

Copyright © 1993, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

TIMING ISSUES IN SEQUENTIAL CIRCUITS

by

Narendra Vasudeva Shenoy

Memorandum No. UCB/ERL M93/97

15 December 1993

TIMING ISSUES IN SEQUENTIAL CIRCUITS

Copyright © 1993

by

Narendra Vasudeva Shenoy

Memorandum No. UCB/ERL M93/97

15 December 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract**Timing Issues in Sequential Circuits**

by

Narendra Vasudeva Shenoy

Doctor of Philosophy in

Electrical Engineering and Computer Sciences

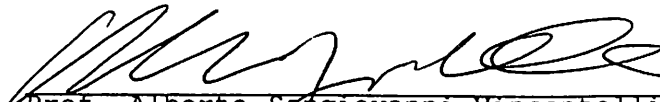
University of California at Berkeley

Professor Alberto Sangiovanni-Vincentelli, Chair

Design automation techniques play a major role in VLSI design. Growth in the complexity of circuits and performance requirements has necessitated the use of computer aided design tools. We examine some of the problems faced in the design of high performance synchronous circuits. Synchronous circuits use complex clocking schedules and circuit structures to capture and store data signals. The performance metric is measured by the periodicity of signals in the clocking schedule.

The first aspect of this thesis is devoted to the analysis of synchronous circuits. A clocking schedule must satisfy constraints that arise from the circuit topology and delay distribution on gates, wires and memory elements. Analysis to examine if a clock schedule is error-free is first considered. Improving the performance metric without changing the circuit topology the next issue considered. Flexibility in changing the clock signals and borrowing time across level-sensitive memory elements provides ample freedom for improving the circuit performance and should be exploited.

The second aspect focusses on performance improvement by transforming the circuit structure. A technique to use existing combinational delay optimizers repeatedly to solve the sequential performance problem is proposed. The approach uses an innovative notion of "perturbation" to extract timing constraints. If the difference between the largest and the smallest delays of paths between a pair of memory elements is significant, the short path can cause erroneous latching of data. This is known as the short path problem. We investigate this problem and propose a solution based on active delay insertion.



Prof. Alberto Sangiovanni-Vincentelli
Thesis Committee Chairman

... of the ...
... of the ...
... of the ...

The ...
The ...
The ...
The ...
The ...
The ...
The ...
The ...

The ...
The ...
The ...
The ...

The ...
The ...
The ...

To the memory of U. Vasudeva Shenoy

Contents

List of Figures	vii
List of Tables	ix
List of Theorems and Procedures	x
Acknowledgments	xii
1 Introduction	1
1.1 Synchronous circuits	1
1.2 Design process	5
1.3 Timing issues in VLSI design	6
1.4 Timing issues: an example	7
1.5 Thesis overview	13
2 Prelude	16
2.1 Circuit model	16
2.1.1 Combinational circuits	16
2.1.2 Memory elements	19
2.1.3 Clock signals	21
2.2 Timing in VLSI circuits: a review	21
2.2.1 Timing analysis	21
2.2.2 Timing optimization	23
2.3 Definitions	26
2.3.1 Clocking scheme	26
2.3.2 Combinational circuit	27
2.3.3 Memory elements and circuit clocking constraints	28
2.3.4 An example	31
2.4 Discussion	33
3 Clock Schedule Verification	35
3.1 Overview	35
3.2 Theoretical issues	36
3.2.1 Solving the late equation set	38

3.2.2	Solving the early equation set	40
3.3	Uniqueness of solutions	42
3.3.1	Uniqueness condition for late equation set	43
3.3.2	Uniqueness conditions for early equation set	47
3.3.3	Resolving multiplicity	48
3.4	Results	49
3.4.1	Benchmarks	51
3.4.2	Experiments	52
3.5	Discussion	53
4	Clock Schedule Optimization	55
4.1	Overview	55
4.2	Clocking constraints: a new form	56
4.3	Eliminating redundant constraints	63
4.4	Solving the optimization problem	66
4.4.1	A simple algorithm	66
4.4.2	A general algorithm	70
4.5	Results	75
4.5.1	An example	75
4.5.2	Experiments	80
4.6	Discussion	82
5	Resynthesis of Multi-Phase Pipelines	84
5.1	Overview	84
5.2	Definitions	85
5.3	Theoretical results	86
5.4	Resynthesis	90
5.5	The optimization problem	93
5.6	Cycle stealing	94
5.7	Results	98
5.7.1	An example	98
5.7.2	Experiments	101
5.8	Discussion	105
6	Delay Insertion for Short Paths	107
6.1	Overview	107
6.2	Definitions	108
6.3	Is padding always possible?	110
6.3.1	A naive algorithm	115
6.4	A linear programming approach	118
6.5	Refinements	120
6.5.1	Delay model	120
6.5.2	Discrete delay insertion	121
6.6	Relation to wave pipelining	124
6.7	Results	125

CONTENTS

vi

6.8 Discussion	129
7 Conclusions	130
A Optimality of the SmM Delay Model	132
A.1 Introduction	132
A.1.1 CmM delay model	134
A.1.2 SmM delay model	135
A.2 Relating the two models	136
A.2.1 Preliminaries	136
A.2.2 Equivalence of the two models	137
B Quadratic Optimization	141
Bibliography	147

List of Figures

1.1	A synchronous circuit	2
1.2	A synchronous pipeline	5
1.3	An example	8
1.4	High level specification: An example	9
1.5	Example with 2 phase level-sensitive latches	11
1.6	Alternative implementations: edge-triggered circuit	12
1.7	Alternative implementations: level-sensitive circuit	12
1.8	Performance versus area trade-off	14
2.1	Fixed delay model	18
2.2	Simplified min-max delay model	18
2.3	Statistical delay model	18
2.4	An ideal flip-flop	20
2.5	An ideal latch	20
2.6	Two phase clocking scheme	27
2.7	Key to waveforms	29
2.8	Data waveforms at a flip-flop	29
2.9	Data waveforms at a latch	30
2.10	Propagation of data waveforms	30
2.11	Set-up and hold constraints	31
2.12	Example: video coder	32
2.13	Latch graph for video coder	33
3.1	Circuit with multiple fixed points	43
3.2	Latch graph with modified edge weights	50
4.1	Graphical interpretation of the cycle weights	71
4.2	Graphical interpretation of optimality	74
4.3	Latch graph for video coder	75
4.4	Constraint graph for video coder	78
4.5	Constraint graph with edge weights evaluated at $c = 120$	78
4.6	Clocking scheme for video coder	79
4.7	Constraint graph for video coder with negative cycle	80

5.1	Multi-phase pipeline circuit	86
5.2	Graph construction	89
5.3	Effect of combinational optimization on long/short paths	91
5.4	Feasible region and current design	92
5.5	Graph modified for cycle stealing	94
5.6	k^{th} resynthesis Region	98
5.7	Pipeline example and associated graph	99
5.8	Graphical solution for example	100
5.9	Example for extended model	100
5.10	addT3- a 3 stage pipeline	104
6.1	Graph for a simple circuit	108
6.2	Short and long path interactions	113
6.3	Area optimization during delay insertions	123
6.4	Wave pipelining	124
A.1	Robust clocking: an example	136
A.2	Skew effect: an example	138
B.1	Plot of \mathcal{L} versus α	145

