationship described by (2.11) is shown in Figure 10.

```
for i = 1 to $|V|$ do
  for j = 1 to $|V|$ do
    $d^{(0)}(i, j) = w(e : i \rightarrow j)$
for k = 1 to $|V|$ do
  for i = 1 to $|V|$ do
    for j = 1 to $|V|$ do
      $d^{(k)}(i, j) = \min\left(d^{(k-1)}(i, j),\ d^{(k-1)}(i, k) + d^{(k-1)}(k, j)\right)$
Matrix $D$ contains the shortest paths
```

Figure 10. Pseudocode of the Floyd-Warshall algorithm.

After the elements of the D matrix are initialized with the edge weights $w(i,j)$ (steps 1 through 3), the initial iteration values $d^{(0)}(i,j)$ are determined. A total of $|V|$ iterations are performed (Step 4) on those $d(i,j)$ values. During the iteration (steps 5 and 6), the recursive formula of (2.11) is used (Step 7) to determine the next value of $d(i,j)$. After $|V|$ iterations are performed, the D matrix contains the shortest path values between vertex pairs (Step 8). The Floyd-Warshall algorithm is used in the research described in this dissertation within retiming algorithms to determine all of the possible delay values between any vertex pair.

2.5.3.4. The Bellman-Ford Algorithm

The Bellman-Ford algorithm is used to solve the single-source shortest paths problem by employing the technique of successive approximations [47, 49]. The source vertex is initialized with the value zero, since its distance to itself is always zero. Every other vertex is initialized with the value infinity. These tentative values are decreased at every iteration using a method called **relaxation**. At every iteration step, a tighter solution, an intermediate solution closer to the final result may be obtained. As proven in [49], after $|V|$ steps, either a solution is found, or no solution exists. A solution does not exist in the case where there is a negative weight cycle. An example which contains a negative cycle is presented later in this section.

In the last subsection, a method is explained for using the Bellman-Ford algorithm to solve a linear program. Since the synchronous optimization algorithms presented in this dissertation consist of converting timing constraints to a linear program, the Bellman-Ford method can be used in synchronous optimization algorithms by applying this technique.

Relaxation The Bellman-Ford algorithm uses the technique of relaxation. An attribute $d[v]$ is assigned to each vertex. $d[v]$ is the tentative shortest path value from the source to vertex v. At every iteration of the Bellman-Ford algorithm, this value may possibly be decreased, thereby yielding a near optimal solution which is closer to the optimal solution. At the end of $|V|$ iteration steps, either a solution is found, or it is determined that a solution does not exist. $d[v]$ contains the shortest path value after $|V|$ steps if a solution exists.

To exemplify the relaxation procedure, consider the edge $e : u \rightarrow v$ in Figure 11. Two cases are of interest, the case in which the relaxation yields a smaller $d[v]$ value, thereby obtaining a value closer to the final result (see Figure 11a), and the case that relaxation has no effect (see Figure 11b). In Figure 11a, the tentative shortest path values are $d[u] = 2$, and $d[v] = 8$ before relaxation. However, since the weight of the edge $e : u \rightarrow v$ is three, there exists a shorter path from vertex u to vertex v along edge e with a total path weight equal to $2 + 3 = 5$. Therefore, applying relaxation to edge e sets $d[v]$ to 5, thereby yielding an improved shorter path value for vertex v. In Figure 11b, on the other hand, the $d[v]$ value is 4 before relaxation, therefore, relaxation has no effect, since $2 + 3 = 5$ is not an improved alternative to the already existing tentative value of $d[v] = 4$.

The relaxation procedure is denoted as $RELAX(e)$. $RELAX(e)$ performs relaxation on an edge $e : u \rightarrow v$, which has a weight of $w(e)$. The $d[v]$ value is updated according to the aforementioned procedure. Pseudocode of $RELAX(e)$ is given in Figure 12.

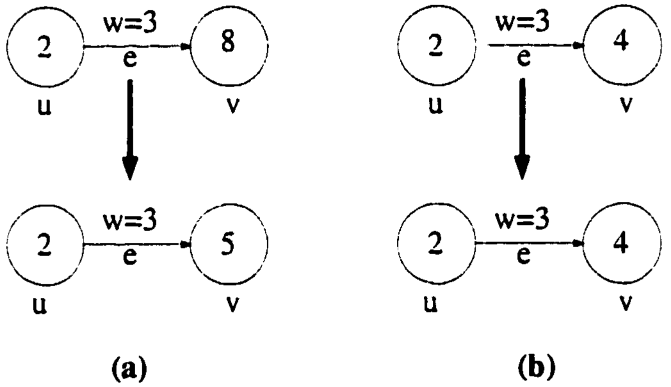


Figure 11. The relaxation procedure. (a) a tighter bound is found after relaxation, (b) relaxation has no effect.

```
if $d[v] > d[u] + w(e)$ then
    $d[v] = d[u] + w(e)$
```

Figure 12. Pseudocode of $RELAX(e)$. $d[v]$ is updated according to the aforementioned procedure.

Using Bellman-Ford to Solve the Single-Source Shortest Paths Problem

The Bellman Ford algorithm uses relaxation as its primary element to solve the single source shortest paths problem. Procedure $RELAX()$ is successively applied to every edge to obtain a tighter bound. After $|V|$ steps, either a solution is found, or it is determined that no solution exists in case a negative cycle in the graph is reachable from the source vertex. To exemplify the operation of the Bellman-Ford algorithm and the effect of negative cycles on the operation of the algorithm, consider the two graphs shown in Figures 13 and 14. A graph containing only positive cycles is depicted in Figure 13, whereas the graph of Figure 14 contains two negative cycles: the dark-colored path $p_n : v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_0$ and the path $p : v_2 \rightarrow v_3 \rightarrow v_0 \rightarrow v_2$. From the shortest paths viewpoint, having such a negative cycle implies that the distance from any vertex v_x to vertex v_0 can be arbitrarily decreased by traversing the aforementioned negative cycle path multiple times since for any path $p_x : v_? \rightarrow v_0$,

$$d(p_x) < d(p_x) + d(p_n) \quad \forall \; p_x : v_? \rightarrow v_0, \;\; p_n : v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_0. \tag{2.12}$$

(2.12) suggests that the shortest path between an arbitrary vertex and vertex v_0 ($d(p_x)$) is undefined since $d(p_x)$ can be made arbitrarily close to $-\infty$ by traversing $v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_0$ multiple times. For example, since the shortest path from the source (vertex v_s) to v_0 can be made arbitrarily small, the shortest path from the source to v_2 can also be made arbitrarily small, since there is an edge between v_0 and v_2 with a weight of three. In the specific example of Figure 14, the Bellman-Ford algorithm is not able to determine the shortest path for any of the edges due to the existence of the aforementioned negative cycle.

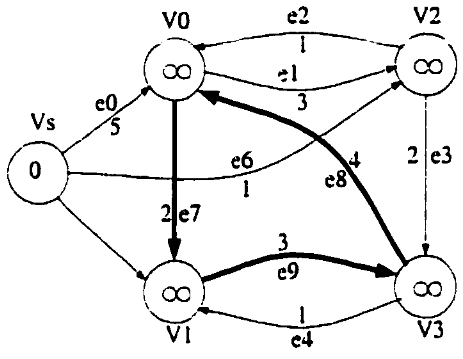


Figure 13. An example graph which does not contain a negative cycle.

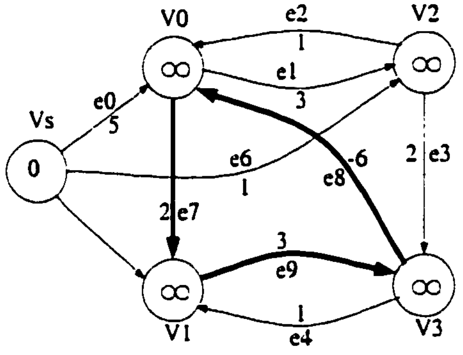


Figure 14. Graph of Figure 13 with a negative cycle along the path $v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_0$. Note that the negative edge weight of e_8 causes the negative cycle.

```
1. $d[s] = 0$, where $s$ is the source vertex
2. for $\forall v \in V - \{s\}$ do
3.   $d[v] = \infty$
4. for $i = 1$ to $|V| - 1$ do
5.   for $\forall e : u \rightarrow v \in E$ do
6.     $RELAX(e)$
7. for $\forall e : u \rightarrow v \in E$ do
8.   if $d[v] > d[u] + w(e)$ then
9.     return FALSE
10. return TRUE
```

Figure 15. Pseudocode of the Bellman-Ford algorithm.

The pseudocode of the Bellman-Ford algorithm for determining the shortest paths from a source vertex v_s to the other vertices in a graph is shown in Figure 15. The $d[]$ value for the source vertex v_s is initialized to zero (Step 1) and the $d[]$ values for the other vertices are initialized to ∞ (Steps 2 and 3). On a graph consisting of $|V|$ edges, every edge is relaxed $|V| - 1$ times (the loop of Step 4). Note that $|V|$ relaxations are unnecessary since the source vertex always has a fixed value of zero. Every edge in the graph (Step 5) is relaxed as summarized by the pseudocode shown in Figure 12 (Step 6).

After $|V| - 1$ passes are completed over all edges, Steps 7 through 10 are performed to check the validity of the results. As mentioned earlier, if there exists negative cycles in the circuit, the resulting solution is invalid, *i.e.*, a solution does not exist. This event occurs if a better solution can be obtained after $|V| - 1$ passes over the edges. In the case where there are no negative weight cycles, a solution is reached after $|V| - 1$ passes and no better solution exists after $|V| - 1$ steps. However, in the case of a negative weight cycle, a better solution is found even after the algorithm is completed. This result indicates a non-convergent set of $d[]$ values. Based on this condition, a post-iteration validity check can be performed on the $d[]$ values. This validity check includes going through every edge after the algorithm has been completed (Step 7) and checking to see if a better $d[]$ value can be obtained using relaxation (Step 8). If a smaller $d[]$ value can be obtained for any vertex using relaxation on an edge e, it indicates a negative cycle which consists of edge e, *i.e.*, the solution is invalid (Step 9). If no such edge exists, the solution is valid and the $d[]$ values contain the shortest paths to the source vertex (Step 10).

Using Bellman-Ford to Solve Linear Programs In the earlier section, the Bellman-Ford algorithm for the solution of the single source shortest paths problem is described. In this section, it is shown that with a simple modification, the same algorithm can be used to solve a Linear Program in the form of (2.10). This simple modification involves substituting the aforementioned $d[]$ values for the unknowns x_n in the linear program, *i.e.*, $d[1]$ is substituted for x_1, $d[2]$ is substituted for x_2, etc. Under this assumption, x_1 is initially zero, and the other variables are calculated relative to x_1. A careful observation of (2.10) shows that this substitution is acceptable since the Linear Program consists of a set of N inequalities, thereby permitting at least one variable to be chosen arbitrarily, which is x_1 ($d[1]$) in this case. The relaxation of edge $e : u \rightarrow v$ is equivalent to solving for the inequality $x_v - x_u \leq w(e)$. For example, the relaxation shown in Figure 11a is equivalent to solving the inequality $x_v - x_u \leq 3$, where initially $x_u = 2$ and $x_v = 8$. Note that only one variable (x_v) is changed after relaxation.

With this modification, the Bellman-Ford algorithm can be used to solve a Linear Program. The algorithms developed for synchronous circuit optimization in this dissertation create timing constraints which are written in the form of a Linear Program. This Linear Program is solved using the Bellman-Ford algorithm. Bellman-Ford runs in $O(|V||E|)$ time, due to the nested loops in Steps 4 ($O(|V|)$) and 5 ($O(|E|)$). Note that the relaxation is $O(1)$, and the post-verification between the steps 7 and 9 require $O(|E|)$ time. The proof showing how the algorithm is guaranteed to converge after $|V| - 1$ passes over all edges can be found in both [49] and [47].

2.5.4. Branch and Bound Algorithms

The Branch and Bound (B&B) algorithms are used to solve problems that cannot be solved in polynomial time effectively [50] though these algorithms are asymptotically exponential. Assume a problem P with input variables x_0 through x_{n-1}, linear constraints c_0 through c_{n-1} on input variables, and an optimum solution S. For simplicity, assume that S is an integer, although it is possible for the outcome to be a real number. Assume that L_0 is the initial lower bound for S. S can be set to $+\infty$ or more sophisticated methods can be used to determine an improved lower bound. The objective of the B&B method is to obtain a lower bound as close to S as possible, *i.e.*, to determine a set of input variables x_0 through x_{n-1} that satisies all of the constraints c_0 through c_{n-1} and yields the lowest lower bound. In this dissertation, the minimum *clock period* of a synchronous digital system is determined using these B&B methods where S is the optimum clock period for a synchronous circuit.

Here is how the B&B method works: Let a variable x_i have three possible values, –1, 0, and 1 (for example, the weight of a specific edge in a graph). For these three values, the original problem set can be partitioned into three subproblems P_1, P_2, and P_3. These subproblems now have one less input variable, $x_0, \ldots x_{i-1}, x_{i+1} \ldots x_{n-1}$, and less constraints since the constraints containing the variable x_i are now much simpler. For the subproblems P_1, P_2, and P_3, the following can be true: **1)** None of the solutions obtained in a subproblem P_n yield a better lower bound than the current lower bound L, in this case subproblem P_n is **pruned** (or disregarded). **2)** No possible solution can be determined to satisfy all of the constraints. One or more of the constraints are unsatisfiable. In this case, the algorithm **fathomes** the subproblem P_n. Assume that P_2 is pruned since the lower bound L_2 obtained from P_2 is higher than the current lower bound, *i.e.*, $L \leq L_2$. Also, assume P_1 is fathomed since one or more of the constraints is not satisfied, *i.e.*, P_1 is *infeasible*. In this case, P_3 is the only possible candidate that yields an improved lower bound than the current lower bound L. If P_3

produces a lower bound L_3 less than L, then L_3 is accepted as the solution. It may be necessary to subdivide the subproblem P_3 into subsubproblems P_{3i} to determine the optimal solution for the overall problem P by considering possible values for another variable, *e.g.*, x_{i-1}. The algorithm can be terminated if a sufficiently small solution is obtained or a certain amount of time has elapsed.

The process of subdividing the problem space into subproblems is called **branching**. At each step of branching, the subproblems are analysed and the bounds are re-adjusted, thereby forming the aforementioned **bounding** step. B&B algorithms provide an efficient approach for solving many of the complex combinatorial optimization problems introduced later in this dissertation. Additional information about B&B algorithms can be found in [50].

Chapter 3. Synchronous VLSI Circuit Optimization Techniques

The performance of synchronous circuits can be increased by pipelining at the expense of increased system latency and area. **Pipelining** converts a combinatorial circuit into its sequential equivalent by breaking the global data paths into local data paths with smaller delay. This change is achieved by inserting registers (memory elements) between the logic blocks. The intermediate processed results are saved in a memory element (or register) and used during the following clock cycle [1]. Thus, this technique increases the rate of data flow by providing concurrent operation, albeit with increased circuit area and system latency.

Retiming is a technique used to increase the clock frequency in pipelined synchronous circuits without affecting synchronous latency. An initial synchronous system is converted via retiming into a functionally equivalent system using techniques originally described by Leiserson and Saxe [14]. The locations of the registers (with respect to the logic elements) are changed so as to minimize the clock period while preserving the system function and latency. The primary distinction between pipelining and retiming is that pipelining converts a combinational circuit into a sequential circuit, increasing system latency. In retiming, alternatively, the register locations within a sequential circuit are optimized such that the circuit operates at the highest possible frequency without increasing the latency.

This chapter is intended to provide background information about the aforementioned two synchronous optimization techniques: pipelining, and retiming. A detailed review of both pipelining and retiming is provided herein. A discussion of pipelining techniques and related work in the field is provided in Section 3.1. Retiming of synchronous circuits and related retiming algorithms are discussed in Section 3.2. With this background information, the incorporation of low-level circuit parameters into the retiming process will be discussed in the following chapter.

3.1. Pipelining of Synchronous Digital Systems

Pipelining is a technique for increasing the performance of a synchronous circuit. After pipelining a combinatorial circuit, the clock frequency of the circuit is increased, resulting in higher synchronous performance. Pipelining has been used to improve the speed of a number of different applications, ranging from combinatorial circuits to microprocessors and DSP-based systems. This section is divided into four subsections: early work in the field of pipelining combinatorial circuits is reviewed in the first subsection. Pipelining of microprocessors and DSPs is discussed in the following two subsections, respectively, followed by a brief review of wave-pipelining in the last subsection.

3.1.1. Pipelining of Combinatorial Circuits

One of the earliest studies of pipelining was by Cotten in 1965 [1] in which he describes the time required for a data signal to reach the system output once it is applied to the system input as the **pipeline fill-up time**, and the rate at which the data flow in the pipeline as the **byte-flow**. A pipelined circuit is depicted in Figure 16 in which the registers are placed between logic elements so as to increase the data flow rate.

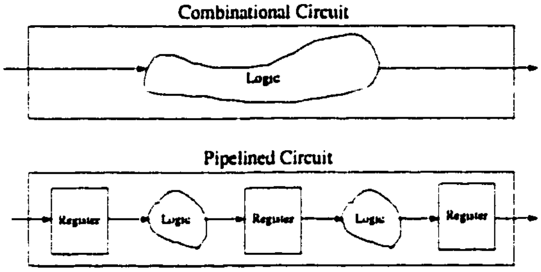


Figure 16. Pipelining breaks global data paths into local data paths with smaller delay so as to increase the data flow rate.

The dependence of the maximum flow-rate on the register delays was further investigated by Cotten in [2] and others [38, 39]. Their work showed that due to the inherent delay of the pipeline registers, the computational speed cannot be increased arbitrarily, but rather is bounded by the register delays. Jump and Ahuja [51] assign costs to registers and the logic elements and study the average delay, the average cost/operation, and the average time/operation ratios in a quantitative framework. In this dissertation, the *delay* of the circuit δ is defined as

$$\delta = N(T_S + T_R), \tag{3.1}$$

where N is the number of pipeline stages in the circuit, and T_S and T_R are the maximum logic and register delays, respectively. This definition corresponds to that of the temporal latency introduced earlier. Note that Jump and Ahuja assume uniform delays for each stage and only the maximum delay is considered. In this same paper, the pipeline efficiency is analyzed using measures such as the average cost per operation $\eta(M)$ and the average time per operation $\tau(M)$. These ratios are defined as

$$\eta(M) = (K_P + K_R N_R)(T_S + T_R)\left(\frac{M + N - 1}{M}\right), \tag{3.2}$$

$$\tau(M) = (T_S + T_R)\left(\frac{M + N - 1}{M}\right), \tag{3.3}$$

respectively, where K_P and K_R are the total cost of the logic elements and a single flip-flop per second, respectively, M is the average number of operations to be performed, and N_R is the total number of stages. In this context the definition of the *cost* is left open-ended, *i.e.*, parameters such as power consumption or area can be used as a measure of the cost depending upon the application. They further show that as the number of operations increase, the term $(M + N - 1)/M$ approaches unity. This term is defined earlier as the **efficiency** of a pipeline by Chen [52] and can be interpreted as N clock periods are required to perform the initial operation and the remaining $M - 1$ operations occur at each following clock period. As the number of operations increase, the performance degradation due to the pipeline fill-up time becomes less significant.

Another specific application of pipelining to combinatorial circuits is arithmetic functions, investigated by Hallin and Flynn [53]. They define the efficiency of pipelining as

$$pipeline\ efficiency = \frac{N}{DG}, \tag{3.4}$$

where N is the number of bits in the operands, D is the delay of each pipeline stage (assumed uniform), and G is the total number of gates in the total system including the latches. A wide range of adders (*e.g.*, carry look-ahead, conditional-sum) and multipliers (*e.g.*, Wallace, fully iterative array) are contrasted. They show that as the pipeline depth is increased by a factor k, the efficiency does not increase by the same factor due to the added overhead of the registers.

Global data paths are broken up into local data paths so as to achieve a specified clock period. Papaefthymiou presents an algorithm in [54] for automating pipelining of a fully combinational circuit in $O(E)$ time.

All of the previously discussed papers assume the pipelines consist of edge-triggered registers only. In [44], Sakallah *et al.* describe synchronizing pipelines consisting of multi-phase latches. A previously introduced timing model [24] is applied and both short and long path constraints are introduced.

3.1.2. Pipelining of Microprocessors

The application of pipelining to processor design by parallelizing the fetch, decode, and execute units was initially studied by Flynn in 1966 [36]. He showed that by parallelizing the events in a SISD (single instruction, single data) machine, it is possible to increase the rate at which the input of the system accepts new data and the rate at which the system outputs processed data. The pipelining concept, applied to the DLX processor [55], is depicted in Figure 17 in which instruction fetch (F), instruction decode

(D), execute (X), memory access (M) and writeback (W) operations are performed in parallel. Although these five operations are necessary to complete an instruction, each instruction effectively requires a single cycle due to the inherent parallelization. Note the pipeline stall (denoted as "s") which occurs at the fetch phase of the sixth instruction. Thus, the pipeline is not fully utilized in the sixth clock cycle. The shaded column denotes the clock cycle in which the pipeline is utilized 100%.

Instruction	Clock Cycle						
	1	2	3	4	5	6	7
1	F	D	X	M	W		
2		F	D	X	M	W	
3			F	D	X	M	W
4				F	D	X	M
5					F	D	X
6						s	F

Figure 17. Pipelining of microprocessors: five primary operations of the microprocessor are pipelined to increase computational speed.

Although pipelining can increase synchronous operation dramatically, it cannot be fully exploited in microprocessor architectures due to instruction dependencies, structural limitations, and branch instructions. Complete parallelization of the code is not possible since some instructions need operands produced by previous instructions [56, 57]. Branch delay and branch prediction methods have been employed to overcome this problem [55, 58, 59]. Those deficiencies that decrease pipeline efficiency are called

hazards and cause pipeline **stalls** (the situation where the execution of an instruction must be delayed due to a hazard). An example pipeline stall is denoted as "s" in Figure 17. An example of this case is the sixth clock cycle on Figure 17.

Pipelining is widely used in supercomputers. The relationship between the degree of central processor pipelining and supercomputer performance is discussed by Kunkel and Smith [60]. They show that overall pipeline performance peaks at six gates per pipeline segment. Using excessive gates per segment degrades the performance since the clock period is increased. Pipeline segments that use too few gates degrade the performance due to data and clock skews within the system. Note that **data skew** is defined as the difference in delay between the maximum and minimum signal propagation times through the combinational logic within the pipeline stages.

3.1.3. Pipelining of DSPs

The application of pipelining to enhance the performance of digital signal processors (DSPs) has been well studied [e.g., 61–63]. Capello and Steiglitz define **completely-pipelined architectures** in [62] in which circuits are pipelined down to the bit level. They apply pipelining to DSP architectures and show that complete pipelining is appropriate for array-connected (mesh-connected) DSP architectures. Capello, LaPaugh, and Steiglitz [64] define an AP-product to measure the efficiency of pipelining, where A is the area of the VLSI chip and P is the clock period. This definition of efficiency is similar to Hallin and Flynn's [53] definition in that the term G has similarities to A

(since the number of gates used in the circuit is directly proportional to the chip area) and D is similar to P. Siomalas and Bowen investigate [63] strategies for designing DSPs such that all building blocks within the DSP are active at the same time. This methodology increases the effective speed of the DSP operations since the most efficient use of pipelining is achieved by maximally exploiting the inherent temporal parallelism. They also demonstrate their method of pipelining on FFT design.

3.1.4. Wave-Pipelining

The clock frequency of a pipelined synchronous system can be increased without increasing the number of registers. This improvement in clock frequency is possible by applying input signals faster than the total delay of the data path.

Successive *waves* of data are sent through combinatorial logic paths. If the data skew is small and sufficiently accurate control of the arrival times at every node is maintained, the successive data waves can act as a pipeline, permitting fewer synchronizing registers to be used. This technique is called **wave-pipelining** and was originally proposed by Cotten in 1969 [2]. A wave-pipelined circuit in which successive data waves propagate through the pipeline is depicted in Figure 18.

Although wave-pipelining can be successfully applied to highly-structured architectures, it fails to work well in those architectures in which the path delays are unbalanced (significant data skew exists). The insertion of active elements in the system to permit wave-pipelining of unstructured architectures has been investigated by Wong, De

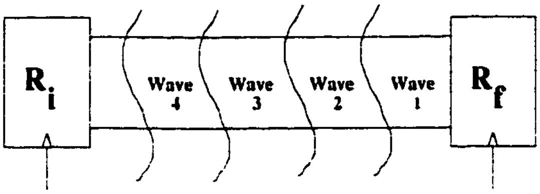


Figure 18. In wave-pipelining, successive data waves propagate through the logic elements, forming an effective pipeline.

Micheli, and Flynn [3]. The primary issue in achieving wave-pipelining is to equalize the path delays within the circuit by **padding** delays. Padding is the process in which those paths that have shorter delays are detected, permitting the insertion of active delay elements along these paths to ensure that all circuit paths have a delay between a lower and an upper bound. Recently, Shenoy *et al.* proposed greedy heuristic algorithms for padding these unequal delay paths [65]. They consider the padding operation as a post-processing step and offer linear programs for solving this problem. Wong, De Micheli, and Flynn also propose algorithms to implement wave-pipelining by padding delays [4]. They demonstrate padding on a 63–bit population counter and show that the clock frequency of the circuit can be increased by two to three times using wave-pipelining. They also show that bipolar technologies, such as CML and super-buffered ECL, are more suitable for wave-pipelining than CMOS since they have inherent uniform delay (*i.e.*, are less sensitive to input waveform and output loading).

3.2. Retiming Techniques for Sequential Circuit Optimization

Retiming is a sequential optimization technique used to increase the operating frequency of a synchronous circuit without increasing the sequential latency of the circuit. The location of the pipeline registers are reorganized so as to achieve the minimum clock period of a synchronous circuit while preserving the latency and the function. The relative timing of the internal events may change, however, the overall behavior of the circuit is preserved. In subsection 3.2.1, the original seminal paper on retiming published by Leiserson and Saxe is reviewed and a survey of more recent work in the field of retiming is presented in subsection 3.2.2.

3.2.1. Overview of the Retiming Process

Leiserson and Saxe originally showed in 1981 [12] that it is possible to obtain a functionally equivalent sequential circuit that operates faster by changing the locations of the registers according to a set of rules. A methodology for optimally determining the location of these registers such that the minimum clock period is achieved while retaining the system latency and functionality is known as **retiming** [12] and is reviewed in [14].

In retiming, a circuit description containing logic delays and path connections is transformed into a directed graph in which the vertices represent logic delays and the edges between these vertices represent connections between these logic elements. Weights are assigned to each edge, defining the number of registers between logic elements. A zero weight edge, for example, shows that no registers exist between those two vertices. Edges are assumed to have no effect on the path delays and each data path is defined from vertex to vertex. In the Leiserson-Saxe algorithm [14], the edge weights are varied such that the function and latency of the original system is preserved, while tracking the effective latency at each vertex. The location of these registers are constrained by specific retiming rules, which ensure functional correctness and equivalency. Note that in minimizing the clock period, a successive search is performed to determine the minimum feasible clock period rather than choosing a register allocation that satisfies a specified clock period. The original sequential circuit from [14] is shown in Figure 19a along with a retimed version which is depicted in Figure 19b. The minimum clock period of the original circuit is 24 **time units** (tu) and the retimed version has a clock period of 13 tu. The clock period is decreased by changing the register locations such that the data paths with large delays are broken into smaller paths with less delay. This process reduces the delay of the limiting critical path which constrains the overall system clock period. The critical paths in the original and retimed versions of the circuit are shown in bold in Figure 19.

An important step in the retiming process is producing a set of edge weights that satisfy a specific set of constraints. The process of solving for a set of feasible edge weights requires a solution for $|E|$ unknowns, where $|E|$ is the number of edges in the sequential circuit. A key aspect of retiming, originally described in [12], is the use of **vertex lags** to reduce the number of unknowns from $|E|$ to $|V|$ based on the observation that, to preserve functional equivalency, edge weight changes cannot be made independently.

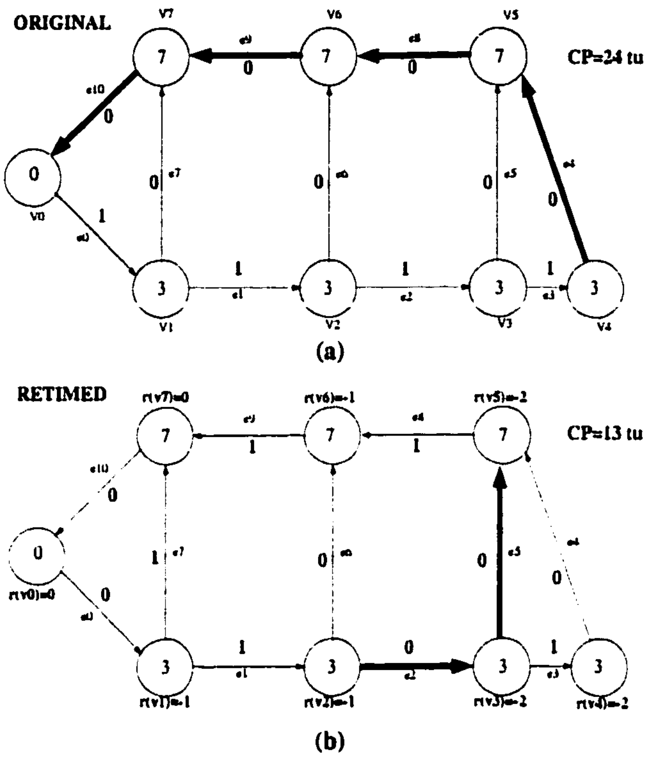


Figure 19. a) The original graph introduced in [14] and b) its retimed version. Note that the sequential latency is four clock periods in both cases. $r(v_n)$ denotes the lag of vertex n after retiming.

The **lag** of a vertex v, $r(v)$, is defined in [14] and adhered to herein. The vertex lag function $r()$ plays a fundamental role throughout the entire retiming process, and is therefore repeated here. The retiming process does not change the vertex delays $d()$, however, the weights of the edges are changed according to the lags assigned to the vertices based on the following formula,

$$w_r(e) = w(e) + r(e.end) - r(e.start), \tag{3.5}$$

where e is an edge and $w(e)$ and $w_r(e)$ are the weight of edge e before and after retiming, respectively. Therefore, the retiming process can be thought of as determining $|V|$ integer vertex lags, $r(0) \ldots r(|V|-1)$, according to the retiming rules defined in [14]. An observation of (3.5) shows that instead of calculating $|E|$ edge weight values during retiming, $|V|$ vertex lags are calculated. Although the result is identical and yields a functionally equivalent circuit, the computational effort is significantly reduced since typically $|V|$ is much less than $|E|$ and the CPU time of these retiming algorithms depend polynomially on both of these unknowns, $|V|$ and $|E|$. The $r()$ function is utilized since the edge weights do not change independently during the retiming process. The $r()$ function represents lags attached to the vertices denoting relative edge weight adjustments, as shown in Figure 20. Let $r(1)$ denote the lag of vertex 1. As shown in Figure 20, changing the lag of the vertex from zero to one affects the weights of all of the edges connected to this vertex. Increasing the lag of a vertex by one has the effect of decreasing the weights of all the edges in front of the vertex by one and increasing the weight of all the edges behind the vertex by one. The retiming process consists of applying these vertex lag adjustments throughout the entire synchronous circuit to minimize the imbalance among all the path delays.

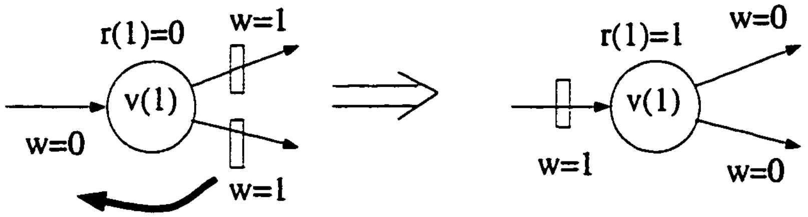


Figure 20. The lag function changes the edge weights while preserving the circuit function. Increasing the lag of a vertex by one has the effect of increasing the weights of all edges by one connected in front of this vertex and decreasing the weights of all edges by one connected behind this vertex.

The retiming process is based on (3.5) and two primary rules: 1) edge non-negativity constraints and 2) long path constraints. These two rules generate inequalities in terms of vertex lags. The set of linear inequalities are solved using the Bellman-Ford algorithm [47] and the resulting lags are used to calculate the retimed edge weights, $w_r(e)$, from (3.5). The two aforementioned retiming rules are repeated here: the edge non-negativity constraint is

$$r(u) - r(v) \leq w(e), \; \forall e : u \rightarrow v, \tag{3.6}$$

which ensures non-negative weights on each edge. The long path constraint is

$$r(u) - r(v) \leq W(u,v) - 1, \; \forall u, v: \; D(u,v) > c, \tag{3.7}$$

where $W(u,v)$ and $D(u,v)$ are matrices containing the total weight and the total delay of the path from vertex u and vertex v, respectively, and c is the target clock period. The last rule states that a register must be placed on all paths with a delay greater than the desired delay, *i.e.*, a path with an excessively large delay must be broken into smaller paths by placing a register in the middle. Note that in calculating the minimum clock period of the circuit, a binary search is performed and the minimum achievable clock period is selected as the clock period of the circuit.

3.2.2. More Recent Work in the Field of Retiming

The field of retiming has developed to include register minimization and improved propagation delay models [13, 14]. A single value is assigned to each vertex and edge, representing the delay of the logic elements and the number of registers between the logic elements, respectively. Using this approach, the problem of clock period minimization for synchronous circuits can be solved using the aforementioned Bellman-Ford algorithm [47]. Similar ideas are used to perform minimum clock pipelining, minimum clock retiming, and approximate minimum clock retiming [54]. Papaefthymiou [66] shows that the total delay of the **cycles** (paths initiating and terminating at the same vertex) in a graph representing a synchronous circuit plays a significant role in defining the maximum clock frequency of a synchronous circuit. Upper and lower bounds for the clock period are derived using the edge weights and total delays of the cycles. The lower bound on the clock period is characterized as the maximum ratio of the delay to weight for any cycle in the graph. In Figure 19a, for example, there are four cycles with total delays (10,20,30,33) and total weights (1,2,3,4). Let C_0 through C_4 represent these cycles, where the lowest index represents the innermost cycle. Thus, C_0 has a total delay $d(C_0)$ and weight $w(C_0)$,

$$\begin{aligned} d(C_0 = v_0 \rightarrow e_0 \rightarrow v_1 \rightarrow e_7 \rightarrow v_7 \rightarrow e_{10}) = 10, \\ w(C_0 = v_0 \rightarrow e_0 \rightarrow v_1 \rightarrow e_7 \rightarrow v_7 \rightarrow e_{10}) = 1, \end{aligned} \tag{3.8}$$

where $d()$ and $w()$ are the delay and weight functions, respectively. Thus, the data must travel C_0 (10 tu) in one clock cycle, cycle two (20 tu) in two clock cycles, etc. The lower bound on the clock period of this circuit is therefore [66]

$$T_{min} = \max\left\{\frac{10}{1}, \frac{20}{2}, \frac{30}{3}, \frac{33}{4}\right\} = 10, \tag{3.9}$$

where T_{min} is the minimum achievable clock period of the circuit. Note that this clock period can only be achieved when the delays are properly distributed. In a circuit in which the delays are non-uniform, this minimum limit is not achievable. For example, the retimed version of Figure 19a has a clock period of 13 tu, whereas the theoretical minimum is 10 tu.

Ishii and Leiserson [67] and Sakallah, Mudge and Olukotun [24] develop a theory for analyzing level-clocked circuitry. In [67], an algorithm is presented for verifying timing in VLSI circuits. Propagation delays of latches are considered to be constant and minimum propagation delays of the logic elements are not considered. Retiming algorithms for synchronous circuits consisting of single-phase [15], two-phase [31], and multi-phase [17] flip-flops have also been developed.

Retiming can be made more effective by combining it with combinational optimization. Algorithms have been proposed by DeMicheli [18] to minimize the cycle time using logic transformations, such as elimination, resubstitution, extraction, and decomposition, while also retiming the synchronous circuit. Another proposed method is to temporarily shift the registers to the periphery of the synchronous circuit, perform logic minimization on the purely combinatorial circuit, and return the registers to within the circuitry [46]. Although this methodology temporarily creates negative edge weights, violating the aforementioned retiming rules [see (3.6)], once the registers are replaced, the negative edge weights are eliminated. This method of performing logic optimization on the combinatorial circuit between the registers is defined as **resynthesis** [46, 68]. Application of this method to multi-phase pipelines is discussed in [69].

Retiming to minimize the number of registers in a sequential circuit was proposed by Leiserson and Saxe and is shown to be equivalent to state minimization in FSMs [14]. Recently, retiming has been extended to cover gated-clocks and precharged circuit structures [32]. Retiming to decrease power dissipation by minimizing the switching activity within the synchronous circuit while preserving functional equivalency has also been demonstrated [35].

Although retiming can significantly reduce the clock period of a synchronous circuit, its usefulness is limited by the nature of the circuit structure. The application of retiming to highly-structured circuits such as FIR filters is studied in [70] by combining retiming with algebraic speed-up techniques. This method is based on the ERB (eliminating retiming bottlenecks) method introduced in [71] in which the computational structure of the original circuit is changed so as to enhance its ability to be retimed.

Chapter 4. Register Electrical Characteristics (RECs)

As described in Chapter 2, the primary objective of this dissertation is to introduce a set of algorithms to optimize synchronous circuits by incorporating low-level circuit issues into the design, analysis, and optimization process. Among these low-level issues are the delays of the registers, the local clock skew, the interconnect, and the logic gates. Before these issues can be successfully incorporated into the synchronous optimization process, these issues must be properly modeled. The main topic of this chapter is the systematic modeling of these low-level circuit characteristics.

In Section 4.1, these low-level circuit effects (called Register Electrical Characteristics or REC's) are introduced and the reason for the introduction of the REC's is discussed. The low-level circuit effects that constitute the REC structure are individually discussed in the following sections: register-related delays, including the set-up time, hold time, clock-to-Q delay, as well as the clock delays are discussed in Section 4.2. The interconnect delay due to the distributed *RC* impedances of the interconnect are discussed in Section 4.3. The dependence of the register and logic element delays on the load are formulated and discussed in Section 4.4. The REC model is developed in Section 4.5. According to the REC model, the paths are defined from edge-to-edge. The calculation of the path delays based on the REC model discussed in the previous section is presented in Section 4.6.

4.1. The Importance of an Effective Timing Model

A synchronous circuit is composed of a combination of logic and synchronizing elements. Examples of typical logic elements include AND, OR, XOR, and NOT gates or more complex structures, whereas synchronizing elements generally consist of one or more registers located between the logic elements. A certain amount of time is required to propagate a signal through the logic elements, since a gate is composed of transistors which require some time to switch on and off. Transmitting a data signal from the input of a logic element to its output also requires a certain amount of time. Previous work in the field [12, 14] has assumed that the delay of the registers, the setup and hold times of the data signals traveling between the logic elements, have negligible delays. However, this is not true. Not only do the registers have non-negligible delay, but these delays are also *different* (or variable). Furthermore, the interconnect between the logic elements and the registers also typically have non-negligible delays.

In a fully synchronous system, the registers receive the global synchronization signal (the clock signal) from a clock distribution network. In a synchronous circuit, the clock source is located at a certain location and the clock signals are distributed via a clock distribution tree using distributed buffers and interconnect. Ideally, in a zero skew circuit, each register receives the clock signal at precisely the same time. In practice, however, the interconnect that is distributing the clock signal has delays and every register receives the clock signal at a different time. Furthermore, in certain applications, it is advantageous to schedule the clock arrival times in order to increase circuit performance within a VLSI circuit [22, 72]. A significant difference between the clock delays of an arbitrary pair of registers may exist. Therefore, the timing characteristics of these registers typically differ.

Furthermore, the delay of a logic element is dependent upon its load. For example, if a register is connected to the output of a logic gate, it will present a different load characteristic than if the gate is connected to another gate. If a register is connected, the output of the logic element is loaded by the input capacitance of the register. Alternatively, if there is no register at the output of the logic element, the output of the logic element is loaded with the distributed *RC* impedance of the interconnect and the input capacitance of the following logic element after the interconnect. These two loads may differ significantly. Thus, if a synchronous circuit optimization technique changes the number of registers at the output of a logic element, the delay of that logic element may be significantly changed during the optimization process.

By omitting the aforementioned *nonidealities* of the interconnect, registers, logic elements, and the clock distribution network, the accuracy of the simulation of a synchronous circuit may be significantly limited. In the worst case, these non-idealities may be sufficiently significant that ignoring these attributes may make the optimization entirely useless. In this dissertation, these non-idealities are modeled and incorporated into the proposed algorithms. These non-idealities are first reviewed in the following section. As described later, these non-idealities are modeled by attaching a number set, *Register Electrical Characteristics* (REC), to each edge of the synchronous circuit. The REC's are used to characterize and quantify the aforementioned non-idealities.

4.2. Register Delays

A typical flip flop (or register) used in a synchronous circuit is depicted in Figure 21. This flip flop is composed of two latches. The initial latch is used to hold the data and the second latch is used to transfer the data to the output upon arrival of the clock signal. The characteristics that define the operation of this flip-flop are the subject of this section. The latching and clocking characteristics of the flip flops are considered in the following subsections. The register-related delays such as the set-up delay, the hold time, the clock-to-Q delay, and the clock delay are the topic of subsection 4.2.1. Although these four delays are register-related, these delays must be divided into two different categories: The set-up, hold, and clock-to-Q delays are related to the structure and impedance characteristics of the flip-flop and are discussed in Subsection 4.2.2, whereas the clock delay is associated with the structure of the clocking circuitry rather than the actual flip flop and is described in Subsection 4.2.3.

4.2.1. $T_{Set\text{-}up}$, T_{Hold}, $T_{C \rightarrow Q}$, and T_{CD}

The data signal departing from the input of LATCH1 must be allowed sufficient time to pass the first transmission gate and latch into the initial transmission gate. This set-up time T_{Set-up} can be described in terms of physical parameters related to the transistors in the first latch [73] or can be derived from SPICE simulations. The input

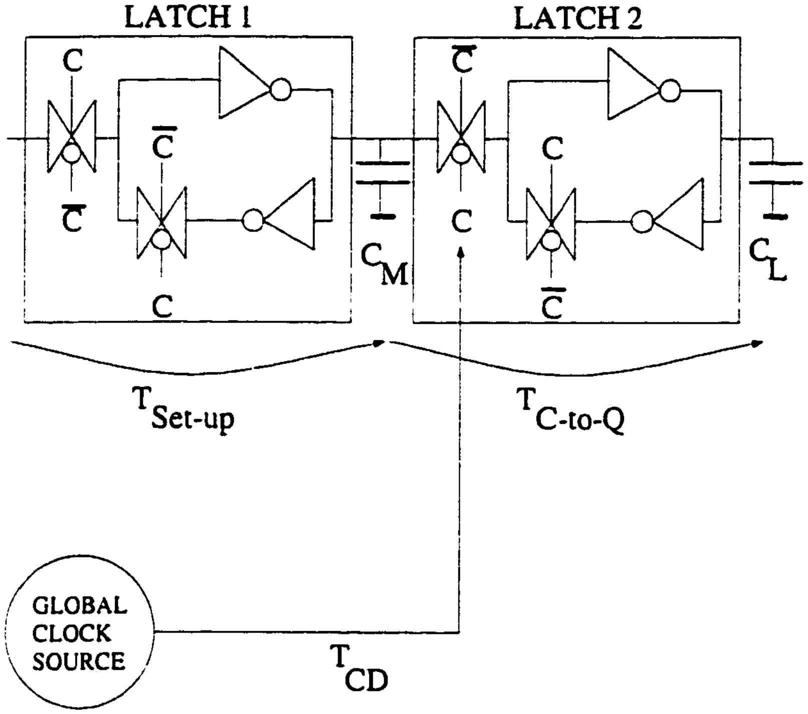


Figure 21. A typical edge-triggered master-slave flip flop is composed of two latches. The set-up time is the time required for the data to successfully latch into the first latch. The hold time is the duration during which the data at the input of the register must be stable after the arrival of the clock signal. The clock-to-Q time is the time required for the data to appear at the output of the flip flop upon arrival of the incoming clock signal.

signal must be stable for a short time T_{Hold} after the clock transition [74] to ensure metastable operation does not occur. Once the data is latched into the initial latch, the time required for the signal to appear at the output of LATCH2 after the clock signal arrives is the clock-to-Q delay ($T_{C \to Q}$) and is related to the delay of the transmission gate and an inverter (if a second buffer is placed at the output of the latch, the delay of this buffer must also be included in $T_{C \to Q}$). The setup time, hold time, and clock-to-Q

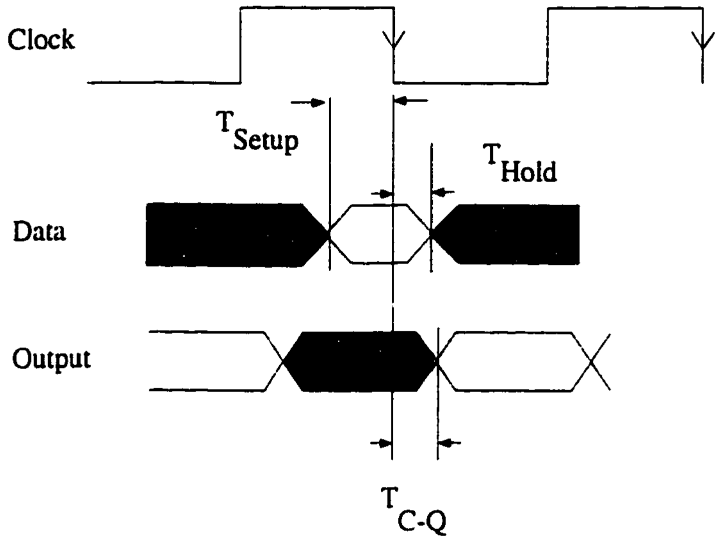


Figure 22. Timing diagrams of the flip-flop shown in Figure 21. The data signal must be stable for T_{Setup} before and T_{Hold} after the arrival of the clock signal. The valid data signal appears at the output of the flip flop $T_{Clock-to-Q}$ after the clock signal arrives.

delay are presented in the timing diagram showm in Figure 22.

An observation noted from Figure 21 is that the initial latch in this structure has a constant loading C_M on its output, since this latch is isolated from the output of the register. This constant load is defined by the drain capacitances of the inverters in the initial latch and transmission gate and the gate capacitance of the following inverter. The delay of the second latch, however, depends upon the output load C_L. Since the output load C_L may change depending upon the logic structure connected to the output of the latch, T_{C-Q} is load dependent. An additional inverter may be used at the output of the second latch to decrease the load dependancy of T_{C-Q}. However, this approach may increase the overall delay of the entire flip flop.

The flip flop receives its clock signal from a global clock source located at a specific location in the circuit. The global clock signal is distributed throughout the entire circuit by a clock distribution network [42]. A flip flop, such as shown in Figure 21, is located at a specific physical location within the circuit. The clock signal C reaches the clock input of the flip flop with a certain delay. This clock delay is characterized as T_{CD}.

The non-idealities of the registers in a synchronous circuit must be properly modeled to successfully optimize a synchronous circuit. Therefore, the T_{Set-up}, T_{Hold}, $T_{C \rightarrow Q}$, and T_{CD} parameters and the load dependancy of the $T_{C \rightarrow Q}$ delay must be incorporated into the REC model. The effects of these register non-idealities are incorporated into the synchronous optimization process in this dissertation: REC Level 1 includes the T_{Set-up}, T_{Hold}, $T_{C \rightarrow Q}$, and T_{CD} values. As presented in future work, REC Level 2 adds additional parameters to characterize the load dependancy of $T_{C \rightarrow Q}$.

4.2.2. Estimating $\mathbf{T_{Set\text{-}up}}$, $\mathbf{T_{Hold}}$, and $\mathbf{T_{C \rightarrow Q}}$

The setup time of a flip-flop T_{Setup} is the time required for the flip-flop to enter the irreversible latching state. An analytical formula for this time is complicated to derive and is not presented here. Interested readers can find the derivation of the formula for latching a CMOS Bistable NAND gate based register in [73]. A simple calculation of the setup time of the flip-flop shown in Figure 21 is the delay through a transmission gate plus two times the delay of the CMOS inverter located in the first stage of the flip-flop. This time is required for a data signal arriving at the input of LATCH1 to propogate through the transmission gate at the input of LATCH1 and through the two cascaded inverters, thereby entering an irreversible latching state within LATCH1 until the next clock signal arrives.

As mentioned earlier, the valid data signal at the input of a register must be stable for a short period of time before (T_{Setup}) and after (T_{Hold}) the clock transition [74]. The hold time T_{Hold} is the time required for a flip flop to lock (or hold) the data in the cascasded inverter pair. For the hold time, the input data signal must be stable so as to ensure that the data signal passes through the transmission gate connecting the inverter pair and irreversibly latches. Otherwise, the data signal may not latch or metastable operation may occur. Typically, the hold time is small and, in some cases, negative, denoting the fast latching capability of the flip flop due to the lead/lag relationship between the clock and data signals.

T_{Setup} does not change due to loading on the output of the flip-flop since the capacitive loading at the intermediate node (C_M) is constant. T_{Setup} can be calculated using SPICE, or derived from standard cell libraries.

The clock-to-Q delay of the flip-flop is dependent on the loading at the output of the register. Specifically, when the data is transferred from LATCH1 to LATCH2, the transmission gate in the LATCH1 stage is ON, thereby completing the feedback loop, since the transmission gate between the two latches is ON, transferring the data from

LATCH1 to LATCH2. Note that before the clock signal arrives, the data is already available at the input of the transmission gate connecting the two latches. Therefore, the clock-to-Q time is entirely dependent on the time required by the top inverter in LATCH2 to charge the output load and the input capacitance of the bottom inverter. A simple approach for estimating low-to-high (τ_{PHL}) and high-to-low (τ_{PLH}) 50% delay values is shown in [74]. Note that the 50% delay is defined as the difference between the time for the input to reach 50% of the maximum value and the output to reach 50% of the maximum value. In [74], τ_{PHL} and τ_{PLH} are defined as

$$\begin{aligned} \tau_{PHL} &= C_{Load}\frac{(V_{OH} - V_{50\%})}{I_{avg,HL}}, \\ \tau_{PLH} &= C_{Load}\frac{(V_{50\%} - V_{OL})}{I_{avg,LH}}, \end{aligned} \tag{4.1}$$

where V_{OL}, V_{OH}, $V_{50\%}$, $I_{avg,LH}$, and $I_{avg,HL}$ are process and design dependent values. Specifically, V_{OL} is the highest voltage that is interpreted as logic 0, V_{OH} is the lowest voltage that is interpreted as logic 1, and $V_{50\%}$ is the average of the two highest and lowest voltage existing in the circuit (typically, $V_{DD}/2$). $I_{avg,LH}$, and $I_{avg,HL}$ are the average current draw from the voltage source during the transition from low to high and high to low, respectively. Approximating the delay as the average of τ_{PHL} and τ_{PLH}, the following formula can be derived for the clock-to-Q delay:

$$T_{C\rightarrow Q} = T_{C\rightarrow Q_0} + m \times C_{Load}, \tag{4.2}$$

where $T_{C\rightarrow Q_0}$ is the intrinsic value of the clock-to-Q delay, C_{Load} is the load capacitance, and m is a process-dependent parameter describing the load dependence of the clock-to-Q delay in terms of the device parameters related to the top inverter of LATCH2. The parameter m can be derived from SPICE simulations or from the timing data in a standard cell library. Note that the load dependence of the clock-to-Q output signal can be decreased by inserting a single buffer or tapered buffers at the output of the second latch [75], thereby increasing the output drive capability of the latch.

4.2.3. Estimating T_{CD} from the Layout

A procedure is described to demonstrate how clock delays can be estimated for a practical circuit. It is assumed that an integrated circuit can be partitioned into regions of similar clock delay. Each clock delay region is composed of both a deterministic and a probabilistic delay component, as shown below:

$$T_{CD}(region = n) = T_{CD_{const}}(region = n) + \mathfrak{T}, \tag{4.3}$$

where $\mathfrak{T}$ is a uniformly-distributed random variable and $T_{CD_{const}}(region = n)$ is the deterministic clock delay attached to region n.

The nondeterministic statistically-based component of the clock path delay, represented by the delay component $\mathfrak{T}$, is due to variations in process parameters within the integrated circuit. Since transistor parameters, such as the channel mobility, threshold voltage, and oxide thickness, may vary across the die, some variations of the clock delay are expected. These variations, however, tend to be small for registers belonging to the same local data path (and therefore physically close to each other). Methods have been proposed in the literature for reducing the process dependence of the clock skew to less than 10% of the total path delay [76].