List of Tables

2.1	Variables at a memory element	28
2.2	Clocking constraints	32
2.3	Clocking issues	34
3.1	Simplified clocking constraints	36
3.2	Table of iterations - late equation set	49
3.3	Table of iterations - early equation set	50
3.4	Benchmark statistics	52
3.5	Clock verification with unit fanout delay model	52
3.6	Clock verification with library delay model	53
4.1	Conservative clocking constraints	57
4.2	Inequalities for correct clocking	58
4.3	Optimal clock computation with unit delay fanout model	81
4.4	Optimal clock computation with library delay model	82
5.1	Arrival and required times for resynthesis	102
5.2	Area-clock period trade-off	103
5.3	Pipeline resynthesis for minimum clock period using unit fanout model	104
5.4	Pipeline resynthesis for addTn	105
6.1	Delay insertion using unit delay fanout model	127
6.2	Delay insertion using library delay model	127
6.3	Wave pipelining using unit delay fanout model	128
6.4	Wave pipelining using library delay model	128
B.1	Iterations for quadratic programming	146

List of Theorems and Procedures

Lemma 3.2.1	37
Corollary 3.2.2	37
Lemma 3.2.3	37
Proposition 3.2.4	37
Corollary 3.2.5	37
Procedure 3.2.1	38
Lemma 3.2.6	38
Lemma 3.2.7	39
Theorem 3.2.8	39
Procedure 3.2.2	40
Lemma 3.2.9	41
Lemma 3.2.10	41
Theorem 3.2.11	42
Lemma 3.3.1	43
Lemma 3.3.2	44
Lemma 3.3.3	44
Lemma 3.3.4	45
Theorem 3.3.5	45
Lemma 3.3.6	46
Lemma 3.3.7	46
Lemma 3.3.8	46
Lemma 3.3.9	47
Theorem 3.3.10	47
Lemma 3.3.11	47
Lemma 3.3.12	47
Theorem 3.3.13	48
Property 4.2.1	58
Lemma 4.2.1	58
Corollary 4.2.2	59
Theorem 4.2.3	59
Procedure 4.2.1	61
Theorem 4.4.1	67
Theorem 4.4.2	68
Theorem 4.4.3	68

Lemma 4.4.4	69
Procedure 4.4.1	70
Lemma 4.4.5	72
Lemma 4.4.6	72
Lemma 4.4.7	73
Theorem 4.4.8	73
Corollary 4.4.9	74
Theorem 5.3.1	88
Lemma 5.3.2	90
Theorem 5.5.1	93
Lemma 5.6.1	95
Proposition 5.6.2	98
Lemma 6.3.1	110
Theorem 6.3.2	111
Lemma 6.3.3	111
Corollary 6.3.4	114
Corollary 6.3.5	115
Procedure 6.3.1	115
Lemma 6.3.6	116
Corollary 6.3.7	117
Lemma 6.3.8	117
Corollary 6.3.9	117
Corollary 6.3.10	117
Procedure 6.3.2	118
Theorem 6.4.1	119
Theorem 6.5.1	120
Theorem 6.5.2	121
Lemma A.2.1	138
Proposition A.2.2	139
Lemma A.2.3	139
Theorem A.2.4	140
Corollary A.2.5	140
Lemma B.1.6	141
Lemma B.1.7	141
Lemma B.1.8	142
Lemma B.1.9	142
Lemma B.1.10	143
Procedure B.1.1	143

Acknowledgments

It has been a long and winding 19 years since I ambitiously embarked on the project of educating myself. This thesis to a certain extent reflects a culmination of these efforts. It is hoped that writing this thesis has made me realize what Bacon meant when he said "Reading maketh a full man, conference a ready man, and writing an exact man".

Prof. Alberto Sangiovanni-Vincentelli, my advisor, teacher, mentor all rolled into one; from you I learnt the art of delivering coherent lectures, the need for meticulousness in research and the skill of technical writing. Thanks also for the financial and moral support over the years. Prof. Robert Brayton has been much more than a chairperson on my qualifying committee and co-advisor. He has guided this research much like his own project, listened patiently to proofs and gibberish for hours, and encouraged me to find my own bearings.

Dr. T. G. Szymanski has been an illuminati in the research area of this thesis. His guiding hand can be seen in Chapters 3 and 4, and Appendix A. Inspiration from your research, discussions and support certainly jump-started this thesis. Thanks Tom, I wish there was some way I could repay you for your ever-willingness to guide me. You have been my advisor away from school. I also wish to thank Prof. J. Rabaey and Prof. S. Oren for agreeing to be committee members on my qualifying exam. Prof. S. Oren, I also thank you for taking the time to be the external reader for my thesis.

To Rajesh and Anand; your presence has made the journey of my life much more enjoyable. There are few things in life that I cherish more than your friendship. I would also like to take a moment to wish you both jugfulls of felicitations for linking your lots with Bhavana and Shevani. I also take this opportunity to thank a few teachers from St. Xaviers' High School, Bombay; Ms. Joannes, Ms. Carvalho, Mrs. Seshan, Ms. Ghadiali, Mrs. Gandhi, Mr. Rafael, and Fr. Aran. I hope we can live up to what you taught us— "duc in altum"! Certain people at the Indian Institute of Technology, Bombay did believe that I was cut out for research and encouraged me to that effect.

Thanks to Prof. A. N. Chandorkar, Prof. J. Vasi, and Prof. U. B. Desai. If one man convinced me to apply to Berkeley it was a Makarand; thanks Mac for your advice and friendship over the years.

The Bay area has been a very hospitable place to live in. But the presence of Madhu, Nirranjan, Diane and Sushil, Savita and Munnu, lil' Usha, KJ, Paola and Luciano, Rajeev, Asha and Mots, Sajeena and Sushil has made it even more fun. And yes a extra-special thanks to Pratap; for the MNF sessions, for willingly volunteering to eat experimental cooking and sipping untested cocktails and among many many other things being a friend. I also wish to thank Manisha and Milind, Vidya and Ravi, Sashikala and Narasimha for being wonderful hosts on several occasions.

The Cad-group provides an excellent environment for research and a wonderful support group. It would be an Herculean effort to thank everyone (past and present), but I cannot but list Luciano Lavagno, Sharad Malik, Rajeev Murgai, Alexander Saldanha, Ellen Sentovich and Kanwar Jit Singh. Flora, Elise, Kia, Heather and Genevieve have always been willing to help on the organizational side of things. Thanks for their patience and help. Kudos to Brad and Mike for the excellent support in the Cadgroup. NSF and DARPA are acknowledged for funding this research. The Cadgroup also had its fair share of visitors. Many thanks and regards to K. Kodandapani, Paul Gutwin, and Masamichi Kawarabayashi. I also wish to thank Kurt Keutzer for several basketball games during post-conference hours.

There are numerous "faces" at the RSF to thank for pick-up basketball games. And yes Bill Watterson, for a wonderful pair of cartoon characters; to Calvin for his indispensable advice and Hobbes for his sense of propriety. To the writers of Superman and Batman series of comics; you provided the means of escape to a little kid in me that still wants to believe in super heroes and legends. To Sam and his merry crew (over the years) at Brewed Awakening (Coffee Connection not so long ago) who provided wonderful service that made the 4 o'clock coffee break a pleasure.