The deterministic component of the clock delay for a region n $[T_{CD_{const}}(region = n)]$ can be calculated based on geometric distance information derived from the circuit layout. The distance information from a source node to any sink node in an IC layout can be obtained either from routing information or using exploratory Steiner tree approaches [77, 78]. Calculation of the interconnect resistances and capacitances directly from the IC layout or Steiner tree can be achieved with standard circuit extraction techniques [79]. With this information, well known techniques can be employed to estimate the individual clock delays (*e.g.*, [75, 80–83]).

As observed from (4.3), each region n is attached a deterministic clock delay $T_{CD_{const}}$ characterizing that region. A floorplan of a simple integrated circuit consisting of six clock delay regions is depicted in Figure 23. As an example, assume that the total interconnect capacitance of the clock line driving region A is $C_{int} = 300$ fF, the total interconnect resistance of the clock line is $R_{int} = 200\ \Omega$, the on-resistance of the clock source is $R_{tr} = 300\ \Omega$, and this clock line drives 100 registers (four NMOS and four PMOS devices per register). The geometric size of the NMOS and PMOS devices are 10 μm x 1 μm and 20 μm x 1 μm, respectively. Therefore, the total load capacitance C_L of the module A can be approximated by (ignoring bulk capacitance)

$$C_L = 100 \times 4(W_P L_P + W_N L_N)\frac{\epsilon_{ox}}{t_{ox}} = 21\ pF, \tag{4.4}$$

where W_P and W_N are the gate widths of the PMOS and NMOS load transistors, L_P and L_N are the gate lengths of the PMOS and NMOS load transistors, ϵ_{ox} is the permittivity of the gate oxide, and t_{ox} is the gate oxide thickness (a value of 200 Å is assumed). Therefore, the deterministic component of the clock delay (defined at the 50% point) from the clock source to module A can be estimated as [75, 82]

$$T_{CD_{const}}(region = A) = 0.4R_{int}C_{int} + 0.7(R_{tr}C_{int} + R_{tr}C_L + R_{int}C_L) =$$

$$0.4 \times 60\ ps + 0.7(90 + 6300 + 4200)\ ps \approx 7.4\ ns.$$

This simple example presents an approach for estimating the delays within a clock distribution network. More sophisticated approaches for estimating IC area and performance characteristics can be found in [84–86].

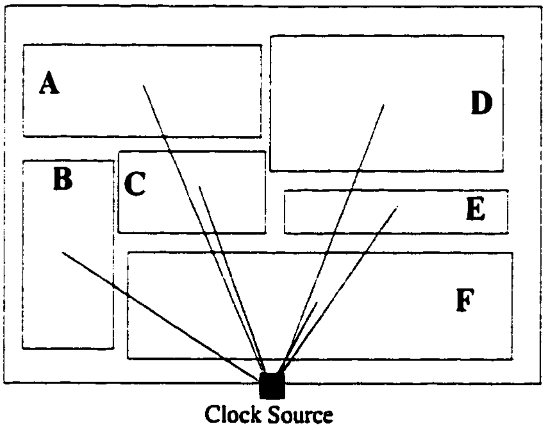


Figure 23. A block diagram of an example integrated circuit layout. The chip area is assumed to be partitioned into regions of similar clock delay.

4.3. Interconnect Delays

A section of interconnect between two nodes in a circuit can be described as having a delay T_{Int} caused by a distributed *RC* impedance with distributed resistance R_{Int} and distributed capacitance C_{Int}, respectively. Since each edge in a graph representation of a synchronous circuit represents an interconnect line, every edge e_n can be thought of as being a distributed *RC* line with a delay of $T_{Int}(e_n)$ with a distributed resistance and capacitance of $R_{Int}(e_n)$ and $C_{Int}(e_n)$, respectively. Determining these interconnect delays forms the topic of this section. Specifically, the interconnect delay T_{Int} between two nodes in a synchronous circuit and the division of T_{Int} into T_{Int1} and T_{Int2} in the case where a register exists is discussed in Subsection 4.3.1. Estimating the interconnect delay from a circuit layout is discussed in Subsection 4.3.2. In subsection 4.3.3, the REC Level 2 approach is described for modeling the interconnect delays.

4.3.1. T_{Int1} and T_{Int2}

In the event that a register exists, the interconnect delay of edge e_n is assumed to be separated into two values of $T_{Int1}(e_n)$ and $T_{Int2}(e_n)$, where T_{Int1} and T_{Int2} are the pre-register and post-register delays, respectively. In Figure 24, a register located between two vertices is depicted. The pre-register interconnect delay T_{Int1} and the post-interconnect delay T_{Int2} are caused by the distributed *RC* impedances of the connection elements between the vertices and the register (*e.g.*, a metal layer in a VLSI integrated circuit). As explained in the following subsection, T_{Int1} and T_{Int2} depend not only on

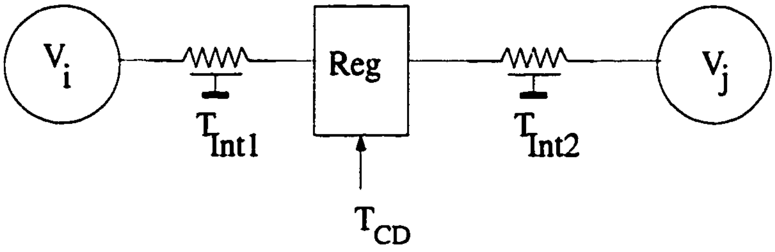


Figure 24. The pre-register and post-register interconnect delays are caused by the distributed *RC* impedance of the connections between the register and the logic elements.

the distributed *RC* impedance, but also the input capacitance of the logic elements and/or the registers, and the output impedance of the driving logic element and/or the register.

4.3.2. T_{Int} Calculated from the Layout

The interconnect resistance can be calculated according to the following formula:

$$R_{int} = \rho \frac{L_{int}}{W_{int} H_{int}}, \tag{4.5}$$

where ρ is the resistivity of the interconnect (for example, 3 $\mu\Omega cm$ for aluminum, 1000 $\mu\Omega cm$ for polysilicon), and L_{Int}, W_{Int}, and H_{Int} are the length, width and height of the interconnection [75].

Interconnect capacitance can be approximated with a maximum 10% error over a wide range of physical parameters using the following empirical formula [82].

$$C_{Int} = \epsilon_{ox} L_{Int} \left[1.15\left(\frac{W_{Int}}{t_{ox}}\right) + 2.80\left(\frac{H_{Int}}{t_{ox}}\right)^{0.222} + \tag{4.6}$$

$$\left[0.06\left(\frac{W_{Int}}{t_{ox}}\right) + 1.66\left(\frac{H_{Int}}{t_{ox}}\right) - 0.14\left(\frac{H_{Int}}{t_{ox}}\right)^{0.222}\right]\left(\frac{t_{ox}}{W_{sp}}\right)^{1.34}\right],$$

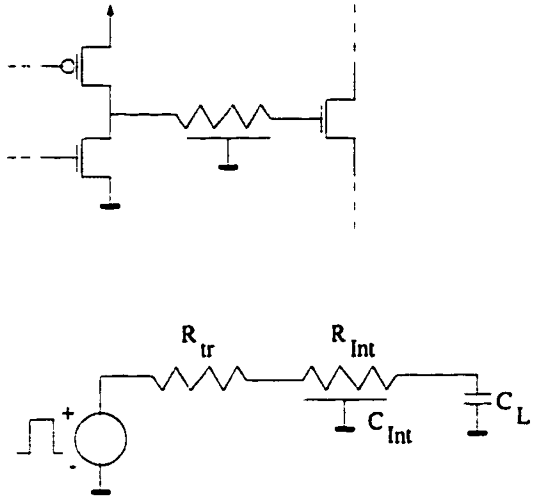


Figure 25. A logic stage (or a register) connected to another logic stage or register through an interconnect. The 50% interconnect delay T_{Int} caused by the RC impedance of the interconnect and the load capacitance is calculated based on physical parameters of the interconnect [75].

where H_{Int}, t_{ox}, W_{sp}, and ϵ_{ox} are process dependent parameters, L_{Int} is the interconnect length, and W_{Int} is the interconnect width [75].

The circuit shown in Figure 25 characterizes a typical CMOS logic block (or a register) connected to another logic block (or a register) through an interconnect [75]. The 50% delay through the interconnect can be approximated by the following formula [75, 82],

$$T_{Int\,50\%} = 0.4R_{Int}C_{Int} + 0.7(R_{tr}C_{Int} + R_{tr}C_L + R_{Int}C_L), \tag{4.7}$$

where R_{tr} is the ON resistance of the MOS transistor driving the interconnect, C_L is the load capacitance of the input of the logic circuit at the end of the interconnect, and R_{Int} and C_{Int} are the interconnect resistance and capacitance, respectively. Note that although R_{tr} is voltage dependent, the average value can be used as an approximation. The approximate value of R_{tr} can be calculated as [75]:

$$R_{tr} \approx \frac{L_{tr}/W_{tr}}{\mu C_{gox}(V_{DD} - V_T)}, \tag{4.8}$$

where L_{tr}, W_{tr}, V_T, μ, and C_{gox} are the length, width, threshold, mobility, and oxide capacitance of the transistor driving the interconnect.

When (4.5) and (4.6) are combined and incorporated into (4.7), the interconnect delay can be approximately expressed in terms of the primary design parameters, L_{Int} and W_{Int}:

$$T_{50\%} = k_1 L_{Int}^2 + k_2 L_{Int}^2 \frac{1}{W_{Int}} + k_3 L_{Int} W_{Int} + k_4 L_{Int} + k_5 C_L + k_6 \frac{L_{Int}}{W_{Int}} C_L, \tag{4.9}$$

where k_1, k_2, k_3, ... , k_n are process dependent constants for a specific driving transistor size, oxide thickness, interconnect height, and interconnect resistivity. Given a specific interconnect width, for a significantly large interconnect length and small capacitive loads, the interconnect length dominates the delay as shown below:

$$T_{50\%} \propto L_{Int}^2, \tag{4.10}$$

whereas for significantly large capacitive loads and small interconnect length, the load capacitance dominates the delay as follows:

$$T_{50\%} \propto C_L, \tag{4.11}$$

and in general, for comparable loading from the output capacitance and the interconnect capacitance, (4.9) can be summarized as

$$T_{50\%} \propto k_7 L_{Int}^2 + k_8 C_L, \tag{4.12}$$

where k_7 and k_8 are process dependent constants which include the effects of the interconnect width. An important observation from (4.12) is the square dependence of the interconnect delay on interconnect length. This behavior is due to both the interconnect resistance and capacitance being linearly dependent on the interconnect length, thereby making the *RC* constant and consequently the interconnect delay a square dependence on the interconnect length.

4.3.3. Load-Dependant Model for Interconnect Delays

As described in Subsection 4.3.2, the interconnect delays can be described in terms of the output drive of the logic gate, distributed interconnect resistance and capacitance, and the load capacitance. The 50% interconnect delay in terms of these parameters are approximated by

$$T_{Int} = 0.4 R_{Int} C_{Int} + 0.7 (R_{tr} C_{Int} + R_{tr} C_L + R_{Int} C_L), \tag{4.13}$$

where R_{Int} and C_{Int} are the interconnect resistance and capacitance, respectively [82], R_{tr} is the on resistance of the MOS transistor driving the interconnect, and C_L is the load capacitance. Rewriting the following formula for the interconnect delay,

$$T_{Int} = 0.4 R_{Int} C_{Int} + 0.7 R_{tr} C_{Int} + 0.7 R_{tr} C_L + 0.7 R_{Int} C_L, \tag{4.14}$$

which can be formulated as

$$T_{Int} = 0.7R_{tr}(C_{Int} + C_L) + 0.4R_{Int}(C_{Int} + C_L) + 0.3R_{Int}C_L. \tag{4.15}$$

R_{tr} and R_{Int}, (4.15) can be approximated in terms of a *load-dependant* m parameter and the sum of the capacitances being driven as follows:

$$T_{Int} = m(C_{Int} + C_L). \tag{4.16}$$

Note that the third term $0.3R_{Int}C_L$ does not depend on $(C_{Int} + C_L)$, however, by appropriately choosing the m parameter in (4.16), the error can be minimized. A more accurate model of the interconnect delay could be developed by using two m parameters if higher accuracy is required as follows:

$$T_{Int} = m_1C_{Int} + m_2C_L. \tag{4.17}$$

In this dissertation, the model in (4.16) is used.

4.4. Non-Ideal Logic and Register Delays

To model the load dependence of the register delays, every flip flop is assigned an output slope m which is a function of the output drive capability (or output transconductance) of the inverter in the second latch. An input capacitance c is also assigned to each flip flop which is a function of the drain capacitances of the first transmission gate, as shown in Figure 21.

A typical CMOS logic element (such as a NAND gate) is depicted in Figure 26. A logic element u can be characterized by its intrinsic delay $d(u)$, input load $c(u)$,

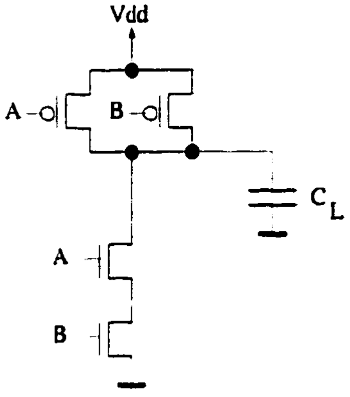


Figure 26. A CMOS NAND gate. Note that the delay of this element is dependent on the output load. The element has an output slope m characterizing this load dependence.

and output slope $m(u)$. It is possible to use a different load for each input to obtain more accurate results. However, the input load is assumed to be the same. The output slope $m(u)$ is a function of the drive capability of the transistors and is related to design parameters such as the W/L ratio of the transistors as well as process parameters such as the threshold voltage of the transistors V_T, carrier mobility μ, and the oxide thickness C_{ox} [75].

An inverter is depicted in Figure 25, which drives a capacitive load of magnitude C_L through an RC interconnect. For this structure, the 50% delay is defined by (4.7). A similar formula could be derived for any logic gate, such as the NAND gate shown in Figure 26. From (4.7), it is observed that the $m(u)$ parameter is linearly dependent on the W/L ratio of the transistors as formulated by (4.8). For the NAND gate, the $m(u)$ parameter is also dependent on the sizes of the NMOS and PMOS transistors.

A high performance design practice is to use increasingly larger (or tapered) NMOS transistors [87, 88] from the bottom to the top of the serial chain of N channel transistors to balance the capacitance and resistance effects of the transistors so as to obtain the highest speed from the NAND structure. In general,

$$m(u) = f(W_p/L_p, W_n/L_n, V_{TP}, V_{TN}, \mu_p, \mu_n, C_{ox}). \tag{4.18}$$

The intrinsic delay $d(u)$ of the logic element is related to the process parameters and geometric dimensions of the logic element. The input capacitance of the element $c(u)$ is dependent on the sizes of the gate and the oxide thickness. All of these three parameters can be derived from SPICE simulations.

In this section, the model for the load dependent logic elements and registers is presented. Modeling of the load dependancy in general is mentioned in Subsection 4.4.1 for both the logic elements and the registers. The non-uniform logic delays for multi-input and multi-output logic elements are reviewed in Subsection 4.4.2. Synchronous optimization techniques using these more complicated models are introduced as future work in this dissertation.

4.4.1. Model of Logic Delay

For a logic element with an output slope of m driving a capacitive load of c, the 50% delay T_D can be approximated by

$$T_D = T_0 + mc, \tag{4.19}$$

where T_0 is the unloaded delay (or intrinsic delay) of that logic element [89]. Note that input waveform effects are ignored and a *fast ramp* input waveform is assumed [73, 90]. Using this additional electrical information, the delay of a logic element can be denoted as $d_0[m, c]$, where d_0 denotes the intrinsic delay of this vertex, and the $[m, c]$ pair denotes the output slope m (estimated using linear regression on the propagation delay vs. the output load curves) and the input load c of the logic element at this vertex, respectively. To specify the vertex delay information attached to a vertex (or logic element) u, the notations $d_0(u)$ (the intrinsic delay of vertex u), $m(u)$ (the slope of vertex u), and $c(u)$ (the input capacitance of vertex u) are used.

For each vertex u in the circuit attached to an edge $e : u \rightarrow v$, the delay at that vertex can be calculated as

$$d(u) = d_0(u) + m(u)C_{Tot}, \tag{4.20}$$

where e is the edge connected to the output of vertex u, $m(u)$ is the output slope of vertex u, and C_{Tot} is the total load at the output of the vertex.

4.4.2. Model of Non-Uniform Logic Element Delays

Each output of a multi-output logic element may be loaded by a different circuit, therefore, a different output slope is attached to each individual output of the logic elements. An $i \times o$ delay matrix for each vertex must be considered due to non-uniform delays through different input-to-output paths, where i and o are the number of inputs and outputs of the vertex, respectively. In Table 1, the individual path delays of a

1 bit adder	Output	
Input	S (ps)	Co (ps)
A	250	250
B	230	230
Ci	180	210

Table 1: Delay of a 1 bit adder for different input and output pairs using a 0.8 μm 5 V CMOS technology. Note the non-uniformity due to asymmetry in the logic paths within the adder.

1–bit adder among different input and output pairs are presented for a 0.8 μm, 5 volt CMOS technology. Note the non-uniform delay values due to asymmetry in the logic paths within the adder.

To fully characterize the non-uniform vertex delays, $i \times o + i + o$ different delays are attached to each vertex. Specifically, $i \times o$ values are required to model the individual signal path dependent non-uniform delays. i capacitive values are attached to the inputs to model the input load capacitances and o slopes are attached to the outputs to characterize the output slew rates. Non-uniform delays become more significant as the circuit becomes less structured, since the difference among signal path delays among different input-output pairs increases. The **granularity** of a vertex can be defined as the depth of the logic represented by this vertex. If the granularity is high, more logic elements are represented by a particular vertex. As the granularity of each vertex increases, the non-uniform signal path delay model becomes more significant. For example, a NAND gate is represented by a vertex with low granularity, whereas a 4–bit adder has a higher granularity. Non-uniform delays can be quite significant in a 4–bit

4 bit adder	Output				
Input	S0 (ps)	S1 (ps)	S2 (ps)	S3 (ps)	Co (ps)
Cin	209	384	600	813	856
A0	260	410	631	850	895
B0	229	392	615	836	836
A1	x	272	414	640	681
B1	x	239	392	620	662
A2	x	x	274	413	459
B2	x	x	239	393	448
A3	x	x	x	272	244
B3	x	x	x	239	228

Table 2: Delay of a 4–bit adder for different input and output pairs using a 0.8 μm, 5 V CMOS technology. "x" denotes the delays that cannot be measured since a change at the specified input is not propagated to the specified output.

adder. The delays of an example 4–bit adder with different input-output delays are shown in Table 2. The difference among the delays is more noticeable as compared to the 1–bit adder shown in Table 1. This disparity is due to the more complicated signal paths leading to an increased difference among the input-to-output signal path delays within the 4–bit adder.

The data listed in Tables 1 and 2 are obtained from SPICE analysis on a transmission–gate based 1–bit and 4–bit adder, respectively, using a 0.8 μm, 5 volt CMOS technology. The different input-to-output delay values emphasize the importance of considering non-uniform signal path delays. Simply using the maximum delay listed in the table as the delay of the logic element results in inaccurate estimates of the total path delays, since, as is described in the following sections, the minimum vertex delay can create *short paths*, which may cause a catastrophic circuit maloperation [23, 91].

4.5. Modeling the RECs

Research described in this dissertation focuses on modeling the aforementioned non-idealities of the clock distribution network, registers, and the interconnect by attaching a number set, the **Register Electrical Characteristic** (REC), to each edge in the graph representation of the synchronous circuit. The REC model is developed using two different complexity levels. REC Level 1 includes only the basic clock distribution, register, and interconnect delays, whereas REC Level 2 further incorporates the load dependence of the register and logic element delays. Both REC models and the derivation of the path delays under these two models are presented in the following subsections.

4.5.1. Basic REC Model (Level 1)

A variety of register types selected prior to beginning the synchronous optimization process may be used at different locations within the circuit due to the specific speed, power, and area tradeoffs peculiar to that portion of the circuit. T_{Set-up} and T_{Hold} may also change for different register cell instances. Thus, T_{Set-up} and T_{Hold} can vary per edge. Therefore, selecting a specific register to satisfy a set of performance-based design requirements changes these two parameters for each edge. A similar argument is valid for $T_{C \rightarrow Q}$, thereby requiring that variable register delays be considered.

The interconnect between a pair of logic elements or between a register and a logic element have different delays, since the elements may be located at various locations of the circuit, thereby requiring different interconnect lengths for each connection. This varying requirement for the interconnect lengths makes consideration of the interconnect delays necessary.

The clock delays, as mentioned before, play a crucial role in the performance of a synchronous circuit. The variation between the clock arrival times across a VLSI circuit creates clock skews that can be exploited to improve the performance of an IC [80, 92–94]. Alternatively, uncertainity in the clock skew values may cause maloperation in the circuit, creating race conditions. Therefore, consideration of the clock delays in synchronous circuit optimization algorithms is crucially important to achieving satisfactory accuracy in the optimization process.

These three delay components (the register, clock, and interconnect delays) form the basis for the REC Level 1 model. In the REC Level 1 model, the clock, register, and interconnect delays are modeled by attaching the following number set to each edge in the graph representation of a synchronous circuit,

$$T_{CD} : T_{Set-up}/T_{Hold}/T_{C \rightarrow Q} - T_{Int1}/T_{Int2}. \tag{4.21}$$

This number set is used to characterize an edge e in the graph representation of the circuit as shown in Figure 27. T_{CD} is the clock delay from the global clock source to each register on edge e. T_{Set-up} and T_{Hold} are the times during which the data signal must be stable before and after the clock signal, respectively, to ensure proper latching operation. T_{C-Q} is the time required for the data to appear at the output of the register upon arrival of the clock signal for all registers located on edge e, and T_{Int} is the total interconnect delay along edge e and can be considered as being composed of pre-register and post-register components, T_{Int1} and T_{Int2}, if one or more registers are located along that edge.

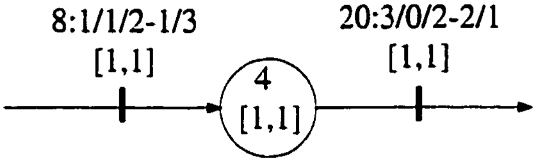


Figure 27. A path containing two edges and a vertex between the two edges. The REC values are attached to each edge to characterize the registers, interconnect, and the clock delays located along the edges.

Consider the example shown in Figure 27. The REC Level 1 value of $8 : 1/1/2-1/3$ is attached to the first edge. This REC value implies that $T_{CD} = 8$ tu (every register along this edge is in the same clock delay region and receives the global clock signal with a delay of 8 tu). Register delays are $T_{Set-up} = 1$ tu, $T_{Hold} = 1$ tu, and $T_{C-Q} = 2$ tu, *i.e.*, every register located along this edge has a set-up delay of 1 tu, a hold time of 1 tu, and a clock-to-Q delay of 2 tu. The total interconnect delay along this edge is 4 tu which is composed of the pre-register interconnect delay, $T_{Int1} = 1$ tu, and the post-register interconnect delay, $T_{Int2} = 3$ tu. Also, note that the delay of the logic element represented by the vertex shown in Figure 27 is 4 tu. This REC Level 1 Model forms the basis of the algorithms introduced in this dissertation.

4.5.2. The Enhanced REC Model (Level 2)

The clock-to-Q delay T_{C-Q} is edge dependent since each edge is connected to a different vertex, thereby changing the capacitive loading on the registers located on each edge. This variation occurs since each vertex represents a variety of possible logic elements, placing a different output load on the register driving the vertex. Hence, the same registers driving different vertex inputs will have different T_{C-Q} delays.

The enhanced register delay REC model has the following form,

$$T_{CD} : T_{Set-up}/T_{Hold}/T_{C-Q_0} - C_{Int1}/C_{Int2}\,[m, c], \tag{4.22}$$

where m denotes the output slope of each register and c denotes the input capacitance of each register located along the edge. The interconnect capacitances rather than the interconnect delays are used in the REC Level 2 model to more accurately characterize the effects of the interconnect lines. C_{Int1} and C_{Int2} are the pre-register and post-register interconnect capacitances. The parameter m is derived from the output delay vs. capacitive load curves used to characterize the logic gate delays within a standard cell library. The set-up time T_{Set-up} and hold time T_{Hold} of a register are the times required for a register to enter the irreversible latching point. T_{Set-up} can be derived in terms of the physical circuit parameters related to the shape of the clock and data signals and the gain of the positive feedback circuit within the register [73]. In this dissertation,

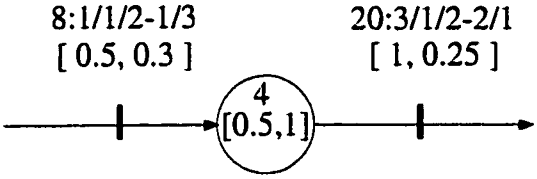


Figure 28. The graph of Figure 27 with attached load dependent parameters. The values in the brackets are the enhanced REC values *m* and *c*.

the transistor characteristics of a register associated with an edge are assumed to be constant and the shape of the clock and data signals are assumed to behave as a "fast ramp [90]." Therefore, the set-up time of the register is assumed to be constant. T_{C-Q} is a load dependent clock-to-Q delay, where T_{C-Q_0} is the unloaded delay for T_{C-Q} (or the intrinsic clock-to-Q delay). Therefore, T_{C-Q} can be modeled as

$$T_{C-Q} = T_{C-Q_0} + m(C_{Int2} + C_L), \tag{4.23}$$

where C_L is the input capacitance of the cell loading the register. Note that the total capacitive load on the output of a register is the sum of the post-register interconnect capacitance and the input capacitance of the cell loading the register. Assuming negligible interconnect resistance, a first-order approximation can be used to model a distributed *RC* impedance [83] as a single lumped capacitance. An example graph with enhanced REC values is shown in Figure 28. The values in the brackets are the enhanced REC values used to model the load-dependancy (*i.e.*, *m* and *c*) of the logic elements and registers.

Values of C_{In1}, C_{Int2}, and C_L are included in Figure 29 for the path shown in

Figure 28. According to (4.23), the sum of $C_{Int2} = 3$ **cu** (capacitive units) as shown in the REC of the first register and the input load of the vertex $C_L = 1$ cu, form the load of the first register.

The $[m, c]$ pair is depicted in Figure 29 for the logic element (the vertex). The parameter m describes the output slope of the vertex derived from a characterized standard cell library, whereas the parameter c characterizes the input load of the logic element which can be obtained from the standard cell libraries or estimated from circuit simulation. An example of a standard logic element is shown in Figure 26. For this specific element, the input capacitance (per input) consists of the sum of the NMOS and PMOS gate capacitances. Therefore, the parameter c can be calculated from the total gate area. For this case, the load c is

$$c = 2W_G L_G C_{ox}, \tag{4.24}$$

where W_G and L_G are the gate width and length and C_{ox} is the oxide capacitance.

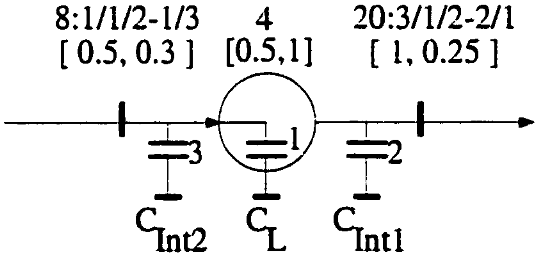


Figure 29. C_{Int2} and C_L are the loads at the output of the initial register. The circle is the logic element (or vertex) with a delay of 4 tu, and load dependent m and c values of 0.5 and 1, respectively.

In Figure 29, a logic element u is depicted by a circle with the load dependent values $[m(u), c(u)]$ of the logic element shown above the circle. The two edges e_1 (on the left) and e_2 (on the right) have $[m, C_{Int}]$ values depicted on top of the edges. The delay d_u of the logic element u shown in Figure 29 consists of an intrinsic value $d_0(u)$ (e.g., 4 tu), and a load dependent value $m(u)C_{Tot}$, as shown in (4.20). The input capacitance of the logic element u is 1 cu. The total load C_{Tot} at the output of the logic element u in Figure 29 can be calculated as

$$C_{Tot}(w(e_2) > 0) = C_{Int1}(e_2) + c(e_2), \tag{4.25}$$

where $C_{Int1}(e_2)$ is the pre-register interconnect delay of the second edge, and $c(e_2)$ is the input capacitance of the registers located along the second edge. If a register is not located on the second edge, the load changes as described by

$$C_{Tot}(w(e_2) = 0) = C_{Int1}(e_2) + C_{Int2}(e_2) + c(v), \tag{4.26}$$

where $C_{Int1}(e_2)$ and $C_{Int2}(e_2)$ are the pre-register and post-register interconnect delays on edge e_2, and $c(v)$ is the input capacitance of the logic element connected to the output of edge e_2. Note that the interconnect resistance on edge e_2 is ignored.

4.6. Path Delays

By attaching delay components to the registers located on the edges (the connections between the logic elements), the local path must be defined from edge-to-edge [19, 23]. By assigning a clock delay to each edge (see Section 4.2.3), the circuit is assumed to be partitioned into regions of similar clock delay, *i.e.*, registers that are located on the same edge are physically located within the same clock delay region. Therefore, registers that end up on the same edge after synchronous circuit optimization are assumed to have a similar clock delay. Registers that move to different edges are assumed to have the clock and register delays of the new edge.

Depending on the level of the REC model being used, the calculation of the path delays change. In the REC Level 1 Model, the load dependence of the register and logic element delays are not considered. The path delays are defined without the load dependent components when the REC Level 1 model is used. In Subsection 4.6.1, calculation of the path delays using the REC Level 1 model is studied. On the other hand, if REC Level 2 is used, the path delays are calculated by considering the load dependent values. Formulation of the path delay using REC Level 2 is described in Subsection 4.6.2.

If parallel paths exist between two edges, the minimum and maximum local data path delays, $T_{PD_{min}}$ and $T_{PD_{max}}$, are defined. $T_{PD_{min}}(i,j)$ and $T_{PD_{max}}(i,j)$ are the minimum and maximum of all of the path delays between edges e_i and e_j, respectively. $T_{PD_{max}}$ is used to determine if any long paths exist. Alternatively, $T_{PD_{min}}$ is used to determine the existence of any race conditions caused by the short paths. Register setup and hold times, interconnect delays, minimum and maximum logic delays, and the clock skew characteristic values are used to determine $T_{PD_{min}}$ and $T_{PD_{max}}$. The calculation of the long path and short path delays is described in subsections 4.6.1 and

4.6.3, respectively. If multiple registers exist along an edge, the internal path delays must also be considered. Subsection 4.6.4 is devoted to determining these internal path delays [95].

4.6.1. Path Delays using REC Level 1 Model

Using the REC level 1 model of (4.21), the local data path delay $T_{PD}(i,j)$ from edges e_i to e_j has four components as follows:

$$T_{PD}(i,j) = T_{Reg}(i,j) + T_{Int}(i,j) + T_{Logic}(i,j) + T_{Skew}(i,j). \tag{4.27}$$

$T_{Reg}(i,j)$ is the register related delay associated with path $p : e_i \rightarrow e_j$. The setup time is used to calculate the long paths, and the hold time is used to determine the permissible time window to prevent race conditions. Therefore, the minimum and maximum register delays are defined as follows:

$$\begin{aligned} T_{Reg_{min}}(i,j) &= T_{C \rightarrow Q}(i) - T_{Hold}(j), \\ T_{Reg_{max}}(i,j) &= T_{C \rightarrow Q}(i) + T_{Setup}(j). \end{aligned} \tag{4.28}$$

$T_{Int}(i,j)$ is the interconnect-related delay component for path $p : e_i \rightarrow e_j$, describing the interconnect delay before the first logic element on path p and after the last logic element on the same path. Note that the interconnect delays along the path between the first and the last logic element on path p are included in the logic delay $T_{Logic}(i,j)$. According to this definition, $T_{Int}(i,j)$ is defined as follows:

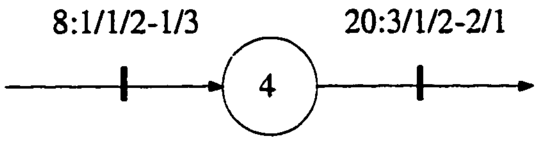
$$T_{Int}(i,j) = T_{Int2}(i) + T_{Int1}(j). \tag{4.29}$$

Note that although it is possible to consider the minimum and the maximum interconnect delays, $T_{Int_{min}}$ and $T_{Int_{max}}$, interconnect delays are assumed to be constant, for simplicity, since the accuracy improvement is negligible by using the minimum and maximum values.

$T_{Logic}(i,j)$ is the delay of the logic elements between e_i and e_j including the interconnect delay of the zero weight edges along the path between these edges. Minimum and maximum values of the logic delay, $T_{Logic_{min}}$ and $T_{Logic_{max}}$, are used to determine the path delay boundaries. Note that, although the minimum and maximum delay values for logic elements can be considered, it is assumed in this dissertation, that the intrinsic delay of the logic elements is constant. However, the load-dependancy of the delay values are included in the REC level 2 model.

$T_{Skew}(i,j)$ is the difference between the clock delays of edges e_i and e_j as defined in 2.2.

The example path shown in Figure 30 has two edges. The delay of the local data path between the two edges can be calculated using (4.27). The parameters for the first and the second edges (e_i and e_j, respectively), are as follows: For edge e_i, $T_{CD} = 8$, $T_{Set-up} = 1$, $T_{Hold} = 1$, $T_{C\to Q} = 2$, $T_{Int1} = 1$, and $T_{Int2} = 3$ tu. For edge e_j, $T_{CD} = 20$, $T_{Set-up} = 3$, $T_{Hold} = 1$, $T_{C\to Q} = 2$, $T_{Int1} = 2$, and $T_{Int2} = 1$ tu.



$$T_{PD} = T_{Reg} + T_{Int} + T_{Logic} + T_{Skew}$$

$$T_{PD_{min}} = (2-1) + (3+2) + 4 + (8-20) = -2\ tu$$

$$T_{PD_{max}} = (2+3) + (3+2) + 4 + (8-20) = 2\ tu$$

Figure 30. The path shown in Figure 27. The minimum and maximum path delays, $T_{PD_{min}}$, and $T_{PD_{max}}$ between the two edges are calculated using (4.27).

In Figure 30, the components of the path delay T_{PD} are grouped into register delay T_{Reg}, interconnect delay T_{Int}, logic delay T_{Logic}, and the local clock skew T_{Skew}. The minimum and maximum register delays are calculated using the clock-to-Q, setup, and hold times as follows:

$$T_{Reg_{max}} = T_{C\rightarrow Q}(i) + T_{Set-up}(j) = 2 + 3 = 5\ tu,$$

$$T_{Reg_{min}} = T_{C\rightarrow Q}(i) - T_{Hold}(j) = 2 - 1 = 1\ tu.$$

The total interconnect delay is the sum of $T_{Int2}(i)$ and $T_{Int2}(j)$, therefore

$$T_{Int} = T_{Int2}(i) + T_{Int1}(j) = 3 + 2 = 5\ tu.$$

The total logic delay consists of the logic element delay of the vertex between edges which is equal to $T_{Logic} = 4$. Since there are no zero-weight edges along the path from e_i to e_j, T_{Logic} does not have any additional interconnect delay components. The clock skew between the two edges is the difference of the clock delays of these edges and is equal to

$$T_{Skew}(i,j) = T_{CD}(i) - T_{CD}(j) = 8 - 20 \ = \ -12 \ tu.$$

Therefore, the minimum and maximum path delays between edges i and j are

$$T_{PD_{min}} = T_{Reg_{min}} + T_{Int} + T_{Logic_{min}} + T_{Skew} = 1 + 5 + 4 - 12 \ = \ -2 \ tu,$$

$$T_{PD_{max}} = T_{Reg_{max}} + T_{Int} + T_{Logic_{max}} + T_{Skew} = 5 + 5 + 4 - 12 = 2 \ tu,$$

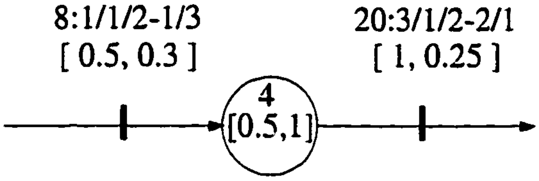
which suggests that, due to the negative $T_{PD_{min}}$, this path can cause a race condition.

4.6.2. Path Delays using REC Level 2 Model

Using the REC level 2 model of (4.22), the minimum and the maximum values of the local data path delay $T_{PD}(i,j)$ from edges e_i to e_j can be calculated as

$$\begin{aligned} T_{PD_{min}}(i,j) &= T_{C \to Q_0}(i) + m(i)[c(e_i.end) + C_{Int2}(i)] + T_{Logic_{min}}(e_i.end, e_j.start) \\ &\quad + m(e_j.start)[C_{Int1}(j) + c(j)] - T_{Hold}(j) + T_{Skew}(i,j), \\ T_{PD_{max}}(i,j) &= T_{C \to Q_0}(i) + m(i)[c(e_i.end) + C_{Int2}(i)] + T_{Logic_{max}}(e_i.end, e_j.start) \\ &\quad + m(e_j.start)[C_{Int1}(j) + c(j)] + T_{Setup}(j) + T_{Skew}(i,j), \end{aligned} \tag{4.30}$$

where $T_{Logic}(i,j)$ is the total vertex-to-vertex delay of the logic elements between the end vertex of e_i ($e_i.end$) and the start vertex of e_j ($e_j.start$) including the interconnect delays due to the capacitive loading between vertices $e_i.end$ and $e_j.start$.



$$T_{PD} = T_{Reg} + T_{Load} + T_{Logic} + T_{Skew}$$

$$T_{PD_{min}} = (2 - 1) + (0.5(1 + 3) + 0.5(2 + 0.25)) + \ 4 \ + (8 - 20) = -3.875 \ tu$$

$$T_{PD_{max}} = (2 + 3) + (0.5(1 + 3) + 0.5(2 + 0.25)) + 4 + (8 - 20) = 0.125 \ tu$$

Figure 31. The path shown in Figure 28. The path delay T_{PD} between the two edges is calculated from (4.30).

The example path shown in Figure 31 has two edges. The delay of the local data path between the two edges can be calculated using (4.30). The parameters for the first and the second edges (e_i and e_j, respectively) are as follows: For edge e_i, $T_{CD} = 8$ tu, $T_{Set-up} = 1$ tu, $T_{Hold} = 1$ tu, $T_{C \rightarrow Q_0} = 2$ tu, $C_{Int1} = 1$ cu, $C_{Int2} = 3$ cu, $m = 0.5$ ru, and $c = 0.3$ cu, where "cu" denotes "capacitive units," and "ru" denotes "resistive units." For edge e_j, $T_{CD} = 20$ tu, $T_{Set-up} = 3$ tu, $T_{Hold} = 1$ tu, $T_{C \rightarrow Q_0} = 2$ tu, $C_{Int1} = 2$ cu, $C_{Int2} = 1$ cu, $m = 1$ ru, and $c = 0.25$ cu. In Figure 31, the components of the path delay T_{PD} are grouped into the load independent register delay T_{Reg}, the load dependent register and logic delay T_{Load}, the logic delay T_{Logic}, and the clock skew T_{Skew}. The total load independent register delay is

$$T_{Reg_{min}} = T_{C \rightarrow Q_0}(i) - T_{Hold}(j) = 2 - 1 = 1 \ tu,$$

$$T_{Reg_{max}} = T_{C \to Q_0}(i) + T_{Setup}(j) = 2 + 3 = 5\ tu.$$

The total load dependent register and logic delay is the sum of the load dependent register delay component on edge e_i and the load dependent delay component on vertex $e_i.end$. Therefore,

$$\begin{aligned} T_{Load} &= m(i)[c(e_i.end) + C_{Int2}(i)] + m(e_j.start)[C_{Int1}(j) + c(j)] \\ &= 0.5(1 + 3) + 0.5(2 + 0.25) \\ &= 3.125\ tu. \end{aligned}$$

The total load independent logic delay consists of the logic element delay of the vertex between edges which is equal to $T_{Logic} = 4$ tu. Since there are no zero-weight edges along the path from e_i to e_j, T_{Logic} does not have any additional interconnect delay components. The clock skew between the two edges is the difference of the clock delays of these edges and is equal to

$$T_{Skew}(i, j) = T_{CD}(i) - T_{CD}(j) = 8 - 20 \ = \ -12\ tu.$$

Therefore, the path delay between edges i and j is

$$\begin{aligned} T_{PD_{min}} &= T_{Reg_{min}} + T_{Load} + T_{Logic_{min}} + T_{Skew} = 1 + 3.125 + 4 - 12 = -3.875\ tu, \\ T_{PD_{max}} &= T_{Reg_{max}} + T_{Load} + T_{Logic_{max}} + T_{Skew} = 5 + 3.125 + 4 - 12 = 0.125\ tu. \end{aligned}$$

4.6.3. Short Paths

A short path is defined as a path that causes improper circuit operation due to a minimum delay value lower than zero. Zero path delay indicates a marginal race condition. If a path delay between edges e_i and e_j is zero, the final register on edge e_j is clocked at the exact same time the data is latched. If the path delay is less than zero, *i.e.*, $T_{PD_{min}}(i,j) < 0$, the final flip flop is clocked *before* the corresponding data signal arrives and is successfully latched, thereby causing a race condition. Therefore, a short path p_{Short} is defined as

$$T_{PD_{min}}(p_{Short}) \leq 0, \tag{4.31}$$

where $T_{PD_{min}}$ is the minimum delay among all parallel paths between e_i and e_j.

4.6.4. Internal Short and Long Paths

If multiple registers exist on an edge, the delay of the final register may be different than the delay of the internal registers, since the output load may be different. The clock-to-Q delay of the final register on an edge must drive the input capacitance of the vertex loading the last register, C_L. The delay of the internal registers on an edge e, $T_{PD_{Internal}}(e)$, can be calculated as follows:

$$T_{PD_{Internal_{min}}}(e) = T_{C\rightarrow Q_0}(e) - T_{Hold}(e) + m(e)c(e), \tag{4.32}$$

$$T_{PD_{Internal_{max}}}(e) = T_{C\rightarrow Q_0}(e) + T_{Set-up}(e) + m(e)c(e), \tag{4.33}$$

where $T_{Set-up}(e)$, $T_{Hold}(e)$, $T_{C \to Q_0}(e)$, $m(e)$, and $c(e)$ are the REC Level 2 values attached to edge e. Note that the maximum value of this delay is also called the **internal delay of edge** e [23]. The load dependent component $m(e).c(e)$ is not considered if the REC Level 1 Model is used.

Although the likelihood that the delay of an internal path will be greater than the largest register-logic-register path (*i.e.*, a local data path) delay is small, the worst case delay of an internal path is considered in this dissertation for completeness. Furthermore, certain circuits exist, such as a counter or shift register, in which this type of direct register-to-register path is common.

Note that the load for the last register is different on edge e. For the last register, the load is $c(v)$ instead of $c(e)$, where $c(v)$ is the input load of the vertex attached to edge e. If this load is higher than $c(e)$, a maximum value higher than the value calculated in (4.33) is obtained. Therefore, $c(v)$ is used rather than $c(e)$ for the load value. Though unlikely, internal short paths may exist if the hold time of the registers on an edge is extremely high.

Chapter 5. Retiming with RECs

Earlier retiming algorithms have assumed ideal conditions for the nonlogical portion of the data paths, specifically ignoring the temporal characteristics of the registers, the interconnect, and the clock distribution network. Without including these delay components, existing retiming algorithms are not sufficiently accurate for their use in the development of practical high speed circuits. For this reason, clock distribution, variable register, and interconnect delays must be integrated into the retiming process in order to ensure that retiming becomes a practical and useful design methodology.

Both register and interconnect delays are similar in magnitude to the delay of the logic elements. Also, variations in clock delay between widely separated registers may create clock skews which can drastically affect circuit operation. Undesirable clock skew can produce a net negative delay within a local data path. This implies the existence of a race condition, which must be avoided as a condition imposed on the retiming process.

In most retiming algorithms proposed to date, registers are assumed to have zero delay (*e.g.*, [14, 31]) or equal delay (*e.g.*, [18]). In [18], the set-up (t_s) and hold (t_p) times are non-zero constant values, creating an effective clock period of $T_{PD} + t_s + t_p$, where T_{PD} is the worst case path delay of the synchronous circuit. Since constant register delays are assumed throughout the circuit, t_s+t_p is added to each individual local data path, biasing the clock period by this amount. However, this simple summation is not sufficiently accurate since each local data path typically has a different register delay.

Integrating clock skew into the retiming process was first proposed in [22, 43]. The strategy of integrating clock skew and variable register delays into retiming by attaching electrical information describing the register to the edges of the graph representing the synchronous circuit was first introduced in [19]. Following this work, the integration of clock skew into retiming was discussed in [16]. In this dissertation, constraints are placed on the clock skew to permit the use of standard linear programming methods. Variable register and interconnect delays were not considered. In [37], a branch and bound algorithm is briefly introduced to solve the general retiming problem while considering variable non-zero clock skew and register and interconnect delays and is explained in greater detail in [23]. In general, there has been a growing interest in making retiming into a more practical and useful design methodology, evidenced by [14, 16, 18, 19, 22, 23, 29, 31, 33, 37, 43].

The synchronous circuit optimization problem is approached in this dissertation as a two-step process: 1) optimization of the clock distribution network by buffer insertion and clock tree synthesis to meet a specified clock skew schedule [80, 92–94], and 2) optimization of the synchronous circuit via retiming given that the clock scheduling process has previously been performed to satisfy a specific set of clock skew specifications [19, 23, 37, 91].

Optimizing the clock distribution network followed by retiming may create a sub-optimal result. This sub-optimality manifests itself in many practically-applied algorithms, since optimality is sacrificed to prevent excessive CPU times in existing algorithms. Ishii, Leiserson, and Papaefthymiou published research results in simultaneous retiming and clock tuning [31]. Ishii, *et al.* report an $O(V^{11})$ algorithm to perform simultaneous retiming and clock tuning and comment that this problem is far too complicated to be practical unless optimality is sacrificed. They present a sub-optimal $O(V^3(1/e)lg(1/e) + (VE + V^2 lgV)lg(V/e))$ algorithm which calculates the retiming result with $e\%$ accuracy, where e is a user-selected error factor [31]. In this dissertation, the following methodology is assumed: clock skew scheduling followed by retiming.

A retiming algorithm is presented in this chapter which incorporates variable register and interconnect delays and non-zero localized clock skew. Either rising edge or falling edge triggered D flip flops and a single phase clock are assumed throughout the synchronous digital circuit. To accomplish the integration of the variable clock distribution, interconnect, and register delays into the retiming process, a path between logic elements is defined in this dissertation as the traversal from weighted edge to weighted edge, an edge being interpreted as a connection between logic elements containing zero, one, or more registers. With this definition, clock, register, and interconnect delays are assigned to each edge. Thus, as registers are shifted from edge to edge, different clock skews and register delays are considered in each of the local path delays. This process permits both maximum clock periods and race conditions to be detected on a path-by-path basis. Estimates of register delays on zero weight edges (*i.e.*, interconnections between logic elements that contain no registers) derived from the circuit layout are required in order to include the effects of variable register delays on the retimed circuit. This approach, therefore, initially requires approximate (or estimated) values of the register, clock distribution, and interconnect delays which can be replaced with more accurate values as the exploratory retiming process becomes better specified [19, 23, 37].

The retiming algorithm *RETSAM* presented in this chapter uses a branch and bound approach. Although *RETSAM* determines a retimed circuit that will operate at its maximum clock frequency, enhanced computational efficiency can be obtained by placing certain conditions related to the monotonicity of the path delays on the REC values. These monotonicity conditions permit the use of standard linear programming methods during the retiming process. These conditions and the feasibility of their application to practical circuits are presented in this chapter.

This chapter is organized as follows. The steps taken to incorporate the REC's into the retiming process are introduced in Section 5.1. In Section 5.2, the Sequential Adjacency Matrix (SAM), which includes the edge-to-edge delays of a graph, is introduced. Timing constraints, derived from the SAM, are described in Section 5.3. The proposed retiming algorithm *RETSAM* is presented in Section 5.4. Monotonicity restrictions that may be placed on the RECs to permit the use of standard linear programming methods to perform retiming with the additional electrical delay information are described in

Section 5.5. Results of applying the proposed algorithm *RETSAM* to MCNC benchmark circuits are presented in Section 5.6 and some conclusions are drawn in Section 5.7.

5.1. Incorporating REC's into the Retiming Process

In order to consider the effects of clock distribution, variable register, and interconnect delays, REC's are assigned to each edge of the graph in the following form: $T_{CD} : T_{Set-up}/T_{Hold}/T_{C-Q_0} — C_{Int1}/C_{Int2}$ $[m, c]$. By attaching delay components to registers located on edges, the local path must be defined from edge-to-edge [19, 23] rather than vertex-to-vertex, as in existing retiming algorithms [14]. The original digital correlator introduced in [14] is depicted in Figure 32. A modified version of this graph in which an REC is assigned to each edge is shown in Figure 33. By assigning a clock delay to each edge, the circuit is assumed to be partitioned into regions of similar clock delay, *i.e.*, registers that are located on the same edge are physically located within the same clock delay region. Therefore, registers that end up on the same edge after retiming are assumed to have similar clock delay. Registers that move to different edges are assumed to have the clock and register delays of the new edge. Since registers on different edges may be considered to have different clock and register related delays, moving a register from one edge to another edge during retiming will not only create different local data paths with different logic, register, and interconnect delays, but may also change the localized clock skew of the new local data paths.

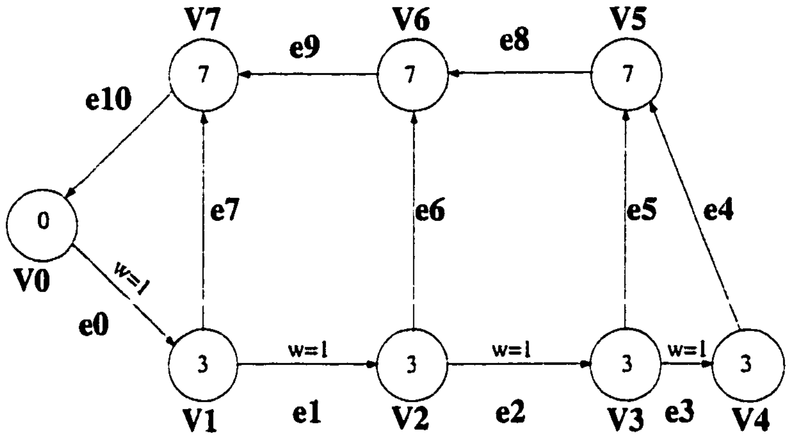


Figure 32. Graph of the digital correlator in [14]. The edges that are not labeled are assumed to have zero weight.

The clock-to-Q delay $T_{C-Q_0}(e_i)$ is edge dependent since each edge is connected to a different vertex, thereby changing the capacitive loading on the registers located on each edge. This variation occurs since each vertex represents a variety of possible

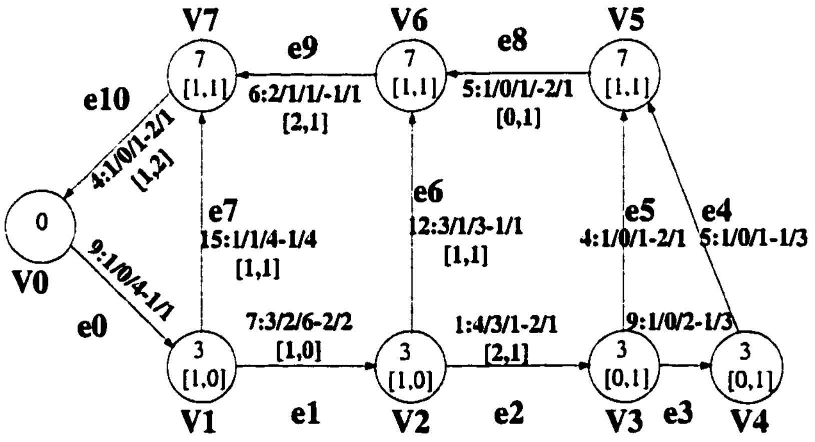


Figure 33. Graph of the digital correlator [14] with added REC values. The edges that do not have $[m, c]$ values assigned are assumed to have values $m = 0$ and $c = 0$.

logic elements, placing a different output load on the register driving the vertex. Hence, the same registers driving different vertex inputs will have different $T_{C-Q_0}(e_i)$ delays. Furthermore, a variety of register types selected prior to the retiming process may be used at different locations within the circuit, due to the specific speed, power, and area tradeoffs peculiar to that portion of the circuit. T_{Set-up} and T_{Hold} may also change for different register cell instances. Thus, $T_{Set-up}(e_i)$ and $T_{Hold}(e_i)$ can vary per edge. Therefore, selecting a specific register to satisfy a set of performance-based design requirements will change the value of T_{Set-up} and T_{Hold} for each edge. A similar discussion is valid for $T_{C-Q_0}(e_i)$. The varying loading exacerbates this delay variation, thereby requiring that variable register delays be considered during the retiming process.