To my mother, and brother Vasant; the sacrifices you have made to support me cannot be described. This thesis is the result of as much effort (if not more) on your parts than mine. I am also grateful to my uncles Vithal, Umanath and Ramachandra and my aunts who provided comfort and courage to go on with my studies. A special thanks to Surekha and Surendra and their families. And yes the Lord Almighty (I agree with the philosophy that it is better to thank God intermittently than to spend your life as an atheist, only to discover that He does exist!).

Chapter 1

Introduction

The impact of Very Large Scale Integrated (VLSI) circuits in modern life can be seen in various electronic facilities with which we pamper ourselves. VLSI circuits can be found in modems, facsimile machines, television sets — to mention a few. Over the years, growth in the complexity of VLSI designs has enabled designers to include well over a million transistors on each chip. Designers are faced with the daunting task of packing more functionality into a smaller area and creating a circuit that operates faster than the previous generation. Design Automation (DA) techniques play an invaluable role in this complex process.

This thesis deals with some of the problems that arise in *synchronous circuit* design. Algorithms for ensuring correct operation of a circuit and for performance optimization of a circuit are presented. Implementation issues and experience gained through experiments are also described.

Section 1.1 provides an insight to some of the issues that arise in the design of a synchronous circuit. The advantages and disadvantages of a synchronous design style are also described. The design process is outlined in Section 1.2. Section 1.3 presents some of the timing issues that arise in VLSI design. The research in this thesis is motivated by an application in Section 1.4.

1.1 Synchronous circuits

VLSI circuits can be broadly classified into two categories depending on their mode of operation.

- A synchronous circuit is characterized by the presence of special *periodic* signals (called *clock signals* or *phases*) and special circuit components (called *memory elements*) that are

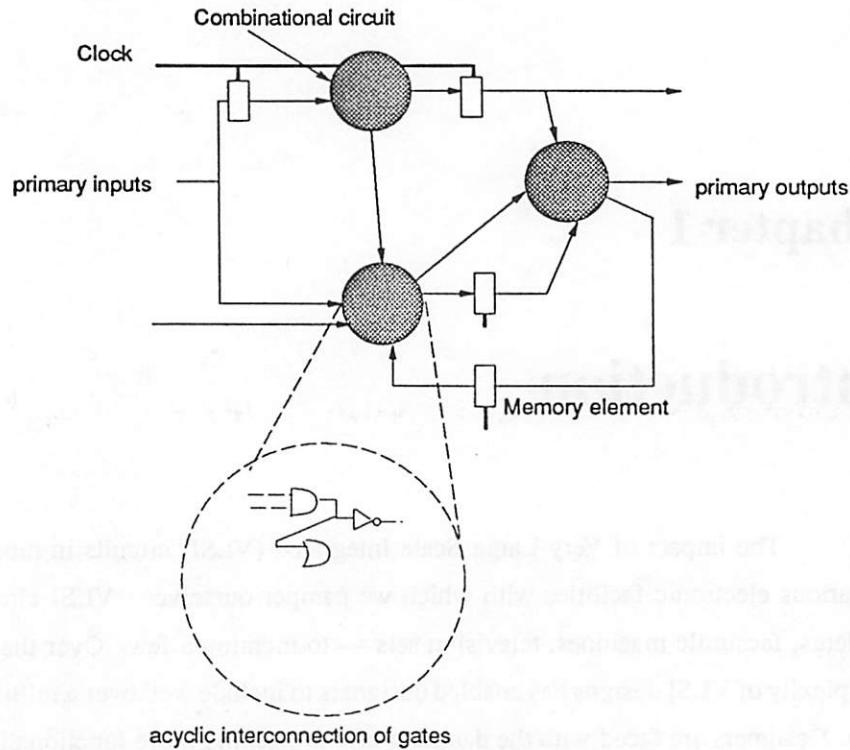


Figure 1.1: A synchronous circuit

used to store and regulate the flow of data. There can be one or more clock signals, operating at the same or different periods (frequencies). A rise or fall of a clock phase constitutes a clock event. A clock schedule is a set of clock signals with all clock events specified. A performance metric for a synchronous circuit is the smallest period common to the clock signals, called the **clock period**. Smaller the clock period, higher is the regard for the design. A synchronous circuit is an interconnection of gates and memory elements. It may be partitioned into regions of combinational logic separated by memory elements (Figure 1.1). Each combinational region is an acyclic interconnection of gates. Thus every cycle in the circuit is broken by at least one memory element. The flow of data between combinational regions is periodically regulated by clock signals controlling the intervening memory elements. The ratio of the time interval that a phase is high to the clock period is called the **duty cycle** for the phase. If all the phases have the same duty cycle, the clock schedule is said to have the said duty cycle.

- An **asynchronous circuit** is an arbitrary interconnection of gates and latching structures. In contrast to synchronous circuits, cyclic structures of logic gates are permitted and there are

no clock signals dedicated for latching data.

A major challenge in synthesizing asynchronous circuits is the susceptibility of designs to *hazards*. A hazard is defined as any transition of a signal that is not prescribed by the designer. The causes for a hazard can be traced to

1. the distribution of delays on gates in the circuit, or
2. the logic function implemented by the design.

The actual delay of a gate depends on the fabrication process and on factors related to the operating environment (like temperature, exposure to radiation etc.). This makes it difficult for a designer to ensure that a circuit is hazard-free. Automated approaches to synthesize hazard-free designs have been recently proposed [47, 7, 31]. This problem is mitigated in synchronous designs due of the presence of memory elements. Hazards can appear in a combinational region, but the clock schedule is designed so that the flow of hazards through memory elements into adjacent combinational regions, is prevented or carefully controlled.

During the design of a synchronous circuit special attention must be paid to the routing of clock lines. Clock signals have to be distributed from input pads (sources) to memory elements (sinks) on a chip. Signal propagation through long metal lines leads to degradation of the signal. Consequently a signal at a memory element may not have sharp transitions, although the same signal has sharp transitions at its source. This gives rise to several problems; two prominent issues are the following.

1. At a single sink, the asymmetry of rise and fall times along the clock distribution path may lead to a narrowing of the clock pulse at the sink.
2. Now consider two clock lines reaching a pair of sinks, such that data from the output of a memory element at the first sink propagates to the input of a memory element at the second sink. Clock signals that are not overlapping at the source pins, may appear overlapping at the memory elements. This can cause a memory element to permit flow of data, when in fact it should have impeded it, or vice versa.

These problems are attributed to clock skew. However, recent advances in physical DA have provided techniques to overcome the clock skew problem [74, 34]. Despite the problems associated with uncontrolled skew, efforts have been also made to control clock skew to the designer's advantage

[15]. Asynchronous circuits are free from clock skew problems, since there are no clock signals to be routed across the chip.

An important issue in VLSI design is the power dissipation per unit area of the design. The pads of a design typically account for more than 50 per cent of the power dissipation. Rest of the power is dissipated in evaluation of logic in combinational regions, in driving clock lines high and low and due to leakage currents. In an asynchronous circuit, the power is dissipated in switching of gates and in additional circuitry required to detect signal completion. Power dissipation can be classified as

1. quiescent; caused due to leakage currents, dominant in portable applications, and
2. dynamic; caused due to switching activity, dominant in high performance computation oriented designs.

The power dissipation depends to a large extent on the choice of technology (CMOS, ECL, BiCMOS etc.) and also on the design style.

The advantages of designing a synchronous circuit are:

- Hazards occurring internally in the circuit do not affect the outputs.
- Modularity of combinational regions due to the relative isolation provided by the intervening memory elements.
- Ease of testing. Techniques such as SCAN [14] can be used to test and detect errors in a fabricated chip.
- Relative insensitivity to actual gate delays. If gate delays violate respective bounds, the circuit may not operate at the desired frequency. However, it may be possible to operate it at a lower frequency.

The drawbacks of synchronous designs are:

- Increase in area, due to the presence of memory elements and routing of clock signals.
- Care has to be taken in distributing clock signals across the design.

A special class of synchronous designs are pipeline circuits, in which data flows in one direction. They are also termed as flow-forward circuits. A multi-phase pipeline consists of stages of combinational logic separated by memory elements (see Figure 1.2). Each stage has inputs from

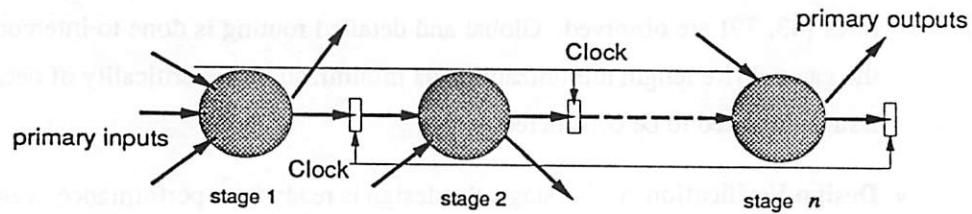


Figure 1.2: A synchronous pipeline

the previous stage and perhaps some inputs from the external world. Each stage has outputs feeding the next stage and perhaps also to the environment. Data path designs are typically pipelines. The depth or level of a pipeline is the maximum number of memory elements along a path from an input to an output.

1.2 Design process

Structured VLSI design proceeds through several steps. They are briefly outlined below.

- **Design Specification** In the first step, a formal behavior of the design is specified. This translates a conceptual idea into a description that may be verified for various properties that must be satisfied by the design. A large design may be sub-divided into several modules that may be independently designed and “glued” together at a later stage. This requires budgeting of resources (like chip area, power dissipation) and timing constraints on the independent modules.
- **Logic Design** The formal specification is translated into a set of Boolean equations and registers. Logic synthesis techniques are used to optimize the circuit for area, for speed, or for power. At this stage, since the circuit has no tangible realization associated with it, it is called an *unmapped* circuit. *Technology mapping* is a process by which the circuit is associated with an interconnection of gates and memory elements from a library, depending on the technology of implementation. The resulting net-list is called a *mapped* circuit.
- **Physical Design** Gates and memory elements in the net-list are given their physical attributes. The circuit is represented as a set of polygons on several layers. A placement tool is used to find locations for the polygons on a two dimensional plane. The goal is to minimize the total area and also to take the criticality of the nets into account, while ensuring that design

rules [43, 79] are observed. Global and detailed routing is done to interconnect the pins of the gates. Wire length minimization, via minimization and criticality of nets are some of the issues that need to be considered.

- **Design Verification** At this stage, the design is ready for a performance evaluation. A timing verification is done to ensure that the memory elements latch data signals correctly and that hazards are not permitted to race around the circuit, corrupting computed values. A functional verification is carried out to ensure correct logical behavior of the circuit under all possible environment inputs. Circuit simulation may also be used to compute the delays of critical paths. Rectification of an error discovered in this step, may mean repeating one or more of the previous steps.
- **Performance Issues** Chip design often involves a complex trade-off of resources for requirements. High performance is typically attained at a penalty in area and power. At every level of the design it is necessary to evaluate various strategies available to meet the goals. Problems that arise in performance analysis and optimization of synchronous circuits are discussed in detail in the following Section.

1.3 Timing issues in VLSI design

Due to the physical nature of gates and memory elements, signals undergo a delay when propagating through them. Consequently, the computation of a Boolean function takes a finite amount of time. A path in a circuit is called a **critical path** if delays of gates on the path prevent the circuit from operating faster. Path length is used to denote the delay incurred by a signal propagating along the path. For a combinational circuit, with all inputs arriving at the same time, the longest paths in the circuit form a set of critical paths. The simplest timing problem that arises in circuit design is known as **timing analysis**. Informally, timing analysis can be defined as a procedure which identifies critical paths. Sequential timing analysis is complicated by the fact that memory elements latch data. A sequential timing analyzer identifies critical paths from one memory element to another and ensures the stability of data signals when memory elements latch data. The constraints for correct operation of a circuit are termed as *clocking constraints*. Timing analysis procedures operate on a gate level description or a transistor level description of a circuit. A timing analysis tool is termed as a **static analyzer** if it does not take the Boolean nature of gates into account. Results from static timing analysis are always pessimistic. There may exist topological

paths in a circuit which are never exercised during operation, and thus can never contribute to the delay of the circuit. Such a path is termed a **false path** - a path through which no transition of signals can propagate. A timing analysis procedure is termed **dynamic** if it can detect false paths and identify the delay of true paths.

Often, the design of a combinational region has to meet constraints on the largest and smallest delay that any path in the region can have. This is known as the *combinational performance* problem. Efforts to solve this problem have used critical path restructuring, gate decomposition, buffer optimization and transistor sizing. All previous methods have dealt with controlling the longest delay of the circuit. The problem of ensuring that path delays exceed their respective lower bound has not been addressed so far. Combinational performance optimization techniques are important for obtaining fast implementations, but they form only a subset of the techniques available to the designer. The flexibility provided by the presence of memory elements and clock signals is exploited by sequential optimization techniques. To evaluate a synchronous circuit for performance, it is necessary to compute a clock schedule with the smallest clock period that satisfies all the system constraints and the clocking constraints. This is known as the clock schedule optimization problem. Retiming is a technique which involves repositioning of memory elements so that the clock period is decreased, while preserving input-output behavior. It is common for the designer to have a target clock period as the goal for circuit performance. In such a case, it is possible to specify timing constraints for each combinational region, in order that the target clock period be attained. Each combinational region is resynthesized to ensure that its delays are within the prescribed bounds.