As defined in Section (4.6), the local data path delay $T_{PD}(i,j)$ from edges e_i to e_j is

$$\begin{aligned} T_{PD_{min}}(i,j) &= T_{C-Q_0}(i) + m(i)[c(i) + C_{Int2}(i)] + T_{Logic_{min}}(i,j) \\ &+m(j)[C_{Int1}(j) + c(j)] - T_{Hold}(j) + T_{Skew}(i,j), \end{aligned}$$

$$\tag{5.1}$$

$$\begin{aligned} T_{PD_{max}}(i,j) &= T_{C-Q_0}(i) + m(i)[c(i) + C_{Int2}(i)] + T_{Logic_{max}}(i,j) \\ &+m(j)[C_{Int1}(j) + c(j)] + T_{Set-up}(j) + T_{Skew}(i,j), \end{aligned}$$

where $T_{Logic}(i,j)$ is the delay of the logic elements between e_i and e_j, including the interconnect delay of the zero weight edges along the path between these edges. If parallel paths exist, minimum and maximum local data path delays, $T_{Logic_{min}}$ and $T_{Logic_{max}}$, are defined. If $T_{PD_{min}}(i,j) < 0$, a race condition between e_i and e_j exists since in this local data path the final register is clocked before the data signal arrives and is successfully latched.

If registers R_i and R_j are located on the same edge e_k and are sequentially adjacent, then, according to the definition of the RECs, the clock skew between R_i and R_j is zero from (2.2) since the clock delays of both registers are the same. This assumption is made since registers on the same edge would typically be physically close, and therefore the difference in clock delay to each register and the interconnect delay between these registers would be negligible. Furthermore, since no vertices (logic elements) exist between registers R_i and R_j, when both are on the same edge, the logic delay between the two registers is zero. Since all registers located on the same edge are defined to have the same timing characteristics (REC values), all sequentially adjacent registers located on the same edge have a similar internal path delay. A path composed of multiple registers on an edge could possibly be the critical worst case path of the overall circuit and its delay is defined as $T_{PD_{Internal}}(e_k)$, given by (4.32) and (4.33) as follows:

$$T_{PD_{Internal_{min}}}(e_k) = T_{C \rightarrow Q_0}(e_k) - T_{Hold}(e_k) + m(e_k)c(e_k), \qquad (5.2)$$

$$T_{PD_{Internal_{max}}}(e_k) = T_{C \rightarrow Q_0}(e_k) + T_{Setup}(e_k) + m(e_k)c(e_k). \qquad (5.3)$$

The maximum internal path delay is used to provide a *safety test* to ensure that registers on any edge create long paths due to excessive register delays. In case of excessively high hold time values for internal registers, minimum internal path delay may be considered to ensure that internal race conditions do not exist as explained in Section 4.6.4.

5.2. Sequential Adjacency Matrix (SAM)

A W matrix, defined in [14], contains all vertex-to-vertex path weights. The elements of this matrix, $W(i,j)$, can be calculated as

$$W(i,j) = \min \{w(p) : \ p : v_i \rightsquigarrow v_j\}. \tag{5.4}$$

This matrix can be calculated using an all-pairs shortest path algorithm, such as the Floyd-Warshall algorithm [47, 49, 96]. Also, a W_r matrix is defined in this dissertation as the W matrix after the retiming process has been applied to the circuit.

The Sequential Adjacency Matrix (the SAM or the S matrix) is an $|E| \times |E|$ matrix whose element $S(i,j)$ is the path delay from e_i to e_j. The S matrix element, $S(i,j)$, is calculated from

$$S(i,j) = \max \{T_{PD}(i,j) : \ p : e_i \rightsquigarrow e_j \ \wedge \ w(p) = W(i,j)\}. \tag{5.5}$$

If parallel paths exist between any two edges, the S matrix is composed of two matrices, S_{min} and S_{max}. Equations (5.6) and (5.7) are used to calculate the values of these two matrices. In order to reduce the number of matrices, a combined matrix, S', is used. $S'(i,j)$ contains $S_{min}(i,j)$ if $S_{min}(i,j)$ contains a zero or negative entry, and contains $S_{max}(i,j)$ if no zero or negative entry exists. The importance of $S_{min}(i,j)$ is determined by whether a zero or negative entry exists, thereby denoting a race condition. If $S_{min}(i,j)$ is completely positive, the maximum valued entries in $S_{max}(i,j)$ limit the maximum speed of the circuit. Equation (5.8) is used to calculate the combined matrix, S'.

$$S_{min}(i,j) = \min\left\{T_{PD_{min}}(i,j) : \; p : e_i \rightsquigarrow e_j \;\wedge\; w(p) = W(i,j)\right\}, \tag{5.6}$$

$$S_{max}(i,j) = \max\left\{T_{PD_{max}}(i,j) : \; p : e_i \rightsquigarrow e_j \;\wedge\; w(p) = W(i,j)\right\}, \tag{5.7}$$

$$S'(i,j) = \begin{cases} S_{min}(i,j), & if\ S_{min}(i,j) \leq 0 \\ S_{max}(i,j), & if\ S_{min}(i,j) > 0. \end{cases} \tag{5.8}$$

Note that the S' matrix contains information for only those paths that can potentially cause the circuit to function improperly. Therefore, the zero and negative entries in the S_{min} matrix override the positive entries in the corresponding S_{max} matrix during the calculation of the S' matrix. This choice occurs since negative entries flag race conditions and zero entries flag marginal race conditions which are not permitted to exist in the retimed circuit. To maintain a sufficient margin within the circuit, entries below a specific process dependent parameter k are not permitted. Paths with delays less than or equal to k tu may create race conditions due to statistical process variations within the integrated circuit and are therefore not permitted.

For the remainder of this dissertation, the notation for the combined matrix S' is denoted as S for simplicity. The S matrix of the graph of Figure 33 is shown in Table 3. The light shaded elements of the table indicate those paths with race conditions (negative values) and the dark shaded elements indicate those paths with a path delay greater than the desired clock period. In this example, a target clock period of 37 tu is assumed. Paths with zero delay are marginal race conditions that are not permitted and would appear as light shaded. The unshaded elements of the table indicate those paths that neither limit the maximum performance of the circuit nor create race conditions.

5.3. Timing Constraints

A branch and bound algorithm is presented in this chapter in which unbounded

Table 3: The SAM for the graph of Figure 33. Light shaded entries represent short paths, whereas dark shaded entries represent long paths for $c = 37$ tu. Unshaded entries denote permissible paths.

SAM		to										
		e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10
from	e0	24	14	29	22	29	27	16	4	36	31	30
	e1	33	42	24	17	24	22	11	32	31	26	39
	e2	32	41	56	0	8	6	43	31	15	25	38
	e3	37	46	61	54	10	58	48	36	20	30	43
	e4	29	38	53	46	55	51	40	28	12	22	35
	e5	28	37	52	45	52	50	[illegible]	27	11	21	34
	e6	29	38	55	46	55	51	40	28	60	22	35
	e7	26	35	50	43	50	47	57	25	57	52	32
	e8	18	27	62	35	62	40	29	17	48	11	24
	e9	13	22	57	30	37	35	24	12	60	50	19
	e10	-3	7	22	15	22	20	9	-5	29	24	23

values are initially assumed for the lag ranges. These lag ranges are tightened using timing constraints derived from the SAM. There are four different types of timing constraints: negative edge weight, long path, short path, and internal path. These different types of constraints are explained in greater detail in the following subsections.

5.3.1. Negative edge weight constraints

As introduced in [14], a properly retimed graph contains no negative edge weights. Negative edge weights are permitted for peripheral edges in [46] in order to shift the registers to the periphery of a synchronous circuit. This approach permits combinatorial optimization to be performed on the circuitry placed between the peripheral edges. However, since the retiming algorithm described in this chapter does not exploit this feature of **resynthesis**, negative edge weights are disallowed. As shown in Section 2.4.1, using (3.5), the negative edge weight constraint can be written as

$$w_r(e) \geq 0, \; \forall\, e \in E \,. \tag{5.9}$$

5.3.2. Long path constraints

If a clock period c is desired, then all paths with a delay greater than c must be eliminated. Long paths are represented by entries in the S matrix that exceed a desired clock period c. In Table 3, long paths for $c = 37$ tu are depicted using dark shaded elements. In order to eliminate these long paths, the two edges that create the long path are made nonsequentially adjacent.

Two registers are sequentially adjacent if there exists a zero weight path between the two registers. According to this definition, in order to make two edges, e_i and e_j, nonsequentially adjacent, three approaches are possible: 1) the source or 2) the destination edges can be made zero weight, *i.e.*, all registers can be removed from these edges, or 3) one or more registers can be placed within each zero weight path between the source and destination registers. The first two conditions exist since by eliminating the initial and/or final register of a local data path, a longer path is created which may have a smaller delay (due to negative clock skew). Using the definitions for w_r and W_r described in Section 5.2, these three conditions can be written in terms of path and edge weights as follows:

$$w_r(e_i) = 0, \tag{5.10}$$

$$w_r(e_j) = 0, \tag{5.11}$$

$$W_r(e_i.end, e_j.start) > 0. \tag{5.12}$$

If (5.10) or (5.11) is satisfied, then no registers exist on edge i or j, respectively, and therefore all local data paths between edges i and j are eliminated. If (5.12) is satisfied, all possible paths between edges i and j have a weight of at least one. This violates the definition of sequential adjacency, *i.e.*, no paths exist with a zero weight between these two edges. Intuitively, it is stated in (5.10), (5.11), and (5.12) that either the initial or the final edge does not have any register located on it or there is at least one register along every path between these two edges.

5.3.3. Short path constraints

Short paths appear as zero or negative entries in the S matrix. $S(i,j) \leq 0$ indicates a short path originating at e_i and terminating at e_j. If e_i and e_j form a short path, then the initial and final registers of this path must be made nonsequentially adjacent. Equations (5.10), (5.11), and (5.12) are used to eliminate any catastrophic short paths (or race conditions).

5.3.4. Internal path constraints

Internal long paths are created between two sequentially adjacent registers on the same edge when the edge weight is greater than one and the internal path delay is greater than a specified clock period c. Internal long path constraints can be formulated using (5.3) as

$$w_r(e_i) \leq 1, \quad \forall\, i : T_{PD_{Internal_{max}}}(e_i) > c, \tag{5.13}$$

which suggests that if the internal path delay of an edge is greater than the desired clock period, the weight of that specific edge must be either zero or one to prevent internal long paths.

An edge with a weight of three is depicted in Figure 34. More precisely, edge e_3 of Figure 32 which connects vertices v_3 and v_4 is shown under the assumption, $w(e_3) = 3$. In Figure 34, the internal path delay on edge e_3 (the delay between register pairs r_1, r_2 and r_2, r_3) is constant since multiple registers on the same edge are assumed to have the same delay, and the interconnect delay between internal registers is assumed to be negligible. Since the electrical characteristics of the vertices and registers do not change, the maximum internal path delays are calculated from (5.3) only once before the retiming process is applied. This calculation ensures that before the retiming process begins, any unnecessary internal long paths due to excessive internal path delays are not created. An example graph in which the internal path delay of edge e_2 exceeds the path delay between edges e_2 and e_3, thereby causing an internal long path, is shown in Figure 35. It can be observed from Figure 35 that the internal long path delay exceeds the path delay between e_2 and e_3 since the negative clock skew between these registers decreases the path delay, lowering $T_{PD}(e_2, e_3)$ below the internal path delay.

Internal short paths constraints are similar to the internal long path constraints. For those internal paths with negative internal path delay due to (5.2), the following

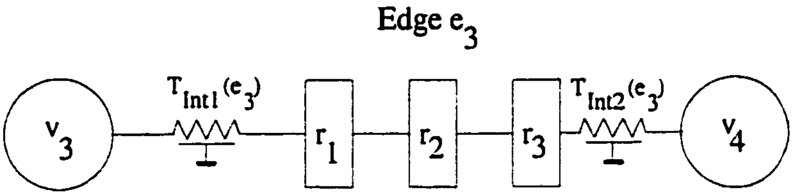
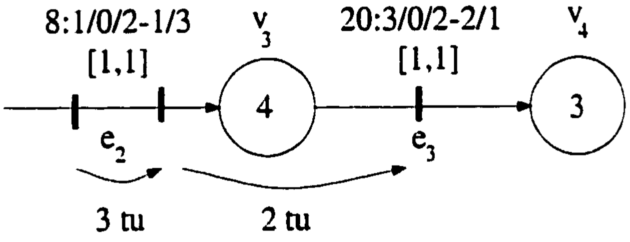


Figure 34. The internal path delay between registers located on the same edge, $T_{PD_{Internal_{max}}}$, is equal due to the definition of the RECs. This example demonstrates the case where $w(e_3) = 3$.



$$T_{PD_{Internal_{max}}}(e_2) > T_{PD_{max}}(e_2, e_3)$$

Figure 35. An example graph in which the internal path delay on edge e_2 exceeds the path delay between e_2 and e_3. This graph exemplifies the importance of considering internal long paths before the retiming process is initiated.

constraint is applied:

$$w_r(e_i) \leq 1, \ \forall i : T_{PD_{Internal_{min}}} \leq 0, \tag{5.14}$$

which suggests that if the internal path delay of an edge is less than zero denoting a race condition, the weight of that specific edge must be either zero or one to prevent internal short paths.

5.3.5. Constraints due to vertex lags

Constraints (2.3), (5.10), (5.11), (5.12), (5.13), and (5.14) are written in terms of edge weights. These constraints can be rewritten as (5.15), (5.16), (5.17), (5.18), (5.19), and (5.20), respectively, to reduce the number of necessary operations.

$$r(e.start) - r(e.end) \leq w(e), \ \forall e \in E, \tag{5.15}$$

$$r(e_i.start) - r(e_i.end) = w(e_i), \tag{5.16}$$

$$r(e_j.start) - r(e_j.end) = w(e_j), \tag{5.17}$$

$$r(e_i.end) - r(e_j.start) \leq W(i,j) - 1, \tag{5.18}$$

$$r(e_i.start) - r(e_i.end) \geq w(e_i) - 1, \ \forall i : T_{PD_{Internal_{max}}}(e_i) > c, \tag{5.19}$$

$$r(e_i.start) - r(e_i.end) \geq w(e_i) - 1, \forall i : T_{PD_{Internal_{min}}}(e_i) \leq 0. \tag{5.20}$$

In order to provide some intuition to (5.15), (5.16), (5.17), (5.18), (5.19), and (5.20), note that, given two vertices u and v, the value $r(u) - r(v)$ can be thought of as "the number of registers taken out of the path $p : u \rightsquigarrow v$." Given this interpretation, it is implied in (5.15) that "the number of registers taken from an edge e cannot be greater than the original weight of the edge," *i.e.*, none of the edge weights can be negative. In a similar manner, it is stated in (5.16) and (5.17) that "the number of registers taken from edge e_i and e_j, respectively, must be equal to the original weight of this edge," implicitly stating that this edge should be made zero weight. In (5.18) it is stated that "the registers taken from the path $p : e_i \rightsquigarrow e_j$ must be less than the original weight of this path minus one," implicitly stating that at least one register should be left along any path between registers e_i and e_j, thereby making this path nonsequentially adjacent. Finally, in (5.19) and (5.20) it is implied that either zero or one register should be left on an edge e that contains an internal long or short path.

5.4. Retiming Algorithm

In this section three algorithms are introduced: 1) Algorithm *RETSAM* to perform retiming of synchronous circuits, 2) Algorithm *CHECKCP* to check the feasibility of a specific clock period, and 3) Algorithm *SOLVELAGS* to determine the vertex lags based on a branch and bound method. These three algorithms are explained in the following subsections.

5.4.1. *RETSAM*: Retiming Algorithm for Synchronous Circuits with Attached Electrical Information

Retiming a synchronous circuit is achieved by performing a binary search of all possible clock periods on a specific circuit graph. The pseudo-code of the retiming algorithm is shown in Figure 36. The lower and upper bounds of the binary search are CP_{min} and CP_{max}, respectively. Initially the lower bound is zero (Step 1). If the original graph does not contain any race conditions, the critical path delay of the original graph defines the upper bound of the binary search (Step 2). The SAM is calculated in Step 3 and used throughout the algorithm. If the original graph contains one or more race conditions, the maximum value in the SAM is used as the upper bound (Step 4). During the binary search, a specific clock period, CP_{target}, is checked for feasibility using algorithm *CHECKCP* (Steps 5 and 6). Depending on whether a solution exists (Step 7) or not (Step 8), the lower and upper search bounds are adjusted and the binary search continues until the minimum clock period is determined (Step 9). An approximate solution can be obtained for the minimum clock period if *RETSAM* is terminated once the binary search bounds become sufficiently tight. This step may significantly reduce the run time requirement of the algorithm, since any target clock periods close to the minimum clock period may require excessive computational time.

```
1. $CP_{min} = 0$
2. $CP_{max}$ = clock period of the original graph
3. Calculate SAM
4. If the original graph has race conditions $CP_{max} = \max\{S(i,j),\ \forall i,j\}$,
5. Choose $CP_{target} = \lfloor \frac{CP_{max}+CP_{min}}{2} \rfloor$,
6. Check for feasibility of $c = CP_{target}$ using algorithm CHECKCP
7. If set of inequalities can be successfully solved, then $CP_{max} = CP_{target}$
8. If not, then $CP_{min} = CP_{target}$
9. Continue this process until $CP_{min} = CP_{max}$
```

Figure 36. Pseudo-code for *RETSAM*

5.4.2. *CHECKCP*: Clock Period Feasibility Check

A feasibility check for a specific clock period CP_{target} is achieved by solving the set of nonlinear inequalities for the vertex lag ranges. If all the constraints are satisfied for every path in the graph, the clock period is considered feasible. Pseudo-code for the algorithm that determines the feasibility of a clock period is shown in Figure 37. Lag ranges are stored in an array called $r[]$. The timing constraints are derived from the SAM.

The most important step in *CHECKCP* is solving for the vertex lags $r()$ using Algorithm *SOLVELAGS*. The objective of the retiming algorithm is to yield a set of vertex lags that satisfy (5.15) through (5.19). To achieve this objective, the vertex lag ranges are initialized with unbounded values $[-\infty \ldots \infty]$. Timing constraints are continuously applied to these vertex lags in order to tighten the ranges until eventually all the constraints are satisfied. Once the vertex lags are each defined, these lag values are used to determine the edge weights of the retimed graph according to (3.5).

5.4.3. *SOLVELAGS*: Determination of the Vertex Lags Using a Branch and Bound Approach

The following types of equalities and inequalities are created from the aforemen-

```
1. $r[0] = [0 \ldots 0]$
2. $r[k] = [-\infty \ldots +\infty], k = 1, \ldots, E - 1$
3. Create a constraint list $L$ for clock period $c$
4. Adjust lags to satisfy all constraints using algorithm SOLVELAGS
5. If all lags are fixed and all constraints are unsatisfied
       → Clock period is not feasible
6. If all constraints are satisfied
       → Clock period is feasible
7. If $c$ is feasible and all lags are not fixed
       → Use lower bounds of the unfixed lags
```

Figure 37. Pseudo-code for clock period feasibility test, CHECKCP

tioned timing constraints:

$$r(v_a) - r(v_b) = k, \tag{5.21}$$

$$r(v_a) - r(v_b) \leq k, \tag{5.22}$$

$$r(v_a) - r(v_b) = k_1 \;\; or \;\; r(v_c) - r(v_d) = k_2, \tag{5.23}$$

$$r(v_a) - r(v_b) = k_1 \; or \; r(v_c) - r(v_d) \leq k_2, \tag{5.24}$$

$$r(v_a) - r(v_b) = k_1 \;\; or \;\; r(v_c) - r(v_d) = k_2 \;\; or \;\; r(v_e) - r(v_f) \leq k_3, \tag{5.25}$$

where $r()$ are vertex lags and k_n are constants. The *or* statements that appear in (5.23), (5.24), and (5.25) prohibit the use of standard linear programming methods [47] and necessitate the use of branch and bound techniques for the general unconstrained retiming problem. However, it is shown in Section 5.5 that a polynomial-time sub-optimal solution is feasible when the path delays are constrained to monotonically increasing delay values.

Note the existence of *multiple choices* in each inequality in (5.23), (5.24), and (5.25) in the form of $s1$ *or* $s2$ *or* $s3$, where $s1$, $s2$, and $s3$ are different choices. $s3 = nil$ (non-existant) implies the form shown in (5.23) and (5.24), whereas $s3 = nil$ and

$s2 = nil$ imply the form shown in (5.21) and (5.22). Notationally, $s1$ must always exist, and $s2 = nil$ and $s3 \neq nil$ is not permitted. Therefore, (5.25) is characterized by $s3 \neq nil$. Note that it is possible to *reduce* the complexity of a multiple choice inequality by either eliminating $s2$ or $s3$. Thus, an inequality originally in the form of (5.25) can be converted to the form of (5.23), thereby reducing its complexity.

To gain insight into how these multiple choice inequalities are created, consider retiming the graph of Figure 33, for which the SAM is shown in Table 3. To achieve a clock period of $c = 37$ tu, the dark shaded and light shaded paths must be avoided, since they represent long paths for $c = 37$ tu and short paths, respectively. To avoid, for example, the path $p : e_3 \rightsquigarrow e_0$, there exists three possible choices, derived from (5.10), (5.11), and (5.12), resulting in the multiple choice inequality,

$$r(3) - r(4) = 1 \ or \ r(0) - r(1) = 1 \ or \ r(4) - r(0) \leq -1, \tag{5.26}$$

which states that to eliminate the path starting at e_3 and terminating at e_0, either e_3 or e_0 must be zero weight, thereby making the path $p : e_3 \rightsquigarrow e_0$ non-existent, or at least one register must be placed between the initial and terminating vertices of the path p.

The pseudo-code of the branch and bound algorithm *SOLVELAGS* that calculates the vertex lags is shown in Figure 38. A list L is maintained to store the timing constraints derived from the SAM (Step 1) which are individually eliminated until no more constraints remain unevaluated (Steps 2 and 18). Within this loop, each constraint

l is evaluated separately (Step 3) to determine if $s3$ (Step 4) or $s2$ (Step 12) is non-existant.

If $s3$ can be eliminated (*i.e.*, $s3$ can never be satisfied using the current vertex lag list), the complexity of the constraint l can be reduced (Steps 5 through 7). Alternatively, if $s3$ is satisfied using the current vertex list without further tightening the boundaries of the vertex lag list, the constraint l can be eliminated (Steps 8 through 11). The same operations are performed on condition $s2$ between Steps 12 and 13.

After $s3$ and $s2$ are evaluated, the lags are adjusted to satisfy constraint $s1$ (Step 14). If $s1$ cannot be satisfied, a vertex lag set does not exist that satisfies all constraints (Steps 15 and 16), since $s1$ is the last possible solution in the *or* chain. If a vertex lag set can be found that satisfies $s1$, the vertex lag that satisfies $s1$ is used (Step 17). After L is completely evaluated and the entire list of constraints is satisfied, a solution exists and the current status of the vertex list is the set of final vertex lags (Step 19).

The solution method for determining the vertex lags of the graph shown in Figure 33 is exemplified in Table 4. The target minimum clock period in this example is 24 tu. To solve for a set of vertex lags that provides a proper retiming, an inequality similar to (5.26) is written for each short or long path shown in Table 3. In this algorithm, the unbounded value $[-\infty \ldots \infty]$ is initially assigned to each vertex lag range. Only one vertex lag [$r(0)$ for simplicity] is initialized to 0 and the other lags are calculated relative to $r(0)$. Bounds of the vertex lag ranges are continuously tightened to determine

```
1. Let $L$ be the constraint list.
2. while $L \neq \emptyset$
3.   Let $l = (s1 \; or \; s2 \; or \; s3) \in L$ be the next timing constraint, where $s1$, $s2$, or $s3$
     is one of the timing constraints. $s2$ and/or $s3$ may be $nil$ (non-satisfiable)
4.   if $s3 = nil$ goto 12.
5.   Constrain vertex lags to satisfy $s3$. Iterate if necessary.
6.   if $s3$ is non-satisfiable with the current vertex lag list
7.     $l = l - s3$ (delete condition $s3$).
8.   else
9.     $L = L - l$ (delete the entire constraint)
10.    goto 18.
11.  endif.
12.  if $s2 = nil$ goto 14.
13.  repeat steps 5 through 11 for $s2$.
14.  Constrain vertex lags to satisfy $s1$. Iterate if necessary.
15.  if $s1$ is non-satisfiable using the current vertex lag list
16.    exit. no solution exists.
17.  else $L = L - l$ (delete the entire constraint).
18. endwhile.
19. Use the vertex list as the solution
```

Figure 38. Pseudo-code for the branch and bound algorithm *SOLVELAGS* that calculates the vertex lags.

Table 4: Example solution for $c = 24$. A single value is shown for equal lower and upper bounds.

Constraint	Type	r(0)	r(1)	r(2)	r(3)	r(4)	r(5)	r(6)	r(7)
		0	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$
$r(0)-r(1) \le 1$	Negativity on e_0	0	$-1..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$
$r(1)-r(7) \le 0$	Negativity on e_7	0	$-1..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-\infty..\infty$	$-1..\infty$
Negativity on e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9		0	$-1..0$	$-2..0$	$-3..1$	$-4..0$	$-3..0$	$-2..0$	$-1..0$
$r(0)-r(1)=1$ or $r(1)-r(0) \le -1$	Long path: $e_0 \rightarrow e_0$	0	-1	$-2..0$	$-3..1$	$-4..0$	$-3..0$	$-2..0$	$-1..0$
Negativity on all edges		0	-1	$-2..0$	$-3..1$	$-4..0$	$-3..0$	$-2..0$	$-1..0$
$r(1)-r(7)=0$ or $r(7)-r(1) \le 0$	Long path: $e_7 \rightarrow e_7$	0	-1	$-2..0$	$-3..1$	$-4..0$	$-3..0$	$-2..0$	-1
$r(1)-r(2) \le 1$	Negativity on e_1	0	-1	$-2..0$	$-3..1$	$-4..0$	$-3..0$	$-2..0$	-1
$r(6)-r(7) \le 0$	Negativity on e_9	0	-1	$-2..0$	$-3..1$	$-4..0$	$-3..0$	$-2..-1$	-1
$r(5)-r(6) \le 0$	Negativity on e_8	0	-1	$-2..0$	$-3..1$	$-4..0$	$-3..-1$	$-2..-1$	-1
$r(2)-r(6) \le 0$	Negativity on e_6	0	-1	$-2..-1$	$-3..1$	$-4..0$	$-3..-1$	$-2..-1$	-1
⋮									
Iteration #1: choose r(5)=-2		0	-1	-2	$-3..-2$	$-4..-2$	-2	-2	-1
Iteration #1: choose r(3)=-3		0	-1	-2	-3	$-4..-2$	-2	-2	-1
$r(2)-r(3)=1$ or $r(3)-r(4)=1$	Short path: $e_2 \rightarrow e_3$	Condition 1 satisfied							
$r(3)-r(4) \le 1$	Negativity on e_3	0	-1	-2	-3	-2	-2	-2	-1
$r(3)-r(4)=1$ or $r(6)-r(7)=0$ or $r(4)-r(6) \le -1$	Long path: $e_3 \rightarrow e_9$	Constraints are not satisfiable							
Iteration #2: choose r(5)=-3		0	-1	-2	$-3..-2$	$-4..-2$	-3	-2	-1
⋮									

a set of vertex lag ranges that satisfies all of the constraints.

In Table 4, the vertex lag non-negativity constraint from (5.15) is applied to each vertex to further tighten the vertex lag ranges (shown in the first three rows). When the constraint from (5.15) cannot be used to further tighten the vertex lag ranges, the long path constraints from (5.16), (5.17), and (5.18) are used (see row 4). Short paths are also eliminated using (5.16), (5.17), and (5.18). Each time the bounds are tightened by applying long (or short) path constraints, (5.15) is applied to the new set and the neighboring vertex lag ranges to ensure non-negative edge weights on each edge. The algorithm may reach a point where the application of the constraints can no longer tighten the bounds (the dark shaded row). Once this occurs, all possible values for each vertex lag are tested. On the first dark shaded row in Table 4, there are two unfixed lags with cardinalities two and three, respectively. Therefore, $2 * 3 = 6$ possible solutions exist and must be evaluated. If a solution is reached, the algorithm is terminated and the resulting vertex lag ranges are used to determine the edge weights of the retimed graph. If all possible solutions are considered and a set of vertex lag ranges cannot be determined that satisfy all constraints, a solution for that specific clock period does not exist.

5.5. Path Delay Monotonicity Constraints

The algorithm *RETSAM* is capable of including arbitrary register properties such as clock delays. In the event, however, that the design freedom permits consideration of the REC delay values as part of the design process, certain constraints can be placed on the REC values which will permit the use of standard linear programming techniques, such as the Bellman-Ford method [47], for solving for the vertex lags. This process yields more computationally efficient results. In the following subsections, the path delay monotonicity constraints that must be applied to the REC values to permit computationally efficient retiming are introduced, design issues relating to the clock distribution network with respect to satisfying the monotonicity constraints are described, and the feasibility of applying these monotonicity constraints to practical circuits is discussed.

5.5.1. Intuition for the Monotonicity Constraints

In practical integrated circuits, variations in clock delay between widely separated registers may create clock skews which can drastically affect circuit operation. An observation of (4.27) is that arbitrary clock skews (in particular, negative clock skews) may cause longer data paths (paths with more edges and vertices) to have less delay than shorter data paths (paths with less edges and vertices). Therefore, unless constraints are placed on the possible clock delays, the data path delays are arbitrary and can quite possibly be negative. This decrease in path delay can occur either due to negative clock skew or to the delay components of the newly placed register being less than the delay components of the original register. Thus, a sub-path p_1 of a longer path p (composed of added edges and vertices) may have a delay greater than path p. An example graph in which this occurs is depicted in Figure 39. In this graph, the sub-path p_1 has a greater delay than path p. The primary cause is due to the effect of negative clock skew which effectively subtracts delay from the local data path p, thereby causing the sub-path p_1 to have greater delay (or the longer path p to have less delay). In the

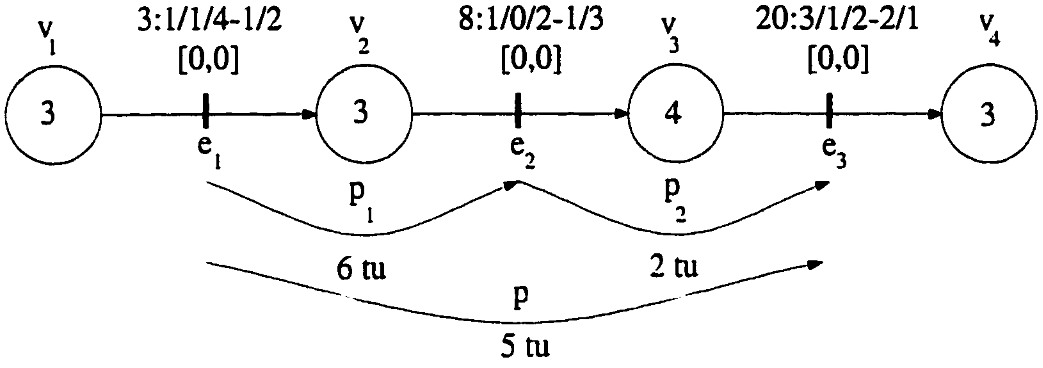


Figure 39. An example graph in which the path delays do not monotonically increase. The sub-path p_1 has a delay greater than its original path p. The cause of this non-monotonic behavior is due to the negative clock skew between edges e_1 and e_2. Note that only the maximum path delay, $T_{PD_{max}}$, is considered and the load-dependence of the delays is assumed to be negligible for simplicity.

specific example shown in Figure 39, sub-path p_1 is 1 tu greater than path p. This 1 tu difference results from the negative clock skew between edges e_1 and e_2, *i.e.*, $T_{Skew}(e_1, e_2) = 3 - 8 = -5$ tu.

When a sub-path of a larger path has a greater delay, there are three choices for removing that path. These choices are:

1. Place a register between the initial and the terminal edges, since a shorter path may have a smaller delay (or a short path may have a larger delay).
2. Remove the initial edge so that the path becomes longer (more edges and vertices). This longer path may have a smaller delay.
3. Remove the terminal edge so that the path becomes longer. This longer path may have a smaller delay.

Conditions 2 and 3 are required since the data path delays are completely arbitrary and any of these conditions may possibly remove the undesirable path. Since these three conditions are used in *RETSAM*, standard linear programming techniques are not possible due to the boolean *or* operation, thereby resulting in the multiple-choice inequalities represented by (5.21)-(5.25).

A strategy to improve the time efficiency of the retiming algorithm is as follows: If certain temporal constraints are placed on the clock delays, it is possible to guarantee that a sub-path p_1 of a larger path p will always have a larger delay, thereby removing the aforementioned conditions 2 and 3. This simplification permits using the standard Bellman-Ford method, since by removing conditions, 2 and 3, the remaining inequalities are linear in the form of $x_i - x_j \leq a_{ij}$, since no boolean *or* operation is being performed.

Assume a path with three edges e_i, e_j and e_k, and two vertices v_a and v_b between these edges. The following condition ensures monotonically increasing path delays:

$$\begin{gathered} T_{Skew}(j,i) \leq \min\left\{\mathrm{T}_1(i,j), \tau_1(i,j)\right\}, \\ T_{Skew}(k,j) \leq \min\left\{\mathrm{T}_2(j,k), \tau_2(j,k)\right\}, \\ T_{Skew}(i,j) \leq \mathrm{T}(i,j). \end{gathered} \tag{5.27}$$

where T_1, τ_1, T_2, τ_2, and T are constant values calculated using the REC parameters attached to edges i, j, and k, and vertices v_a and v_b between these edges. Details of the derivation of (5.27) can be found later in Section 5.5.4.

Equation (5.27) guarantees monotonically increasing delays for each edge-to-edge local data path. If (5.27) is satisfied and all path delays increase monotonically, standard linear programming methods can be applied when retiming a graph, thereby dramatically improving the computational efficiency of the retiming process. Relationship (5.27) does not permit race conditions since race conditions may create sub-paths with delays larger than the original paths. Therefore, race conditions must be eliminated in advance to permit the use of the Bellman-Ford method. Therefore this strategy does not verify the existence of race conditions but instead assumes that all race conditions have been eliminated *a priori*. Given that (5.27) is satisfied for each path in the synchronous circuit, inequalities for longer paths can be written and solved using the Bellman-Ford method. Equations (5.28) and (5.29) must be satisfied to ensure that a proper retiming with RECs has been accomplished. These conditions are similar to those derived in [14].

$$r(u) - r(v) \leq w(e), \forall e : u \rightarrow v, \tag{5.28}$$

$$r(u) - r(v) \leq W(u, v) - 1, \forall i, j \in E : S(i, j) > c, u = e_i.end, v = e_j.start. \tag{5.29}$$

5.5.2. Designing the Clock Distribution Network

In a practical integrated circuit, clock delays to each individual register may vary significantly due to the layout characteristics of the clock distribution network, creating localized clock skew between sequentially adjacent registers. These initial clock delays can be changed by redesigning the clock distribution network, for example, by inserting buffers into certain clock paths. By applying this type of methodology, a clock distribution network can be designed which maintains monotonically increasing path delays and no race conditions, thereby satisfying (5.30) and (5.31). These inequalities are imposed on a path with three edges, i, j, and k, and two vertices, v_a and v_b, between the registers.

$$\begin{aligned} T_{CD}(j) - T_{CD}(i) &\leq \min\{\mathrm{T}_1(i,j), \tau_1(i,j)\}, \\ T_{CD}(k) - T_{CD}(j) &\leq \min\{\mathrm{T}_2(j,k), \tau_2(j,k)\}, \end{aligned} \tag{5.30}$$

$$T_{CD}(i) - T_{CD}(j) \leq \mathrm{T}(i,j). \tag{5.31}$$

Equations (5.30) and (5.31) represent the minimum and maximum clock skew that each individual local data path may have without causing improper circuit operation. As described in Section 5.5.4, values of τ and T are derived from the REC parameters of the path, $e_i \rightsquigarrow v_a \rightsquigarrow e_j \rightsquigarrow v_b \rightsquigarrow e_k$.

Careful observation of (5.30) and (5.31) will show that these expressions represent a family of inequalities which can be solved using the Bellman-Ford method [47]. Therefore, a clock distribution network can be systematically designed with this methodology, as shown by the algorithm presented in pseudo-code form in Figure 40. The process described in Step 5, designing the clock distribution network from the individual clock delays, is discussed in greater detail in [80, 92–94].

In the event that no solution for this set of inequalities exists, a clock distribution network design is not feasible. If this occurs, these monotonicity conditions cannot be satisfied and the algorithm *RETSAM* can be used.

```
1. Calculate the $\tau_1(i,j)$ and $\tau_2(i,j)$ values for each local data path $e_i \leadsto e_j$ in the synchronous circuit
2. Calculate the $T(i,j)$, $T_1(i,j)$, and $T_2(i,j)$ values for each local data path $e_i \leadsto e_j$ in the synchronous circuit
3. Write the inequalities for each local data path using (5.30) and (5.31)
4. Solve the inequalities with the Bellman-Ford method
5. Use the resulting clock delays to design the clock distribution network
```

Figure 40. Pseudo-code version of algorithm for designing the clock distribution network

A key aspect of the results of this research dissertation is the close interaction that exists between the design of the clock distribution network and the computational efficiency of the retiming process. It is shown herein that if the clock distribution network is poorly designed, the retiming process may be greatly degraded. Alternatively, a well designed clock distribution network may significantly enhance the efficiency of the retiming process.

5.5.3. Feasibility Check for the Clock Distribution Network

To verify whether the conditions imposed on the clock delays are feasible in practical circuits, typical values for the RECs are used to exemplify the design process. The original digital correlator presented by Leiserson-Saxe and shown in Figure 32 is used as an example circuit. The logic elements represented by the vertices v_1, v_2, v_3, and v_4 are comparators and are modeled as XNOR gates with a nominal delay value of 3.5 ns. The logic elements represented by vertices v_5, v_6, and v_7 are full adders with a nominal delay value of 4.0 ns. Typical register set-up, hold, and clock-to-Q times of 4.0 ns, 0.5 ns, and 3.0 ns are used. These temporal values are derived from industrial-based standard cell libraries.

The parameters are provided below for an arbitrary path $p : e_0 \rightsquigarrow e_1 \rightsquigarrow e_2$ shown in Figure 32. Interconnect capacitances, C_{Int1} and C_{Int2}, are assumed to each be 0.5 pf, and the output slope of both registers and vertices are assumed to be $m = 1\ k\Omega$ for every edge and vertex along path p. Input capacitances of both the logic elements and registers are assumed to be 0.2 pf. In this case, the τ and T values can be calculated as follows:

$$\begin{aligned} \mathrm{T}_1(e_0, e_1) &= [3 - 3]\ ns - 1\ k\Omega[0.5\ pf + 0.2\ pf] + 1\ k\Omega[0.5\ pf + 0.2\ pf] \\ &\quad +3.5\ ns + 1\ k\Omega[0.5\ pf + 0.5\ pf + 0.5\ pf] = 5\ ns, \\ \tau_1(e_0, e_1) &= 1\ k\Omega[0.5\ pf + 0.5\ pf] - \epsilon - [0.5\ ns - 3\ ns] \\ &\quad +3.5\ ns + 1\ k\Omega[0.5\ pf + 0.5\ pf] = 8\ ns - \epsilon. \end{aligned} \tag{5.32}$$

Assuming a small ϵ value,

$$\min\{\mathrm{T}_1(e_0, e_1), \tau_1(e_0, e_1)\} = 5\ ns. \tag{5.33}$$

$$\begin{aligned}\mathrm{T}_2(e_1, e_2) = [4 - 4]\ ns - 1\ k\Omega[0.2\ pf - 0.5\ pf - 0.2\ pf] \\ +3.5\ ns + 1\ k\Omega[0.5\ ns + 0.2\ ns] = 4.7\ ns,\end{aligned} \tag{5.34}$$

$$\begin{aligned}\tau_2(e_1, e_2) = 1\ k\Omega[0.5\ pf + 0.2\ pf] - [0.5\ ns - 3.0\ ns] - \epsilon \\ +3.5\ ns + 1\ k\Omega[0.5\ ns + 0.2\ ns] = 7.4\ ns - \epsilon.\end{aligned} \tag{5.35}$$

Assuming a small ϵ value,

$$\min\{\mathrm{T}_2(e_1, e_2), \tau_2(e_1, e_2)\} = 4.7\ ns. \tag{5.36}$$

According to (5.27),

$$T_{Skew}(e_1, e_0) \leq 5\ ns,$$

and

$$T_{Skew}(e_1, e_2) \leq 4.7\ ns, \tag{5.37}$$

These two inequalities place restrictions on the clock skew values for edges e_0, e_1, and e_2. For example, (36) suggests that as long as the clock signal on edge e_1 does not lead the clock signal on edge e_0 by more than 5 nanoseconds, or, as (5.37) suggests, as long as the clock signal on edge e_1 does not lag the clock signal on edge e_2 by more than 4.7 nanoseconds, the path delays on $e_0 \rightsquigarrow e_1 \rightsquigarrow e_2$ will increase monotonically. Therefore, for example, clock delays of $T_{CD}(e_0) = 1$ ns, $T_{CD}(e_1) = 2$ ns, $T_{CD}(e_2) = 3$ ns would operate properly since

$$T_{Skew}(e_1, e_0) = 1\ ns \leq 5\ ns, \tag{5.38}$$

and

$$T_{Skew}(e_1, e_2) = -1\ ns \leq 4.7\ ns. \tag{5.39}$$

A methodology for designing clock distribution networks based on non-zero localized clock skew is described in greater detail in [80, 92–94].

5.5.4. Derivation of the Monotonicity Constraints

Computationally efficient retiming cannot be guaranteed on a synchronous circuit with arbitrary clock delays yielding arbitrary data path delays. It may be preferable to design the clock distribution network to improve the computational efficiency of the retiming algorithm by ensuring that all maximum path delays are monotonically increasing, and minimum path delays do not indicate any race conditions. The derivation of the inequalities, (5.30) and (5.31), are provided in this section.

In Figure 41, a path with three registers and two vertices with delays $d(v_a)$ and $d(v_b)$, respectively, is depicted.

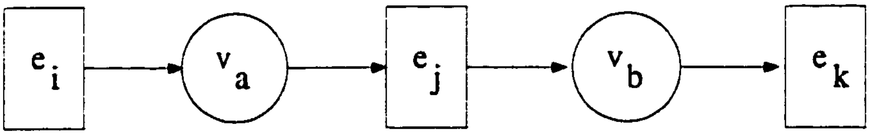


Figure 41. A path p with 3 registers and 2 vertices.

For this path consisting of three registers, i, j, and k, necessary conditions for monotonically increasing maximum path delays (smaller delays for sub-paths of larger paths) are

$$\begin{aligned} T_{PD_{max}}(i,k) &\geq T_{PD_{max}}(i,j), \\ T_{PD_{max}}(i,k) &\geq T_{PD_{max}}(j,k). \end{aligned} \tag{5.40}$$

To provide intuition into how these inequalities are created, consider the graph of Figure 39. In this figure, inequalities in (5.40) can be used to ensure that the sub-paths p_1 and p_2 each have a delay less than the longer path p. More generally, inequalities in (5.40) ensure that paths $e_i \rightsquigarrow e_j$ and $e_j \rightsquigarrow e_k$ each have a maximum delay less than the longer path $e_i \rightsquigarrow e_k$.

The minimum path delays must be greater than zero to ensure that the retimed circuit does not have any race conditions, i.e., every subpath is free of race conditions. For the path above, the conditions that must be imposed on minimum path delays are

$$\begin{aligned} T_{PD_{min}}(i,j) &\geq \epsilon, \\ T_{PD_{min}}(j,k) &\geq \epsilon, \end{aligned} \tag{5.41}$$

where ϵ is a process-dependant safety parameter. Note that the parameter $\geq \epsilon$ is used instead of > 0 to create a consistent set of inequalities, i.e., similar to (5.40).

Using (4.30), the following inequalities can be derived from (5.40) and (5.41).

$$\begin{gathered}
T_{C-Q_0}(i) + m(i)[C_{Int2}(i) + c(v_a)] + d_0(v_a) + m(v_a)[C_{Int1}(j) + C_{Int2}(j) + c(v_b)]+ \\
d_0(v_b) + m(v_b)[C_{Int1}(k) + c(k)] + T_{Set-up}(k) + T_{Skew}(i,k) \geq T_{C-Q_0}(i)+ \\
m(i)[C_{Int2}(i) + c(v_a)] + d_0(v_a) + m(v_a)[C_{Int1}(j) + c(j)] + T_{Set-up}(j) + T_{Skew}(i,j),
\end{gathered} \tag{5.42}$$

$$\begin{gathered}
T_{C-Q_0}(i) + m(i)[C_{Int2}(i) + c(v_a)] + d_0(v_a) + m(v_a)[C_{Int1}(j) + C_{Int2}(j) + c(v_b)]+ \\
d_0(v_b) + m(v_b)[C_{Int1}(k) + c(k)] + T_{Set-up}(k) + T_{Skew}(i,k) \geq T_{C-Q_0}(j) \\
m(j)[C_{Int2}(j) + c(v_b)] + d_0(v_b) + m(v_b)[C_{Int1}(k) + c(k)] + T_{Set-up}(k) + T_{Skew}(j,k).
\end{gathered}$$

$$\begin{gathered}
T_{C-Q_0}(i) + m(i)[C_{Int2}(i) + c(v_a)] + d_0(v_a) + m(v_a)[C_{Int1}(j) + c(j)] \\
-T_{Hold}(j) + T_{Skew}(i,j) \geq \epsilon,
\end{gathered} \tag{5.43}$$

$$\begin{gathered}
T_{C-Q_0}(j) + m(j)[C_{Int2}(j) + c(v_b)] + d_0(v_b) + m(v_b)[C_{Int1}(k) + c(k)] \\
-T_{Hold}(k) + T_{Skew}(j,k) \geq \epsilon.
\end{gathered}$$

Carrying out the cancellations, the following inequalities are obtained from (5.42) and (5.43).

$$
\begin{aligned}
T_{Skew}(i,j) &\geq \left[T_{C\rightarrow Q_0}(j) - T_{C\rightarrow Q_0}(i)\right] + m(j)[C_{Int2}(j) + c(v_b)] - \\
&\quad m(i)[C_{Int2}(i) + c(v_a)] - d_0(v_a) - m(v_a)[C_{Int1}(j) + C_{Int2}(j) + c(v_b)], \\
T_{Skew}(j,k) &\geq \left[T_{Set-up}(j) - T_{Set-up}(k)\right] + m(v_a)[c(j) - C_{Int2}(j) - c(v_b)] \\
&\quad - d_0(v_b) - m(v_b)[C_{Int1}(k) + c(k)],
\end{aligned}
\tag{5.44}
$$

$$
\begin{aligned}
T_{Skew}(i,j) &\geq \epsilon + \left[T_{Hold}(j) - T_{C\rightarrow Q_0}(i)\right] - m(i)[C_{Int2}(i) + c(v_a)] \\
&\quad - d_0(v_a) - m(v_a)[C_{Int1}(j) + c(j)], \\
T_{Skew}(j,k) &\geq \epsilon + \left[T_{Hold}(k) - T_{C\rightarrow Q_0}(j)\right] - m(j)[C_{Int2}(j) + c(v_b)] \\
&\quad - d_0(v_b) - m(v_b)[C_{Int1}(k) + c(k)].
\end{aligned}
\tag{5.45}
$$

These two conditions, (5.44) and (5.45), can be transformed into a single condition by defining two constants, τ_1 and T_1. Assume a path with three edges, i, j, and k, respectively, and two vertices between these edges. The following condition is required to ensure that the maximum path delays increase monotonically with increasing path length and that there are no race conditions,

$$
\begin{aligned}
T_{Skew}(j,i) &\leq \min\left\{\mathrm{T}_1(i,j), \tau_1(i,j)\right\}, \\
T_{Skew}(k,j) &\leq \min\left\{\mathrm{T}_2(j,k), \tau_2(j,k)\right\},
\end{aligned}
\tag{5.46}
$$

where for a path with two edges, a and b, and a vertex v in between the two edges, $\tau_1, \tau_2, \mathrm{T}_1$, and T_2 are defined as follows

$$\mathrm{T}_1(a,b) = \left[T_{C\to Q_0}(a) - T_{C\to Q_0}(b)\right] - m(b)[C_{Int2}(b) + c(v)] +$$
$$m(a)[C_{Int2}(a) + c(v)] + d_0(v) + m(v)[C_{Int1}(b) + C_{Int2}(b) + c(v)],$$

$$\tau_1(a,b) = m(a)[C_{Int2}(a) + c(v)] - \epsilon - \left[T_{Hold}(b) - T_{C\to Q_0}(a)\right]$$
$$+ d_0(v) + m(v)[C_{Int1}(b) + c(v)], \tag{5.47}$$

$$\mathrm{T}_2(a,b) = \left[T_{Set-up}(b) - \mathrm{T}_{Set-up}(a)\right] - m(v)[c(a) - C_{Int2}(a) - c(v)]$$
$$+ d_0(v) + m(v)[C_{Int1}(b) + c(b)],$$

$$\tau_2(a,b) = m(a)[C_{Int2}(a) + c(v)] - \left[T_{Hold}(b) - T_{C\to Q_0}(a)\right] - \epsilon$$
$$+ d_0(v) + m(v)[C_{Int1}(b) + c(b)]. \tag{5.48}$$

The equations in (5.46) set a lower limit on the skew of a local data path to ensure that only one inequality is required, making the system of inequalities linear, thereby permitting the use of the Bellman-Ford method. There is also an upper limit that can be defined as

$$T_{PD_{max}}(a,b) \le C, \tag{5.49}$$

where C is the maximum permitted clock period of the circuit. This upper limit is used to guarantee that each local data path has a delay smaller than the maximum permitted clock period of the synchronous circuit. Since monotonicity is guaranteed by (5.46), a longer path will have a greater delay. Therefore, the upper limit of (5.49) must be imposed on each local data path, since if (5.49) is not satisfied for each local data path, the excessively long local data path delay will place a new lower limit on the clock period of the circuit. If this upper limit is greater than required, the long path must be removed.

(5.49) can be rewritten in terms of the clock delays as follows:

$$\begin{gathered} T_{C-Q_0}(a) + m(a)[c(v) + C_{Int2}(a)] + d_0(v) + m(v)[C_{Int1}(b) + c(b)] \\ +T_{Set-up}(b) + T_{Skew}(a,b) \leq C, \end{gathered} \tag{5.50}$$

which translates to the following equation placing a constraint on the clock delays:

$$\begin{gathered} T_{Skew}(a,b) \leq C - T_{C-Q_0}(a) - m(a)[c(v) + C_{Int2}(a)] - d_0(v) \\ -m(v)[C_{Int1}(b) + c(b)] - T_{Set-up}(b) \end{gathered} \tag{5.51}$$

(5.51) can be converted to a constraint that is similar to (5.46) as follows,

$$T_{Skew}(a,b) \leq \mathrm{T}(a,b), \tag{5.52}$$

where, for a path with two edges a and b, and a vertex v between the two edges, $\mathrm{T}(a,b)$ is defined as

$$\begin{gathered} \mathrm{T}(a,b) = C - T_{C-Q_0}(a) - m(a)[c(v) + C_{Int2}(a)] - \\ d_0(v) - m(v)[C_{Int1}(b) + c(b)] - T_{Set-up}(b). \end{gathered} \tag{5.53}$$

5.6. Experimental Results

The retiming algorithm *RETSAM* is implemented in C on a SUN SPARC workstation. To permit evaluating the proposed retiming algorithm, modified MCNC benchmark circuits [97, 98] have been analyzed with this algorithm and an implementation of the Leiserson-Saxe retiming algorithm [14]. The resulting minimum clock period for each of the retimed benchmark circuits is reported. In the following two sections, the results of the application of the two algorithms to modified MCNC benchmarks is presented. In section 5.6.1, the method that is used to modify the MCNC benchmarks to incorporate the REC values is discussed and the results of the application of the *RETSAM* algorithm to those modified benchmarks is presented. In section 5.6.2, the impact of the pipelining depth on the clock period is discussed on a specific example benchmark circuit.