1.4 Timing issues: an example

Consider the design of the following arithmetic circuit; it takes a 3 bit input vector A , a 2 bit non-zero input key C , and provides a 2 bit output vector S . The inputs A and C are held constant upto C clock periods. Let us ignore the circuitry which controls the application of the inputs and instead focus on the data-path. The behavior of a sequential circuit is described by annotating each input/output vector with a integer subscript; the subscript indicates the clock period frame with which the vector is to be associated. If the vector is constant over all $C - 1$ periods, the subscript is dropped. The output of the circuit is specified in terms of the input and clock period

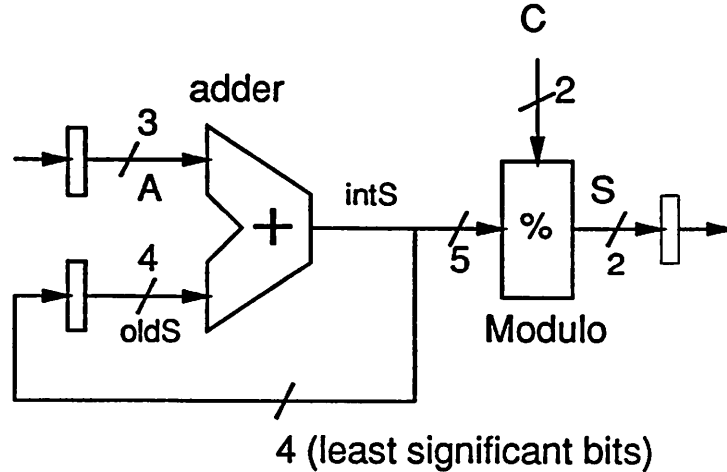


Figure 1.3: An example

frame under consideration.

$$S_k = (kA)\%C \quad k = 1, \dots, C - 1 \quad (1.1)$$

The modulus (%) operation yields the remainder after dividing the term before the operand by the term after the operand. The circuit is implemented in terms of an adder and some combinational logic for the modulus operation. Since the maximum value of the input key C is 3 and the value of A is at most 7, the size of the adder can be restricted to 5 bits at the output. The schematic is shown in Figure 1.3. As a matter of notation, the memory elements are shown in little rectangles with the data input as an arrow leading to it and the data output as an arrow leaving it. The phase controlling it may be omitted if all memory elements are clocked by the same phase (single phase clocking scheme). The high level description of the circuit in BDS [59] is given in Figure 1.4. The BDS description is translated to a logic level description and memory elements are inserted. A sequential logic optimizer SIS[60], is used to obtain a logic circuit which has not been bound to a technology. At this stage the designer is interested in exploring the design space for a performance versus area trade-off.

The area of the circuit is measured as the sum of the area of the logic and the memory elements. The logic is decomposed into 2 input *and* and *or* gates, and *inverters*. The area of the logic is approximated as the number of such gates. Each gate is assigned a unit delay with 0.2 units per fanout. A memory element has three ports, a data input port, a data output port and a control input port. There may also be ports for setting and resetting the memory elements, or for

```

MODEL add-modulo
  S<1:0>, intSo<4:0>=                ! current sum
  A<2:0>,                            ! 2 bit input vector
  C<1:0>,                            ! 2 bit non-zero input vector
  oldS<3:0>,                        ! previous sum
  intSi<4:0>;

  ROUTINE adder;
    intSo = oldS + A;
  ENDROUTINE;

  ROUTINE modulo;
    S<1:0> = 0;
    SELECT C FROM
      [1]: BEGIN
        S<1:0> = 0;                ! modulo 1 is always 0
      END;
      [2]: BEGIN
        S<0> = intSi<0>; ! modulo 2 is the least significant bit
      END;
      [3]: BEGIN
        SELECT intSi<4:0> FROM
          [1, 4, 7, 10, 13, 16, 19]: BEGIN
            S<1:0> = 1;          ! modulo 3 for this set is 1
          END;
          [2, 5, 8, 11, 14, 17, 20]: BEGIN
            S<1:0> = 2;          ! modulo 3 for this set is 2
          END;
        ENDSELECT;          ! modulo 3 for rest is 0 (default)
      END;
    ENDSELECT;
  ENDROUTINE;
ENDMODEL;

```

Figure 1.4: High level specification: An example

asynchronous events but we shall ignore these for the time being. Although there are several types of memory elements, they can be broadly classified into two categories on the basis of behavior.

1. **Edge-triggered memory element (flip-flop):** The memory element samples the input port at the rise (or fall) of the control signal, and provides the value at the output. The output is held stable until the next rising (or falling) edge of the control signal.
2. **Level-sensitive memory element (latch):** The memory element transmits the data at the input to the output, as soon as the control signal goes high (or low) and continues to do so as long as the control remains high (or low). When the control falls (or rises), the output is “latched” to the value at the instant of fall (or rise) and is held at that value until the next rise (or fall) of the control signal.

To summarize, the flip-flops provide better isolation between input and output than latches. Latches permit combinational regions to borrow (steal) time from adjacent regions during the active intervals.

To begin, consider the circuit with flip-flops triggered on the falling edge. Each flip-flop is assigned an area equivalent to six 2 input gates. The area is measured in terms of the total number of 2 input gates in the circuit. The initial area is 159 and a timing analysis yields the best clock to be 30.6 units. Retiming for minimum delay yields a clock period of 10.8 units and an area of 243. Combinational performance optimization techniques on the original circuit result in a circuit with clock period 24.0 units and an area of 209.

The design is changed so that the flip-flops at the inputs and outputs are replaced by level-sensitive (active high) latches on phase ϕ_1 and level-sensitive latches (active-high) on phase ϕ_2 are introduced at the output of the adder (see Figure 1.5). A level-sensitive latch has an area equivalent to three 2 input gates. The circuit area is now 147. We force a duty cycle of 0.3 for the clock. The optimal clock schedule is

$$\text{rise of phase } \phi_1 = 10.9$$

$$\text{fall of phase } \phi_1 = 18.0$$

$$\text{rise of phase } \phi_2 = 16.6$$

$$\text{fall of phase } \phi_2 = 23.7.$$

Combinational performance optimization results in the following clock schedule,

$$\text{rise of phase } \phi_1 = 7.1$$

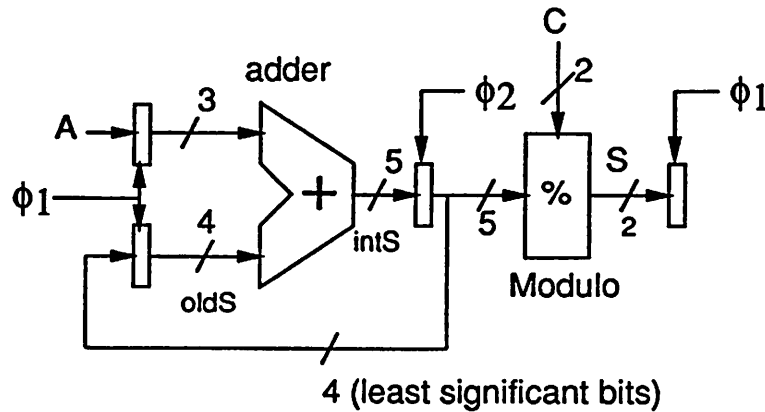


Figure 1.5: Example with 2 phase level-sensitive latches

fall of phase $\phi_1 = 13.2$

rise of phase $\phi_2 = 14.2$

fall of phase $\phi_2 = 20.3,$

at an area of 136 (the area decreases!).