5.6.1. Application of *RETSAM* to Modified MCNC Benchmark Circuits

To evaluate the proposed retiming algorithm, 1989 and 1991 MCNC LGSynth benchmark circuits [97, 98] have been modified to include the effects of variable register, clock distribution, and interconnect delays as well as load-dependant register and logic delays. To incorporate these delay components into the benchmark circuits, RECs are artificially generated using a random number generator. However, to better simulate the effects of the actual clock distribution, interconnect, and register delays, the uniformly distributed numbers generated by the C library function **random()** are converted to a normal Gaussian distribution [99]. For clock delays, a uniform distribution is applied, since the registers are typically distributed over the entire integrated circuit. Physically distant registers may have very different clock delays, since the interconnect impedance between the clock source and these registers and the capacitive loading of the registers may vary over a wide range. This broad variation suggests a wide spectrum of clock delay values and therefore a uniform distribution is applied. For interconnect capacitances, a uniform distribution is also used, since the distance between the registers and the logic elements is assumed to vary uniformly. A Gaussian distribution is used for the register delays and output impedances, since similar instances of the same register cell or register cells of similar delay are most often used; therefore, the delays are approximated as being normally distributed. For those instances in which a negative value for the clock distribution, register, or interconnect delays is obtained from the Gaussian distribution, the approach applied in these experiments is to discard the negative values and redo the sample. The "truncation towards zero" approach (*i.e.*, mapping of negative values to zero) is not applied since this would bias the probability of obtaining zero values (specifically, the probability of obtaining a zero sample would be the integral of the Gaussian distribution from $-\infty$ to zero).

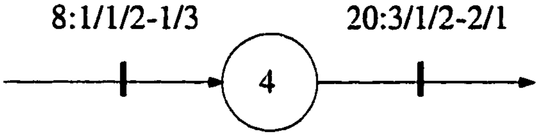
The application of *RETSAM* to the example MCNC benchmark circuits is described in Table 5. The initial five columns describe the properties of the modified benchmark circuits. These properties are 1) the name of the benchmark example as it appears in the MCNC archive, 2) the number of edges and 3) vertices in the graph of each circuit, 4) the latency of the circuit, and 5) the original clock period. The minimum clock period of the retimed circuit using algorithm *RETSAM* is shown in the sixth column. The same circuit shown in the sixth column is then converted to its no-interconnect and no-load delay form (i.e., $C_{Int1} = C_{Int2} = 0$ and $m = c = 0$ for every edge) and retimed using algorithm *RETSAM* and the resulting minimum clock period is provided in the seventh column. In the eighth column, the minimum T_{CP} biased with $T_{C \to Q} + T_{Set-up}$ using the classical LP retiming algorithm *FEAS* [14] is presented. The parameter, $T_{C \to Q} + T_{Set-up}$, shown separately on the eighth column, is the average register delay in the circuit and is included to provide a fairer comparison. For example, the circuit **cm42a** has an initial clock period of 126 tu (fifth column), which is reduced to 56 tu after retiming with the *RETSAM* algorithm (sixth column). When the load-dependence is ignored on all edges ($m = c = 0$) and the interconnect delays are assumed to be zero for every edge ($C_{Int1} = C_{Int2} = 0$), i.e., when only the effects of variable clock-to-Q and setup delays are considered, the retimed minimum clock period is reduced to 44 tu (seventh column). The *LP* based algorithm yields a clock period of 28 tu on a circuit with zero registers, interconnect delays, and zero load-dependence (the eighth column). The average register delay of the original circuit, i.e., the average $T_{C \to Q} + T_{Set-up}$, is 12 tu (the eighth column). Therefore, the sum of the retimed clock period using an *LP* based algorithm and the average register delay is 40 tu.

The sequential circuit represented by the retimed graph contains R registers with various REC values. If the register delays in the circuit are different, as shown in Figure 42 ($T_{C-Q} + T_{Set-up}$ = 2+1 tu vs. 2+3 tu), T_{Set-up} for the initial register and

T_{C-Q} for the final register must be considered when calculating the path delay. In the unusual case where all register delays are equal, $T_{C-Q} + T_{Set-up}$ can be used as a global parameter, as is assumed in [18]. The greater the variance between the register delays and between the clock delays (the more significant the affects of negative clock skew), the greater the minimum clock period becomes, as exemplified by the minimum clock period of certain benchmark circuits listed in the sixth column. This increased delay is due to the *imbalance* among the path delays, thereby increasing the worst case path delay, requiring a larger minimum clock period. Since the circuits listed in Table 5 are relatively balanced, negative clock skew does not significantly impact most of the circuits listed in Table 5, i.e., the seventh and the eigth columns are similar (e.g., 44 tu vs. 40 tu, respectively, in the example of **cm42a**).

In the ideal case where all delays and load factors are similar, these two columns are expected to be identical. However, the seventh column displays smaller values in many cases due to the fact that, since the clock delays are assumed to be zero in the eighth column, the affect of the negative clock skew is non-existant. However, in a few examples such as the last circuit of the LGSynth89 benchmarks (**parity**), the fourth circuit in the LGSynth91 benchmarks (**cm150a**), and the circuit of Figure 33 (**fig33**), the seventh column has lower values. This occurs since the negative clock skew has affected the retimed result in favor of the minimum clock period, i.e., the negative clock skew permits improved retiming results [43].

To provide a comparison between the CPU efficiency of the retiming process with



$$T_{PD} = T_{Reg} + T_{Int} + T_{Logic} + T_{Skew}$$

$$T_{PD} = (2+3) + (1*3+1*2) + \quad (4+1*1) \quad + (8-20)$$

$$T_{PD} = 3\ tu$$

Figure 42. A path containing two registers and a vertex between the two registers. The path delay T_{PD} contains components related to the register delays. If all registers in the circuit are similar, the register delay components would be equal.

and without monotonic delays, the CPU times are included in columns 9 and 10 of Table 5. The CPU times required to retime a circuit with monotonic path delays using a linear programming based algorithm similar to *FEAS* proposed in [14] is listed in column 9. The CPU times using *RETSAM* are listed in column 10. Note the dramatic improvement in CPU efficiency when a linear programming based algorithm is used. This emphasizes the importance of applying path delay monotonicity constraints when retiming a high complexity circuit.

5.6.2. The Impact of Latency on the Retimed Clock Period

In Table 5, the original clock periods of certain modified MCNC benchmark circuits

Table 5: Results of the application of the retiming algorithm to MCNC benchmark circuits

Example	Graph properties				T_{CP} After Retiming (tu)	T_{CP} After Retiming (tu) $C_{Int}=0$	T_{CP} $T_{SKEW}=0$ $T_{REG}=const$ (tu)	CPU Time	
	Edges	Vertices	Latency	Initial T_{CP} (tu)				LP Based (sec)	*RETSAM* (sec)
LGSynth89 - multi-level (netblif)									
C17	26	19	2	112	65	54	42+11	0.33	1.1
C17	26	19	3	112	46	39	29+11	0.33	4.5
C17	26	19	4	112	43	38	26+11	0.33	10.5
C17	26	19	5	112	37	29	14+11	0.33	58
C17	26	19	6	112	36	28	14+11	0.33	116
C17	26	19	7	112	36	28	14+11	0.33	423
b1	34	19	3	93	52	47	24+11	0.32	1.2
cm42a	65	34	3	126	56	44	28+12	0.60	80
cm82a	59	37	4	157	55	44	30+12	0.67	314
majority	26	17	5	112	40	35	24+11	0.31	7.1
parity	138	93	2	244	127	106	99+11	3.6	116
LGSynth91 - multi level (blif)									
C17	19	12	5	71	38	30	14+11	0.28	6.7
b1	16	10	2	66	41	29	16+11	0.27	0.33
cm82a	22	12	3	59	38	34	14+11	0.26	0.66
cm150a	69	38	2	105	60	47	37+11	0.67	58
cm151a	38	22	3	108	52	40	26+12	0.38	64
decod	89	24	4	79	56	47	26+11	1.49	97
parity	47	32	2	103	67	51	30+12	0.53	61
Figure 33 with the added REC values									
fig33	11	8	4	37*	24	17	13+5	0.22	0.45

* denotes a graph that contains race conditions before retiming.

is compared to their corresponding retimed clock periods. The artificial REC values are added to these benchmark circuits using either uniform or normal distribution. To convert the combinatorial LGSynth89 and LGSynth91 circuits, a dummy vertex is added at the input of the circuit which connects all of the circuit inputs to all of the circuit outputs. Thus, a closed-loop circuit is created out of every benchmark circuit similar to that of the Leiserson-Saxe example circuit. This artificial *host* vertex is used to convert the combinatorial MCNC benchmark circuits to a synchronous one by adding multiple registers at every input of the host vertex. The number of added registers determines the latency of the circuit as listed in the fourth column of Table 5.

Arbitrary latencies are chosen for all of the benchmarks circuits listed in this table, with the exception of LGSynth89 benchmark circuit **C17**. The latency of this circuit is varied from two to seven and the results of the application of both *LP* based retiming and algorithm *RETSAM* are shown on the first six rows of Table 5. These six rows permit analysis of the impact of pipelining depth on the retimed clock period. At lower latency values (e.g., 2 and 3), the impact is much more significant. The saturation point is reached at a latency of six ($l = 6$). For $l > 6$, further increasing the latency has no effect on the retimed clock period, since the retimed clock period is limited by the vertex delays between any two registers. This phenomenon of the *theoretical minimum* clock period has been investigated by Papaefthymiou in [66]. Papaefthymiou deduces an *approximate* T_{CP}/l theoretical minimum clock period, although the effects of negative clock skew is ignored in [66].

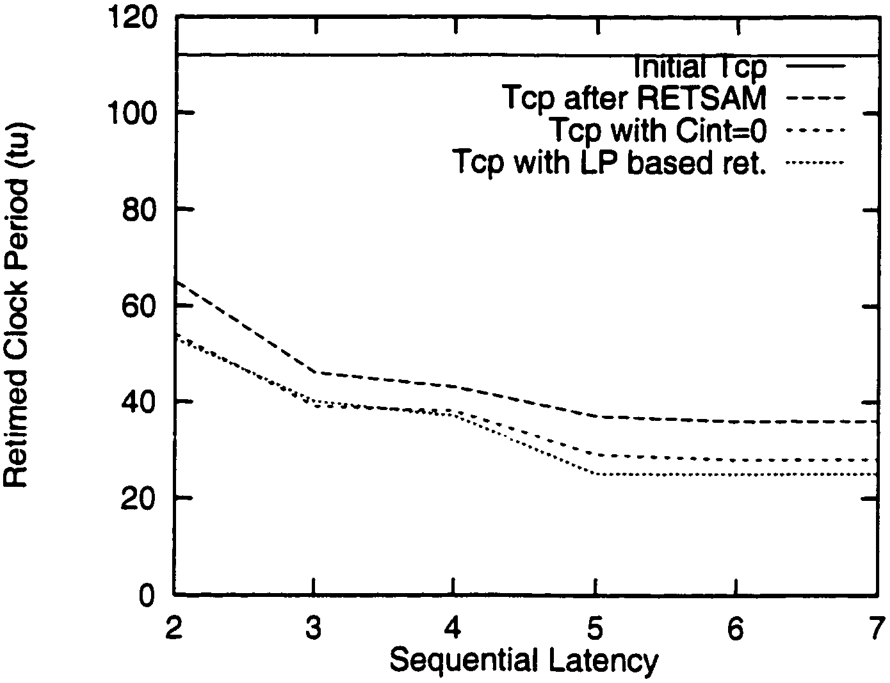


Figure 43. Effect of the pipelining depth on the clock period on C17. The initial clock period is 112 tu which is shown as a solid line.

5.7. Conclusions

A retiming algorithm is presented which considers variable clock distribution, register, and interconnect delays as well as the load-dependence of the register and logic delays. To permit the consideration of these delay components, register electrical characteristics (RECs) are attached to each edge of the graph representing the circuit and the original path delays are redefined to be from edge-to-edge rather than vertex-to-vertex. A set of inequalities are created based on these edge-to-edge path delays, permitting a retimed version of the circuit to operate at the minimum clock period. A general algorithm, *RETSAM*, is presented which supports arbitrary REC values, including excessive negative clock skew. An iterative method using ranges of vertex lags rather than constant vertex lags is used within this retiming algorithm to solve for the edge weights.

A set of monotonicity conditions may be imposed on the REC values to improve the computational efficiency by permitting the use of standard linear programming methods. These monotonicity conditions place constraints on the magnitude of the negative clock skew of each local data path, thereby no longer permitting the clock delays to be of arbitrary value. The feasibility of applying these conditions to practical circuits is discussed. It is shown that retiming cannot be efficiently and accurately performed on a circuit with an improperly designed clock distribution network. Thus, the quality of the design of the clock distribution network can significantly affect the automated design of high performance synchronous circuits when utilizing retiming as a synthesis methodology.

The limitations and advantages of the proposed retiming algorithm are compared with existing retiming strategies using a set of modified MCNC benchmark circuits. The results of applying *RETSAM* to the benchmark circuits show that a more accurate retiming can be performed than with existing retiming algorithms which do not consider variable clock distribution delays and load-dependant register and interconnect delays. Additionally, the clock period can be further minimized due to localized negative clock skew. Finally, clock skew induced race conditions are detected and eliminated.

Summarizing, a new retiming algorithm which considers the effects of variable clock distribution delays and load-dependant register and interconnect delays has been presented. This algorithm represents a significant extension of existing retiming algorithms, permitting the use of retiming for the automated synthesis of higher speed, more reliable pipelined digital systems.

Chapter 6. Conclusions

As achieving higher clock frequencies becomes a more important design issue and the circuit area becomes a less constrained design parameter, the need to develop new methodologies to improve the synchronous speed at the expense of increased circuit area arises. This dissertation addresses methodologies to optimally locate and clock the flip-flops inside a synchronous circuit so as to improve the synchronous clock speed, typically at the expense of circuit area.

A new timing model that permits characterizing the low-level circuit details in a VLSI circuit is presented in Chapter 4. This model, called the Register Electrical Characteristic (REC) model, permits modeling low-level circuit characteristics such as the clock delays, load-dependent register and interconnect delays, and variable and load-dependent register and logic delays. The REC model presented in this dissertation is the first research result that incorporates lower level delay characteristics into the retiming process. More specifically, the set-up time (T_{Set-up}), hold time (T_{Hold}), and the load-dependent clock-to-Q delay ($T_{C \rightarrow Q}$) of the registers, interconnect capacitances along the connections between the registers and logic elements (C_{Int}), the output impedance of the registers and logic elements (m), the input capacitance of the registers and logic elements (c), and the clock delays from the global clock source to each flip-flop (T_{CD}) have been characterized in the REC model. Incorporating the RECs into the synchronous optimization process permits achieving significantly more accurate (e.g., 20–30%) optimization results.

A new branch and bound based retiming algorithm, called *RETSAM*, has been introduced in Chapter 5 that permits retiming a synchronous circuit with arbitrary REC model parameters. The fundamental deviation between *LP* based algorithms existing in the literature and *RETSAM* is the redefinition of the path delays to be from edge-to-edge rather than vertex-to-vertex. With this new definition, paths initiate and terminate at registers rather than logic elements. The timing constraints that are used in existing retiming algorithms have been reformulated to incorporate the REC model. Due to the complexity of these new timing models, *LP* based algorithms could not be used for the retiming process. This necessitates the use of branch and bound techniques in algorithm *RETSAM*, thereby yielding exponential run-time.

If the REC parameters are not restricted and are totally arbitrary, branch and bound techniques must be used and polynomial run-time can not be achieved. However, an important observation of this dissertation is that it is possible to place *monotonicity* constraints on the path delays to permit using *LP* based algorithms on circuits containing restricted REC parameters. These path delay monotonicity constraints, presented in Section 5.5, permit using *LP* based algorithms by ensuring that the path delays increase monotonically. As described in Section 5.5.2, it is possible in many cases to design the clock distribution network to permit computationally inexpensive retiming. This is achieved by specifying the clock delays along a given path so as to achieve monotonically increasing path delays. As described in Section 5.5, in a graph with monotonically increasing path delays, the aforementioned complex timing constraints can be reduced to the simple inequalities found in [14], thereby permitting the use of *LP* based methods. This important result of this dissertation demonstrates the significant correlation between the clock distribution design process and the retiming process. It is shown in Section 5.5 that a poor design of the clock distribution network deteriorates the quality of the retiming process. Thus, the retiming and clock distribution design processes are inextricably intertwined.

Both algorithm *RETSAM* and an implementation of the *LP* based methods have been evaluated on several MCNC benchmark examples. Since the standard MCNC benchmark circuits do not incorporate the REC parameters, these parameters have been created artificially using standard and uniform distribution models. Although the algorithm *RETSAM* is impractically slower than *LP* based retiming algorithm, as demonstrated in Table 5, this algorithm may prove useful for small portions of a more complex circuit (e.g., a sub-circuit of an ALU in a complex microprocessor). In other words, improved accuracy may be obtained by retiming small subcircuits of a larger circuit using *RETSAM*, which may provide better intuition into how best to retime the entire circuit. However, for the larger circuits, *LP* based methods should be used due to the computational complexity of algorithm *RETSAM*. However, *LP* based methods can only be used after ensuring monotonically increasing delays, i.e., the clock delays must be calculated using the formulas presented in Section 5.5.2.

In summary, this research described in this dissertation presents an approach to retiming synchronous circuits with added low-level circuit characteristics by modeling the lower level circuit details using the REC model. The development of the REC model is the key result of the research presented in this dissertation which permits incorporating low level circuit characteristics into the retiming process for the first time. This objective is achieved at the expense of CPU time for generic circuits that contain unrestricted clock delays. The exponential time algorithm *RETSAM*, presented in this dissertation, is useful for retiming small circuits, such as a sub-portion of a larger circuit. The extremely accurate results obtained from this unrestricted retiming can be used to gain intuition into the functionality of a larger circuit. Alternatively, for circuits with restricted clock delays, the path delays increase monotonically, and the *LP* based algorithms that exist in the literature can be used. Thus, circuits with restricted REC values can be retimed asymptotically as fast as the circuits not containing REC values, thereby not requiring additional time complexity for the added generality.

Chapter 7. Future Work

The research results presented in Chapter 5 describe a timing model that incorporates low-level circuit details such as non-zero clock skew, load-dependent interconnect delay, and load-dependent register and logic delays. Incorporating these characteristics into a retiming algorithm permits automating the pipelining process of large VLSI circuits without losing accuracy. Significant improvements in the accuracy and reliability of the synthesized circuits using automated techniques is achieved by considering these low-level circuit effects.

As mentioned in Chapter 5, the combinatorial MCNC benchmark circuits are converted to their pipelined counterpart by using an artificial host vertex and adding multiple registers to the input of the input of the host vertex. The retiming process distributes these registers to proper locations inside the benchmark circuit, thereby yielding a minimum clock period. It is not clear, however, what the relationship is between the pipelining depth and the minimum obtainable clock period after retiming. As the results from investigating the example **C17** suggests in Table 5, increasing the pipelining depth does not decrease the minimum clock period beyond the *saturation latency* (e.g., 6 for the example of **C17**). Therefore, a study of the relationship between the latency and the minimum obtainable clock period may lead to important research results that shine light on undiscovered aspects of the pipelining-retiming relationship.

In Section 7.1, the relationship between the pipelining depth and the minimum obtainable clock period is discussed. This relationship is compared to the maximum average-weight cycle research by Papaefthymiou [66]. Possible future research this effect is exemplified by the retiming results on the benchmark circuit **C17**.

In this dissertation, the primary focus has been on applying retiming to general synchronous circuits. With the aforementioned enhancements to the retiming process, synchronous circuits can be accurately retimed using the proposed algorithms. Another possible future research area is the application of these techniques to more specific recursive structures such as IIR filters. Due to the long feedback path, applying automated pipelining to recursive structures such as IIR digital filters poses a great challenge. Although this issue has been addressed by a few research groups, a thorough study incorporating low-level details embodied by the REC models is necessary for improved accuracy. Demonstration of the application of the REC model on IIR filters and quantifying the sampling rate improvement of IIR filters due to pipelining can lead to significant new research results.

In summary, there are two primary areas of possible future research: 1) studying the relationship between the pipelining depth and the minimum obtainable clock period, and 2) applying automated pipelining techniques to recursive structures. In Section 7.1, the relationship between the latency and the minimum clock period is exemplified. A brief theoretical background on pipelining recursive structures such as IIR filters is provided in Section 7.2. With this foundation, possible future research includes enhancing existing methods for applying retiming algorithms to the pipelining of recursive digital signal processing structures.

7.1. Studying the Impact of Pipelining Depth on Retimed Minimum Clock Period

LGSynth89 and LGSynth91 benchmark circuits from the MCNC archive consist of multi-level netblif and blif type combinatorial circuits [97, 98]. These circuits are specified using logic gates and connections among these gates. The retiming algorithm described in Chapter 5, however, requires sequential circuits incorporating the REC values. To convert the MCNC benchmarks to this desired form, a host vertex is added to each benchmark circuit as described in Section 5.6.1.

The latency of the circuits is chosen arbitrarily in all but one case. The latency of the LGSynth89 circuit **C17** is varied from 2 to 7 to analyze the impact of the latency on the minimum clock period. The minimum clock period is calculated by applying three different retiming methods and the resulting clock periods are plotted in Figure 43. The original clock period of the circuit is 112 tu, which is shown by the straight line in Figure 43. The minimum clock period obtained by using the *RETSAM* algorithm is depicted using the dashed line below the initial clock period. The third line shows the retimed minimum clock period by assuming zero interconnect delays and load-dependence (i.e., $C_{Int1} = C_{Int2} = m = c = 0$ for every edge in the circuit). Finally, the bottom line represents the retimed minimum clock period period using *LP* based retiming biased with the average register delay.

Retiming with *RETSAM* always yields higher minimum clock periods among the three different retimed circuits. This behavior is due to the fact that the negative clock skew increases the imbalance between the path delays, thereby resulting in a higher critical path delay. The *LP* based retiming results yield the minimum clock period among the three methods, simply due to the non-existence of clock skew and load-dependence delays. When the load-dependence and interconnect delays are ignored, a result that is in between the two methods is obtained, i.e., only the effect of the negative clock skew is considered, and the load-dependence is ignored.

When three of these results saturate at a latency of six, there is a fixed distance between any two methods. In other words, 36 tu, 28 tu, and 25 tu are the minimum saturated clock periods for these curves. Further increasing the latency (i.e., the pipelining depth) does not result in a circuit with a lower minimum clock period beyond the saturation point since the maximum depth pipelining level is reached at this point and the circuit cannot be pipelined any deeper.

Another important observation from Figure 43 is that the greater the pipelining depth, the less the gains achieved from pipelining. For example, the gain achieved by increasing the latency from two to three is significant, whereas the gain by increasing the latency from three to four is fairly negligible. This limit on the minimum obtainable clock period raises the question of whether it is possible to determine the theoretical limit on the retimed circuit or the optimal pipelining depth.

Previous work in this field is described by Papaefthymiou [66]. In [66], the theoretical lower bound on the minimum clock period of a retimed circuit in terms of cycle delays is studied. The theoretical bound is determined to be:

$$\max \left\{ \frac{\sum\limits_{v \in C_i} d(v)}{\sum\limits_{e \in C_i} w(e)}, \ \forall i, \ C_i \in C \right\} \tag{7.1}$$

where the numerator represents the total delay along a given cycle C_i in the graph, and the denominator represents the total latency of the cycle C_i. The ratio of the total delay to the latency for a given cycle C_i sets a lower bound on the theoretical minimum clock period. Thus, the maximum of these ratios for every cycle in the set of all existing cycles in the circuit (C) is the theoretical limit on the minimum obtainable clock period.

Since a dummy host vertex is used to connect the input of the benchmark circuits to the output, every global data path in these benchmarks is converted to a cycle. Therefore, the theoretical limit on any given benchmark circuit is T_{CP}/l, where T_{CP} is the original unretimed clock period and l is the latency of the circuit. For the specific circuit of **C17**, the ratio of T_{CP}/l is depicted in Figure 44.

As shown in Figure 44, all three retimed results are higher than the theoretical limit. Although the work in [66] presents powerful results, the effect of the clock skew is ignored. The proposed research includes revising (7.1) to provide a better understanding of the limiting effect of cycles in the circuit by considering non-zero clock skew and load-dependent register and interconnect delays. Also, the study of the aforementioned *optimal pipelining depth* can lead to potentially powerful research results.

7.2. Pipelining Recursive DSP Circuit Architectures

Pipelining and retiming techniques described in the previous chapters offer performance improvements for digital synchronous circuits, thereby permitting dramatic performance enhancements. Pipelining techniques may increase the speed of operation

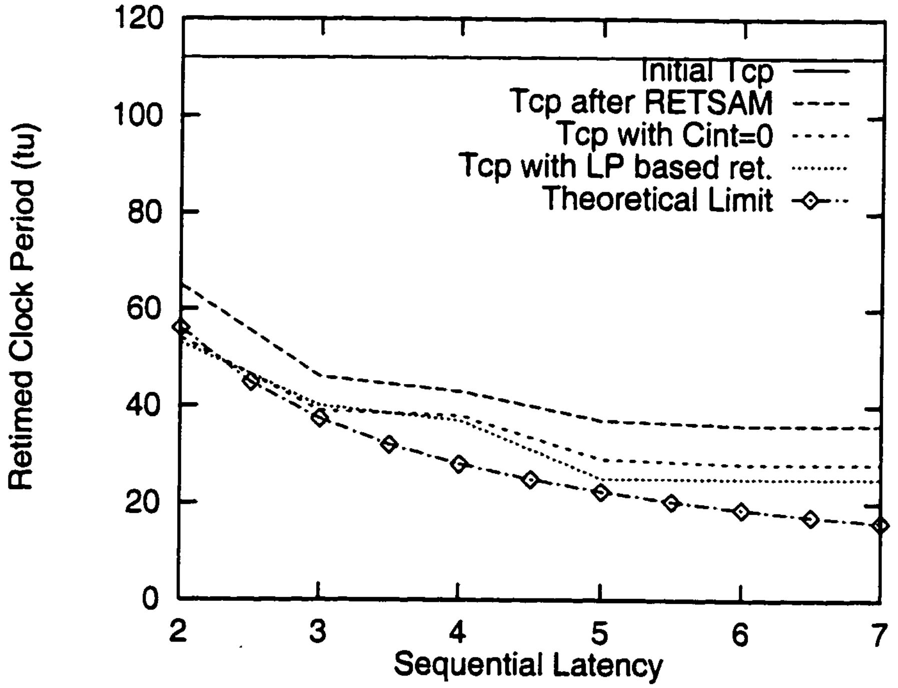


Figure 44. The theoretical limit from Figure 43. Note that the effect of negative clock skew is ignored.

by a few orders of magnitude at the expense of increased circuit area due to the added pipeline registers. As technology advances, physical area has become less significant, since the integration density has improved dramatically due to the reduced size of the transistors and interconnect.