One possible optimization that has not been explored so far involves using associativity and commutativity of arithmetic operations. Consider the following function defined recursively

$$f_k = \begin{cases} (A + f_{k-1})\%C & k > 1 \\ A\%C & k = 1. \end{cases}$$

We will show $f_k = S_k$, for all k by induction. Recall that $S_k = (kA)\%C$ from Equation 1.1. It is easy to verify that $f_1 = S_1$. Assume $f_k = S_k$ for all $k \leq n$; it remains to show $f_{n+1} = S_{n+1}$. By a process of simple substitution, we get the result as follows:

$$\begin{aligned} f_{n+1} &= (A + f_n)\%C \\ &= (A + S_n)\%C \\ &= (A + (nA)\%C)\%C \\ &= (A\%C + ((nA)\%C)\%C)\%C \\ &= (A\%C + (nA)\%C)\%C \\ &= (A + nA)\%C \\ &= ((n + 1)A)\%C \\ &= S_{n+1}. \end{aligned}$$

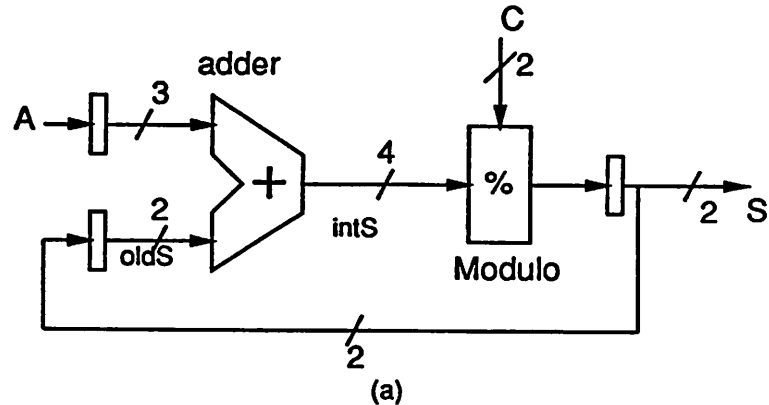


Figure 1.6: Alternative implementations: edge-triggered circuit

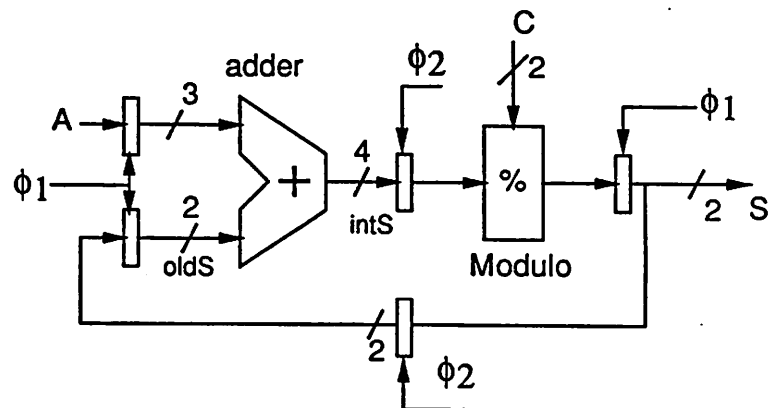


Figure 1.7: Alternative implementations: level-sensitive circuit

Thus we could also choose to implement the circuit using flip-flops as shown in Figure 1.6 or using level-sensitive latches as shown in Figure 1.7. This has the advantage of reducing the feedback lines to 2, thus saving area. None of the optimization techniques at the logic level of a circuit are able to detect this transformation. The transformation takes advantage of the fact that the modulo operation with respect to an integer C , can be used to construct a *group*[3] known as the *modulo C group*. The mathematical properties of a group are exploited to obtain a circuit with different structure but identical behavior. The edge-triggered version has an area of 95 and a clock period of 23.4 units. It can be retimed to yield a clock period of 12.4 units with an area of 107. Optimizing the combinational regions gives a circuit with an area of 117 and clock period of 12.2 units. The level-sensitive version of the circuit has an area of 90 and has the following clocking scheme (duty

cycle set to 0.3),

$$\begin{aligned}\text{rise of phase } \phi_1 &= 7.9 \\ \text{fall of phase } \phi_1 &= 13.8 \\ \text{rise of phase } \phi_2 &= 13.8 \\ \text{fall of phase } \phi_2 &= 19.7.\end{aligned}$$

Combinational optimization gives a circuit with area 97 and the following clock schedule (duty cycle set to 0.3),

$$\begin{aligned}\text{rise of phase } \phi_1 &= 5.6 \\ \text{fall of phase } \phi_1 &= 9.8 \\ \text{rise of phase } \phi_2 &= 9.8 \\ \text{fall of phase } \phi_2 &= 14.0.\end{aligned}$$

Figure 1.8 summarizes some of the choices that the designer has. Each point on the clock period-area graph has the figure number of the circuit associated with it. A trailing R or C implies that the circuit was retimed or underwent combinational optimization. It is quite clear from the above example, that DA techniques for sequential synthesis must focus on two broad areas, which overlap considerably. On one hand, tools must be designed to examine a given circuit for correctness and performance. These are techniques of *analysis*. The other area must focus on transformations that provide alternative implementations with the same input-output behavior. These are termed techniques of *synthesis*. Thus, the goal is to develop an all powerful sequential optimizer, that combines the approaches from synthesis and analysis, and is sentient to the various design trade-offs. This thesis does not promise the “sentient” sequential optimizer, but develops the basic building blocks that would form corner stones of such a system and abet its development.

1.5 Thesis overview

Four interesting problems in timing analysis and optimization have been selected to be included in this thesis. The first three deal with the sequential nature of a circuit. The last problem arises in the synthesis of combinational circuits that have to operate in a sequential environment.

A major issue in the design and implementation of algorithms for design automation, is the modeling of circuit behavior. A synchronous circuit has three essential components; logic gates,

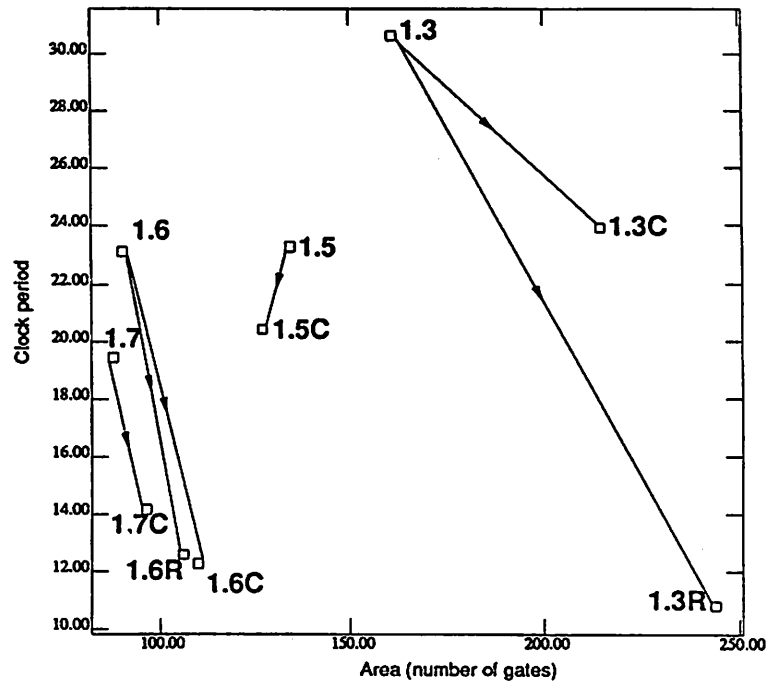


Figure 1.8: Performance versus area trade-off

Data point associated with Figure describing circuit

R - result of retiming

C - result of combinational optimization

memory elements and clocking schemes. The mathematical models for a synchronous circuit form the bulk of Chapter 2. Chapter 2 also provides a comprehensive review of the previous work in timing issues.

The thesis can be broadly divided into two parts.

- The first part focusses on analyzing a given circuit. Chapter 3 describes the *clock verification* problem and presents an algorithm of polynomial complexity to solve it. The *clock schedule optimization* problem is solved in Chapter 4.
- The second part deals with optimization targeted for sequential behavior. Chapter 5 is devoted to pipeline circuits. This Chapter explores the problem of extracting timing constraints for combinational regions, so that the entire circuit operates at a specified clock period. Chapter 6 poses a problem in combinational logic synthesis that has hitherto been unexplored; namely the problem of ensuring that lower bounds on paths in combinational circuits are met. This is relevant for the synthesis of sequential circuits with level-sensitive latches. As an aside, an application to wave pipeline circuits will also be described.

The optimality of the delay model used for the clock schedule optimization problem is discussed in appendix A. Appendix B deals with the development of a quadratic programming algorithm which is used in Chapter 5.

Chapter 2

Prelude

It was nearly two decades ago that the necessity of automated timing analysis was felt. Since then various research efforts have investigated the problem using different delay models and proposed several techniques. We hasten to point that the efficiency of an algorithm and the verisimilitude of the model (to reality) are often antipodal. It is important to develop an efficient algorithm with as realistic a delay model as possible. This Chapter serves three purposes: the first is to introduce the reader to different models present in literature, the second is to provide a brief review of various approaches and lastly, to present the model and definitions that will be used in the remainder of this thesis.

Section 2.1 describes the three components of a synchronous circuit. A review of research efforts in the field of timing issues of synchronous circuits is presented in Section 2.2. Section 2.3 gives the specific model that is used in this thesis.

2.1 Circuit model

A synchronous circuit is modeled as an interconnection of gates and memory elements. The gates may be partitioned into combinational regions isolated by memory elements. A circuit model is composed of models for combinational circuits, memory elements and clock signals.

2.1.1 Combinational circuits

The combinational circuit is an acyclic interconnection of gates. An input to a combinational circuit is called a **primary input** if it is a signal provided by the environment. A combinational

circuit may also have inputs driven by the outputs of memory elements. Similarly, an output of a combinational region may drive an input of a memory element or be fed to the environment. In the latter case, it is called a **primary output**. The delay information of each gate is “composed” to yield the delay information for input-output pairs.

Gate delays

The delay of a gate is an attribute of the physical process of charging a capacitor. In the case of CMOS circuits, the capacitance at the output (and the source/drain to substrate capacitances to a certain extent) prevents the output voltage from switching instantaneously. Similarly in a BJT, junction capacitances are responsible for the transition delay. We only consider gates with single outputs in this thesis. The following are popular delay models for gates.

Fixed Delay A gate is assumed to have a constant delay, known *a priori*, and represented by a real number. This model has been used in [33, 37]. A simple fixed delay model is the **unit delay model**, wherein each gate is assigned a delay of 1 unit. A refinement is the **linear delay model**; the delay of a gate is described by a linear function $\alpha + \beta\gamma$, where α is the intrinsic delay of the gate, β models the load dependent delay and γ is the capacitive load at the output of the gate. A simple example is the **unit delay fanout model**, where $\alpha = 1.0$, $\beta = 0.2$, and γ is the number of fanouts of the gate. For a library of standard cells, the values for α and β are computed using regression analysis on several simulations (using a circuit simulator like SPICE[46]). The linear delay model for a library is also referred to as the **library delay model**. Most combinational optimization techniques have relied on the linear delay model. These models are good only when factors that determine the actual delay of a gate; such as the fabrication process and the operating environment conditions, can be tightly controlled.

Figure 2.1 shows the waveforms at the inputs to a 2 input *and* gate and the corresponding waveform at the output using a fixed delay model with the given parameters.

Min-max delay The delay of each gate is assumed to take on a value between a lower bound and an upper bound. The **simplified min-max delay model**, abbreviated to SmM, assumes that the delay of a gate takes its worst case value. In other words, the interval of uncertainty of the gate output in response to input transitions is made as wide as possible. This is a conservative albeit pessimistic approach. The **consistent min-max delay model**, denoted by CmM for short, assumes that the delay of a gate is a variable that lies between the upper and lower bounds. Assignment of a delay value to a gate is deferred until the verification/optimization procedure. The important point

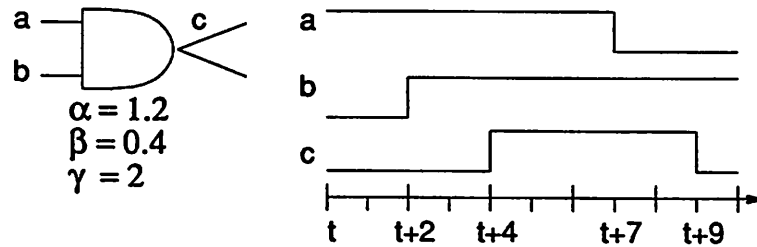


Figure 2.1: Fixed delay model

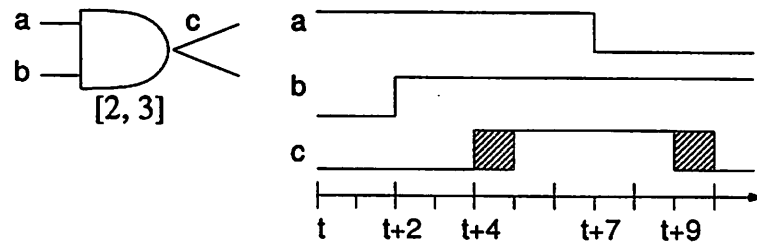


Figure 2.2: Simplified min-max delay model

is that each gate is assigned the same symbolic delay. In Figure 2.2, the input and output waveforms under the SmM delay model are shown. Note there is an interval of uncertainty of unit length at the output during the output rise (from $t + 4$ to $t + 5$) and output fall (from $t + 9$ to $t + 10$).

Statistical delay A probability distribution function (*pdf*) for the delay of a gate is described. Very often a Gaussian distribution, specified by its mean and its standard deviation, is used. An example is shown in Figure 2.3. The probability that the output is high at time $t + 3$ is 0 and as time passes, it increases to 1 at $t + 5$. Similarly, the probability that the output is 0 at time $t + 8$ is 0 and increases to 1 by time $t + 10$. This assumes that the delay of each gate is an independent random variable. In reality, there is a strong correlation between the delays of gates

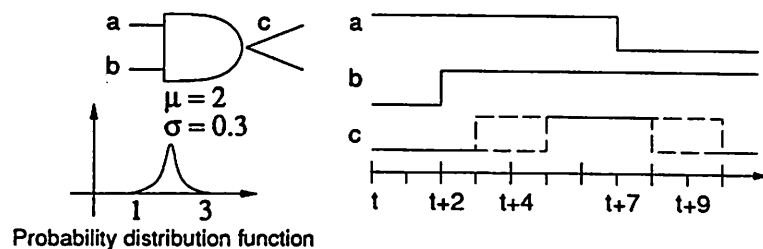


Figure 2.3: Statistical delay model

in a design. A cause for this fact is that the fabrication process tends to bias all gate delays in the same direction; *e.g.* a thin oxide deposition process may be terminated earlier than required — resulting in all transistors having gate oxides of thickness less than the expected value. To capture this correlation, the delay of gate is modeled as a sum of two components; the first component is determined by a random variable that reflects independent variations in the delay of each gate. The second component is determined by a random variable which is the same for all gates on a chip and reflects the effect of high correlation.