Although pipelining techniques offer significant speed improvements, the application of these techniques to structures containing recursive feedback loops must be carefully evaluated. Possible future research is the application of these pipelining techniques to IIR digital filters. Previous work in the field of pipelining recursive structures is briefly overviewed in the following section.

7.2.1. Theoretical Background on IIR Filter Characterization

An IIR filter can generally be characterized by a transfer function $H(z)$ as shown below,

$$H(z) = \frac{N(z)}{D(z)}, \tag{7.2}$$

where the roots of the numerator polynomial $N(z)$ are the *zeros* and the roots of the denominator polynomial $D(z)$ are the *poles* of the digital filter. The numerator $N(z)$ characterizes the feed-forward section of the filter, whereas the denominator $D(z)$ characterizes the feedback portion. A generic IIR filter is not easily pipelined due to the existence of these long feedback path [11] characterized by $D(z)$. However, augmentation of the transfer function using a polynomial $P(z)$ permits the system to be pipelined [5, 6] as follows:

$$H(z) = \frac{N(z)P(z)}{D(z)P(z)}. \tag{7.3}$$

The roots of the polynomial $P(z)$ introduce cancelling pole-zero pairs in the new transfer function. Since the new nominator $N(z)P(z)$ contains more terms than the original nominator $N(z)$, the complexity of the feed-forward portion of the filter is increased. However, the new denominator $D(z)P(z)$ can now be pipelined. Depending upon the form of the function $P(z)$, different levels and qualities of pipelining can be achieved. The two main techniques that exist are scattered look-ahead and clustered look-ahead pipelining techniques as introduced by Parhi in [5]. A sub-category of the clustered look-ahead technique, called the Minimum Denominator Multiplier (MDM), is also introduced by Soderstrand *et al.* in [11] and is reviewed in this section.

7.2.2. Clustered Look-Ahead Pipelining

The clustered look-ahead technique is based on modifying the transfer function shown below by introducing M cancelling pole-zero pairs.

$$H(z) = \frac{\sum_{i=0}^{N} b_i z^{-i}}{1 - \sum_{i=1}^{N} a_i z^{-i}}. \tag{7.4}$$

The pole-zero pairs that are introduced transform $H(z)$ such that the coefficients of $z^{-1}, z^{-2}, \ldots, z^{M-1}$ in the denominator function are zero. The transformed transfer function becomes an M-stage pipelined digital filter.

As an example, a filter with the following original transfer function is considered [5]:

$$H(z) = \frac{1}{1 - \frac{5}{4}z^{-1} + \frac{3}{8}z^{-2}}. \tag{7.5}$$

This filter is not pipelineable since the lowest non-zero coefficient term in the denominator is z^{-1}. However, the filter represented by $H(z)$ is stable since the poles of $H(z)$ are located at $z = \frac{1}{2}$ and $z = \frac{3}{4}$ (inside the unit circle).

By introducing a pole-zero pair at $z = -\frac{5}{4}$, *i.e.*, by multiplying both the numerator and the denominator by $\left(1 + \frac{5}{4}z^{-1}\right)$, the transfer function is transformed to

$$H'(z) = \frac{1 + \frac{5}{4}z^{-1}}{1 - \frac{19}{16}z^{-2} + \frac{15}{32}z^{-3}}, \tag{7.6}$$

which is a two stage pipelined system. Note, however, that the resulting filter represented by $H'(z)$ is not stable since the introduced pole is not inside the unit circle although the original filter is stable.

The transfer functions resulting from the aforementioned conversion correspond to an M-stage pipelined implementation since the output sample $y[n]$ can be described in terms of the *cluster* of N past outputs, $y[n-M]$, $y[n-M-1]$, . . . , and $y[n-M-N+1]$. This technique is therefore called clustered look-ahead pipelining.

7.2.3. Scattered Look-Ahead Pipelining

The scattered look-ahead technique is based on modifying the transfer function based on the following formula [100],

$$\frac{1}{z-p_i} = \frac{z^{M-1} + p_i z^{M-2} + \cdots + p_i^{M-1}}{z^M - p_i^M}, \tag{7.7}$$

where p_i is a given pole of the transfer function, and M is the resulting degree of pipelining that is being achieved. Note that although (7.7) converts pole p_i into a pipelined form, $M - 1$ superfluous terms are introduced in the numerator (the feed-forward section of the filter). For a filter having N poles, $N(M - 1)$ superfluous terms are introduced in the numerator, and the resulting denominator is in the form of $D'(z^M)$ rather than $D(z)$. This transformation introduces N pole-zero cancellations and the resulting transfer function is stable as long as the original transfer function is stable. This behavior occurs since the poles of the modified transfer function are inside the unit circle as long as the original poles are inside the unit circle. The poles of the modified function are located inside the unit circle at the same distance from the origin. Therefore, although this system introduces more terms in the numerator, the resulting filter is guaranteed to be stable as long as the original system is stable.

As an example, consider a filter with the following transfer function [5],

$$H(z) = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}, \tag{7.8}$$

which, by using the proper $P(z)$, can be converted into the form,

$$H'(z) = \frac{1 + a_1 z^{-1} + (a_1^2 + a_2) z^{-2} - a_1 a_2 z^{-3} + a_2^2 z^{-4}}{1 - (a_1^3 + 3a_1 a_2) z^{-3} - a_2^3 z^{-6}}. \tag{7.9}$$

As shown in (7.9), the transfer function is in the form of $H(z^{-3})$ rather than $H(z^{-1})$. As long as the roots of the denominator of $H(z)$ are inside the unit circle, the roots of the denominator of $H'(z)$ will also be inside the unit circle. Since the terms in the denominator are *scattered* in $H'(z)$ (the terms are z^{-3} apart from each other), this technique is called scattered look-ahead pipelining.

7.2.4. Minimum Denominator Multiplier (MDM)

The MDM technique introduced in [11] is a slight modification of the clustered look-ahead pipelining method based on augmenting the transfer function in such a way that the stability of the pipelined IIR filter is achieved only by increasing the pipeline delay without adding non-zero denominator coefficients. This technique, therefore, offers a combination of the two aforementioned techniques by exploiting the stability advantage of the scattered look-ahead pipelining while exploiting the low circuit complexity of the clustered look-ahead pipelining scheme.

To exemplify the MDM technique, consider a second-order IIR filter with the following transfer function described in [11]:

$$H(z) = \frac{N(z)}{D(z)} = \frac{1 + n_1 z^{-1} + n_2 z^{-2}}{1 + d_1 z^{-1} + d_2 z^{-2}}. \tag{7.10}$$

When the denominator is augmented by $P(z)$, the new denominator takes the form,

$$B(z) = D(z)P(z) = 1 + b_1 z^{-(M+E_1)} + b_2 z^{-(M+E_1+E_2+1)}, \tag{7.11}$$

where M is the required pipeline delay, and E_1 and E_2 are the extra delay parameters to ensure stability. b_1 and b_2 parameters are calculated according to

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} p_1^{-E_1} & p_1^{-(E_1+E_2+1)} \\ p_2^{-E_1} & p_2^{-(E_1+E_2+1)} \end{bmatrix}^{-1} \begin{bmatrix} -p_1^M \\ -p_2^M \end{bmatrix} \tag{7.12}$$

by changing E_1 and E_2 according to $0 \leq E_1 + E_2 \leq M - 1$. In the second-order case, the $E_1 + E_2 + 1$ scattering patterns exist and the solution with the smallest maximum magnitude of the superfluous poles provides the lowest round-off errors.

The aforementioned three techniques consider the primary methods existing in the literature for pipelining IIR filters. Possible research includes an investigation of the sampling rates of the pipelined filters and stability considerations by applying the enhanced REC model to the circuit graph of these filters. The application of the REC model is necessary to accurately determine the possible sampling rate enhancements after pipelining these filters. The sampling rate of a group of selected IIR filters must be determined on pipelined and unpipelined IIR filters for different pipelining levels. The results can be used to demonstrate both the effectiveness of pipelining and the limitation of pipelining due to low-level circuit effects, such as interconnect delay, register delay, and clock skew.

Bibliography

[1] L. W. Cotten, "Circuit Implementation of High-Speed Pipeline Systems," *Proceedings of the AFIPS Fall Joint Computer Conference*, Vol. 27, pp. 489–504, November 1965.

[2] L. W. Cotten, "Maximum-Rate Pipeline Systems," *Proceedings of the Spring Joint Computer Conference*, Vol. 34, pp. 581–586, May 1969.

[3] D. Wong, De Micheli, Giovanni, and M. Flynn, "Inserting Active Delay Elements to Achieve Wave Pipelining," *Proceedings of the IEEE Conference on Computer-Aided Design*, pp. 270–273, November 1989.

[4] D. C. Wong, G. De Micheli, and M. J. Flynn, "Designing High-Performance Digital Circuits Using Wave Pipelining: Algorithms and Practical Experiences," *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 1, pp. 25–46, January 1993.

[5] K. K. Parhi and D. G. Messerschmitt, "Pipeline Interleaving and Parallelism in Recursive Digital Filters — Part I: Pipelining Using Scattered Look-Ahead and Decomposition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, No. 7, pp. 1099–1117, July 1989.

[6] K. K. Parhi and D. G. Messerschmitt, "Pipeline Interleaving and Parallelism in Recursive Digital Filters — Part II: Pipelined Incremental Block Filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, No. 7, pp. 1118–1134, July 1989.

[7] M. Lapointe, H. T. Huynh, and P. Fortier, "Systematic Design of Pipelined Recursive Filters," *IEEE Transactions on Computers*, Vol. 42, No. 4, pp. 413–426, April 1993.

[8] P. S. R. Diniz, J. E. Cousseau, and A. Antoniou, "Improved parallel realisation of IIR adaptive filters," *IEE Proceedings — G*, Vol. 140, No. 5, pp. 322–328, October 1993.

[9] J. Chung and K. K. Parhi, "Pipelining of Lattice IIR Digital Filters," *IEEE Transactions on Signal Processing*, Vol. 42, No. 4, pp. 751–761, April 1994.

[10] E. Q. Wong, M. A. Soderstrand, and H. H. Loomis, "Computer-Aided Design of Pipelined IIR Digital Filters," *IEEE International Symposium on Circuits and Systems*, pp. 795–799, May 1992.

[11] M. A. Soderstrand and A. E. de la Serna, "Minimum Denominator-Multiplier Pipelined Recursive Digital Filters," *IEEE Transactions on Circuits and Systems*, Vol. CAS–42, No. 10, pp. 666–672, October 1995.

[12] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," *Proceedings of Annual Symposium on Foundations of Computer Science*, pp. 23–36, October 1981.

[13] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing Synchronous Circuitry by Retiming," *Proceedings of the Caltech Conference on VLSI*, pp. 87–116, March 1983.

[14] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, Vol. 6, pp. 5–35, January 1991.

[15] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming of Circuits with Single Phase Transparent Latches," *Proceedings of the IEEE International Conference on Computer Design*, pp. 86–89, October 1991.

[16] B. Lockyear and C. Ebeling, "The Practical Application of Retiming to the Design of High-Performance Systems," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 288–295, November 1993.

[17] B. Lockyear and C. Ebeling, "Optimal Retiming of Multi-Phase, Level-Clocked Circuits," *Proceedings of the Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, pp. 265–280, March 1992.

[18] G. De Micheli, "Synchronous Logic Synthesis: Algorithms for Cycle-Time Minimization," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-10, No. 1, pp. 63–73, January 1991.

[19] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Integration of Clock Skew and Register Delays into a Retiming Algorithm," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1483–1486, May 1993.

[20] K. N. Lalgudi and M. C. Papaefthymiou, "Efficient Retiming under a General Delay Model," *Proceedings of the Chapel Hill VLSI Conference*, pp. 368–382, 1995.

[21] K. N. Lalgudi and M. C. Papaefthymiou, "DELAY: An Efficient Tool for Retiming with Realistic Delay Modeling," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 304–309, June 1995.

[22] J. P. Fishburn, "Clock Skew Optimization," *IEEE Transactions on Computers*, Vol. C–39, No. 7, pp. 945–951, July 1990.

[23] T. Soyata and E. G. Friedman, "Retiming with Non-Zero Clock Skew, Variable Register, and Interconnect Delay," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 234–241, November 1994.

[24] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and Design of Latch-Controlled Synchronous Digital Circutis," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 111–117, June 1990.

[25] R. B. Deokar and S. S. Sapatnekar, "A Fresh Look at Retiming via Clock Skew Optimization," *Proceedings of the IEEE/ACM Design Automation Conference*, pp. 310–315, June 1995.

[26] B. Lockyear and C. Ebeling, "Optimal Retiming of Multi-Phase, Level-Clocked Circuits," Tech. Rep. 91–10–01, University of Washhington, October 1991.

[27] T. M. Burks, K. A. Sakallah, and T. N. Mudge, "Multiphase Retiming Using *minTc*," *ACM/SIGDA Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pp. 1-9, March 1992.

[28] B. Lockyear and C. Ebeling, "Optimal Retiming of Level-Clocked Circuits Using Symmetric Clock Schedules," *IEEE Transactions on Computer-Aided Design*, Vol. 13, No. 9, pp. 1097–1109, September 1994.

[29] L. Chao and E. H. Sha, "Retiming and Clock Skew for Synchronous Systems," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1.283–1.286, May/June 1994.

[30] T. M. Burks, K. A. Sakallah, K. Bartlett, and G. Borriello, "Performance Improvement through Optimal Clocking and Retiming," *International Workshop on Logic Synthesis*, pp. 1-9, May 1991.

[31] A. T. Ishii, C. E. Leiserson, and M. C. Papaefthymiou, "Optimizing Two-Phase, Level-Clocked Circuitry," *Proceedings of the Conference on Advanced Research in VLSI and Parallel Systems*, pp. 245–264, March 1992.

[32] A. T. Ishii, "Retiming Gated-Clocks and Precharged Circuit Structures," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 300–307, November 1993.

[33] S. Simon, E. Bernard, M. Sauer, and J. A. Nossek, "A New Retiming Algorithm for Circuit Design," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 4.35–4.38, May/June 1994.

[34] S. Simon, J. Hofner, and J. A. Nossek, "Retiming of Circuits Containing Multiplexers," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1736–1739, May 1995.

[35] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming Sequential Circuits for Low Power," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 398–402, November 1993.

[36] M. J. Flynn, "Very High-Speed Computing Systems," *Proceedings of the IEEE*, Vol. 54, No. 12, pp. 1901–1909, December 1966.

[37] T. Soyata and E. G. Friedman, "Synchronous Performance and Reliability Improvement in Pipelined ASICs," *Proceedings of the IEEE ASIC Conference*, pp. 383–390, September 1994.

[38] E. G. Friedman and J. H. Mulligan, "Clock Frequency and Latency in Synchronous Digital Systems," *IEEE Transactions on Signal Processing*, Vol. SP–39, No. 4, pp. 930–934, April 1991.

[39] E. G. Friedman and J. H. Mulligan, "Pipelining of High Performance Synchronous Digital Systems," *International Journal of Electronics*, Vol. 70, No. 5, pp. 917–935, May 1991.

[40] E. G. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems*. IEEE Press, 1995.

[41] R. Jain, A. C. Parker, and N. Park, "Predicting System-Level Area and Delay for Pipelined and Nonpipelined Designs," *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 8, pp. 955–965, August 1992.

[42] E. G. Friedman, "Clock Distribution Design in VLSI Circuits — an Overview," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1475–1478, May 1993.

[43] E. G. Friedman, "The Application of Localized Clock Distribution Design to Improving the Performance of Retimed Sequential Circuits," *Proceedings of the IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 12–17, December 1992.

[44] K. A. Sakallah, T. N. Mudge, T. M. Burks, and E. S. Davidson, "Synchronization of Pipelines," *IEEE Transactions on Computer-Aided Design*, Vol. CAD–12, No. 8, pp. 1132–1146, August 1993.

[45] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Incorporating Interconnect, Register, and Clock Distribution Delays into the Retiming Process," *IEEE*

Transactions on Computer-Aided Design, Vol. 16, No. 1, pp. 105–120, January 1997.

[46] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and Resynthesis: Optimizing Sequential Networks with Combinatorial Techniques," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-10, No. 1, pp. 74–84, January 1991.

[47] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids,*. Holt, Rinehart and Winston, NewYork, 1976.

[48] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity,*. Prentice-Hall, Inc., 1982.

[49] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,*. McGraw-Hill, 1990.

[50] K. G. Murty, *Operations Research — Deterministic Optimization Models.* Prentice-Hall, 1995.

[51] J. R. Jump and S. R. Ahuja, "Effective Pipelining of Digital Systems," *IEEE Transactions on Computers*, Vol. C-27, No. 9, pp. 855–865, September 1978.

[52] T. C. Chen, "Parallelism, pipelining, and computer efficiency," *Computer Design*, Vol. 10, pp. 69–74, January 1971.

[53] T. G. Hallin and M. J. Flynn, "Pipelining of Arithmetic Functions," *IEEE Transactions on Computers*, pp. 880–886, August 1972.

[54] M. C. Papaefthymiou, "On Retiming Synchronous Circuitry and Mixed-Integer Optimization," Master's thesis, Massachussetts Institute of Technology, August 1990.

[55] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach,*. Morgan Kaufmann Publishers Inc., 1990.

[56] G. S. Tjaden and M. J. Flynn, "Detection and Parallel Execution of Independent Instructions," *IEEE Transactions on Computers*, Vol. C-19, No. 10, pp. 889–895, October 1970.

[57] E. M. Riseman and C. C. Foster, "The Inhibition of Potential Parallelism by Conditional Jumps," *IEEE Transactions on Computers*, Vol. C-21, No. 12, pp. 1405–1411, December 1972.

[58] G. Kane, *MIPS RISC Architecture,*. Prentice-Hall, 1988.

[59] INTEL Corporation, *The Intel Pentium*^{TM} *Processor. A Technical Overview*, 1994.

[60] S. R. Kunkel and J. E. Smith, "Optimal Pipelining in Supercomputers," *Proceedings of the Annual Symposium on Computer Architecture*, pp. 404–411, 1986.

[61] N. R. Strader II, "VLSI Bit-Sequential Architectures for Digital Signal Processing," *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing*, pp. 931–934, February 1987.

[62] P. R. Capello and K. Steiglitz, "Completely-Pipelined Architectures for Digital Signal Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-31, No. 4, pp. 1016–1023, August 1983.

[63] K. O. Siomalas and B. A. Bowen, "Synthesis of Efficient Pipelined Architectures for Implementing DSP Operations," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-33, No. 6, pp. 1499–1508, December 1985.

[64] P. R. Capello, A. LaPaugh, and K. Steiglitz, "Optimal Choice of Intermediate Latching to Maximize Throughput in VLSI Circuits," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 935–938, April 1983.

[65] N. V. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Minimum Padding to Satisfy Short Path Constraints," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 156–161, November 1993.

[66] M. C. Papaefthymiou, "Understanding Retiming through Maximum Average-Weight Cycles," *Proceedings of the Annual Symposium on Parallel Algorithms and Architectures*, pp. 338–348, July 1991.

[67] A. T. Ishii and C. E. Leiserson, "A Timing Analysis of Level-Clocked Circuitry," *Proceedings of the MIT Conference on Advanced Research in VLSI*, pp. 113–130, March 1990.

[68] 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*, Vol. CAD–12, No. 5, pp. 568–578, May 1993.

[69] N. V. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Resynthesis of Multi-Phase Pipelines," *Proceedings of the 30th Design Automation Conference*, pp. 490–496, June 1993.

[70] Z. Iqbal, M. Potkonjak, S. Dey, and A. Parker, "Critical Path Minimization Using Retiming and Algebraic Speed-Up," *Proceedings of the 30th Design Automation Conference*, pp. 573–577, June 1993.

[71] S. Dey, M. Potkonjak, and S. G. Rothweiler, "Performance Optimization of Sequential Circuits by Eliminating Retiming Bottlenecks," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 504–509, November 1992.

[72] T. G. Szymanski and N. Shenoy, "Verifying Clock Schedules," *Proceedings of the IEEE Conference on Computer-Aided Design*, pp. 124–131, 1992.

[73] E. G. Friedman, "Latching Characteristics of a CMOS Bistable Register," *IEEE Transactions on Circuits and Systems-I:Fundamental Theory and Applications*, Vol. CAS–40, No. 12, pp. 902–908, December 1993.

[74] S.-M. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits Analysis and Design*. Mc Graw Hill, 1996.

[75] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*. Addison Wesley, 1990.

[76] M. Shoji, "Elimination of Process-Dependent Clock Skew in CMOS VLSI," *IEEE Journal of Solid-State Circuits*, Vol. SC-21, No. 5, pp. 875–880, October 1986.

[77] J. Cong, A. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Performance-Driven Global Routing for Cell Based IC's," *Proceedings of the IEEE International Conference on Computer Design*, pp. 170–173, October 1991.

[78] T. Chao, Y. Hsu, J. Ho, K. D. Boese, and A. B. Kahng, "Zero Clock Skew Routing with Minimum Wirelength," *IEEE Transactions on Circuits and Systems — II: Analog and Digital Signal Processing*, Vol. CAS II–39, No. 11, pp. 799–814, November 1992.

[79] W. S. Scott and J. K. Ousterhout, "Magic's Circuit Extractor," *IEEE Design and Test of Computers*, Vol. 3, No. 1, pp. 24–34, February 1986.

[80] J. L. Neves and E. G. Friedman, "Circuit Synthesis of Clock Distribution Networks Based on Non-Zero Clock Skew," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 4.175–4.179, May/June 1994.

[81] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-2, No. 3, pp. 202–211, July 1983.

[82] T. Sakurai, "Approximation of Wiring Delay in MOSFET VLSI," *IEEE Journal of Solid-State Circuits*, Vol. SC-18, No. 4, pp. 418–426, August 1983.

[83] T. Sakurai, "Closed-Form Expressions for Interconnect Delay, Coupling, and Crosstalk in VLSI's," *IEEE Transactions on Electron Devices*, Vol. ED–40, No. 1, pp. 118–124, January 1993.

[84] C. Ramachandran and F. J. Kurdahi, "Combined Topological and Functionality Based Delay Estimation Using Layout-Driven Approach for High Level Applications," *Proceedings of the ACM/IEEE European Design Automation Conference*, pp. 72–78, September 1992.

[85] C. Ramachandran, F. J. Kurdahi, D. D. Gajski, A. C. H. Wu, and V. Chaiyakul, "Accurate Layout Area and Delay Modeling for System Level Design," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 355–361, November 1992.

[86] S. D. Rao and F. J. Kurdahi, "Hierarchical Design Space Exploration for a Class of Digital Systems," *IEEE Transactions on VLSI Systems*, Vol. VLSI-1, No. 3, pp. 282–295, September 1993.

[87] B. S. Cherkauer and E. G. Friedman, "A Unified Design Methodology for CMOS Tapered Buffers," *IEEE Journal of Solid State Circuits*, Vol. SC-30, No. 2, pp. 151–155, February 1995.

[88] B. S. Cherkauer and E. G. Friedman, "Design of Tapered Buffers with Local Interconnect Capacitance," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 3, No. 1, pp. 99–111, March 1995.

[89] M. Shoji, *CMOS Digital Circuit Technology,*. Prentice-Hall, 1988.

[90] A. I. Kayssi, K. A. Sakallah, and T. M. Burks, "Analytical Transient Response of CMOS Inverters," *IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications*, Vol. 39, No. 1, January 1992.

[91] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Monotonicity Constraints on Path Delays for Efficient Retiming with Localized Clock Skew and Variable Register Delay," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1748–1751, May 1995.

[92] J. L. Neves and E. G. Friedman, "Topological Design of Clock Distribution Networks Based on Non-Zero Clock Skew Specifications," *Proceedings of the IEEE Midwest Symposium on Circuits and Systems*, pp. 468–471, August 1993.

[93] J. L. Neves and E. G. Friedman, "Synthesizing Distributed Buffer Clock Trees for High Performance ASICs," *Proceedings of the IEEE ASIC Conference*, pp. 126–129, September 1994.

[94] J. L. Neves and E. G. Friedman, "Design Methodology for Synthesizing Clock Distribution Networks Exploiting Non-Zero Localized Clock Skew," *IEEE Transactions on VLSI Systems*, Vol. VLSI-4, No. 2, June 1996.

[95] S. H. Unger and Chung-Jen Tan, "Clocking Schemes for High-Speed Digital Systems," *IEEE Transactions on Computers*, Vol. C-35, No. 10, pp. 880–895, October 1986.

[96] D. B. Johnson, "Efficient Algorithms for Shortest Paths in Sparse Networks," *Journal of the Association for Computing Machinery*, Vol. 24, No. 1, pp. 1–13, 1977.

[97] R. Lisanke, "Logic Synthesis and Optimization Benchmarks User Guide: Version 2.0," Tech. Rep., Microelectronics Center of North Carolina, December 1988.

[98] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0," Tech. Rep., Microelectronics Center of North Carolina, January 1991.

[99] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C,*. Cambridge University Press, 1990.

[100] H. G. Martinez and T. W. Parks, "A Class of Infinite-Duration Impulse Response Digital Filters for Sampling Rate Reduction," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-27, No. 2, pp. 154–162, April 1979.

Publications

Journal Publication

- T. Soyata and E. G. Friedman, "Incorporating Interconnect, Register, and Clock Distribution Delays into the Retiming Process," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, Vol. CAD–16, No. 1, pp. 105–120, January 1997.

Conference Publications

- T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Monotonicity Constraints on Path Delays for Efficient Retiming with Localized Clock Skew and Variable Register Delay," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1748–1751, May 1995.
- T. Soyata and E. G. Friedman, "Retiming with Non-Zero Clock Skew, Variable Register, and Interconnect Delay," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 234–241, November 1994.
- T. Soyata and E. G. Friedman, "Synchronous Performance and Reliability Improvement in Pipelined ASICs," *Proceedings of the IEEE ASIC Conference*, pp. 383–390, September 1994.
- T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Integration of Clock Skew and Register Delays into a Retiming Algorithm," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1483–1486, May 1993.