Although the delay models described above assume behavior independent of input pins and of the phase of the transitions, it is easy to extend the delay models to reflect

1. input pin to output pin dependencies, and
2. rise and fall transitions at the input and output pins.

2.1.2 Memory elements

A memory element has three ports, an input port, an output port and a control port. In addition it may have inputs for set/reset and conditional control. The primary function of a memory element is to provide a means of storing past history of the circuit. It stores a value (logical 1 or 0) at its output until the control decides to store a fresh value. There are a variety of circuit structures that can be used as memory elements for data storage (see[23] for a detailed description of various memory elements). The different memory elements may be classified into two categories based on their behavior. We restrict attention to two memory elements which are representative of the two categories; falling edge-triggered D flip-flops (FEDFF) and active-high level-sensitive latches (AHLISL).

1. Edge-triggered memory element (flip-flop): As briefly explained in Chapter 1 (Section 1.4), an edge-triggered element is sensitive to falling (or rising) edges of the control signal. At the instant of occurrence of this edge, the input is sampled and the value is presented at the output. The output then stores this value until the next occurrence of the falling (or rising) edge.
2. Level-sensitive memory element (latch): For a latch, the data at the input port is transmitted to the output, throughout the interval that the control signal is high (or low). This is called an active-high (or active-low) latch. The output stores the input data value at the close of the interval until the start of the next active interval.

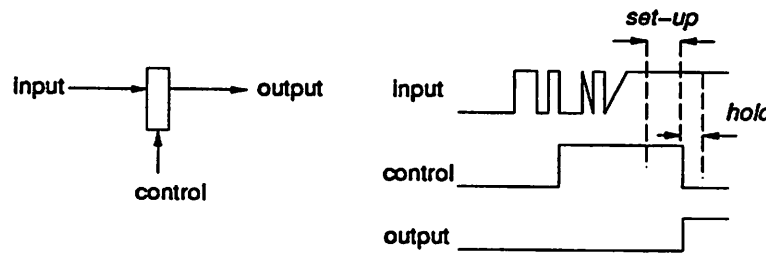


Figure 2.4: An ideal flip-flop

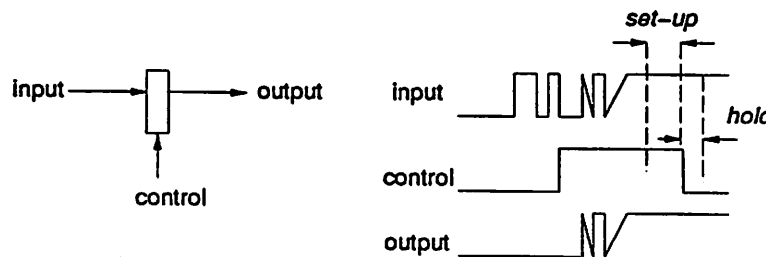


Figure 2.5: An ideal latch

For correct operation the data is required to be stable before the latching edge (the falling edge for a FEDFF or an AHL_{SL}), by an amount of time called the *set-up* time — this is known as the *set-up constraint*. The data has to remain stable after the latching edge for an amount of time called the *hold* time — this is known as the *hold constraint*. Figure 2.4 shows the behavior of an ideal FEDFF and Figure 2.5, an ideal AHL_{SL}.

An implementation for a memory element has the following deviations from ideal behavior.

1. The data takes a finite amount of time to propagate from the input port to the output port, after the clock event (rising for AHL_{SL} and falling for FEDFF) that causes an output change occurs.
2. The effect of an event at the clock port takes a finite amount of time to be seen at the output port, assuming that the data input has been ready, prior to the event at the clock port.
3. For correct behavior, there are requirements on the minimum pulse-width that the control signal can have.

An excellent description of memory element models may be found in [75].

2.1.3 Clock signals

Although there are no restrictions on the input to the control port of a memory element, we focus exclusively on circuits with clock phases connected to these ports. This precludes circuits with conditional clocking and circuits with data signals driving control ports. A reason for doing so is that hazards on the control port can cause incorrect behavior (functional and timing). Since we propose to use static timing analysis techniques, hazard detection/correction is downright difficult, if not impossible.

The signals in a circuit may be classified into two mutually exclusive sets; data signals which appear in combinational regions and inputs/outputs of memory elements, and clock signals which drive control ports of memory elements. A clocking scheme consists of a set of periodic clock signals (called phases). The rise and fall of the phases constitute the clock events. In this thesis, all phases are assumed to have the same periodicity, called the clock period.

2.2 Timing in VLSI circuits: a review

We review various research efforts in the area of sequential timing. This review is not complete (it would take an entire book to do so), but is intended to provide a brief description of the evolution of timing issues and to underline salient features that are borrowed in our approach.

2.2.1 Timing analysis

An excellent survey by Hitchcock, Sr. [21] describes several early efforts in timing analysis [50, 80, 58, 27, 30, 42, 22]. These timing analyzers suffer from poor modeling of memory elements (only flip-flops are assumed present). If a circuit has only FEDFF's, then it suffices to restrict attention to the timing analysis of combinational regions. Given arrival times at the inputs, we then need only compute arrival times at the outputs. An efficient approach is to levelize the gates in the circuit; namely starting at the inputs defined as level 0, assign a level to each gate which is one more than the maximum level of the gates that drive it. Then compute arrival times for a gate at level i only after all gates at level $i - 1$ have been processed. We shall have occasion to use this in the algorithms we develop.

In the early eighties, timing analysis on transistor net-lists was an area of intense research [48, 26, 68]. The first of these, CRYSTAL [48] uses a switch-level approach (advocated by a genre of simulators) to detect critical paths in the circuit. The delay model is based on the

RC model (lumped and distributed) described in [43, 52]. **CRYSTAL** uses a value-independent approach (unless specified, with the caveat that specifying data values may cause the program to fail in identifying critical paths eliminated by the specification). A problem inherent with value-independent analysis is that changes may be propagated oblivious of other conditions in the circuit. **CRYSTAL** provides satisfactory results to circuits with simple clocking schemes, since it does not have a notion of a clock signal (but rather treats them as inputs from the environment) and is unable to model level-sensitive latches.

TV [26] is a timing analysis program for nMOS circuits. A set of rules are used to determine signal flow before timing analysis. It uses breadth-first analysis to speed up the analysis.

The approach in **LEADOUT** [68] uses the notion of a causality graph and uses compiled code techniques to analyze the circuit. Multi-phase clocks are permitted and latches are correctly handled. It constructs a set of equations relating various events in the circuit. The delay information is stored in a separate database, which permits flexibility in analysis. **LEADOUT** is remarkable for its power (handling circuits with over 50,000 transistors in a few minutes on a VAX 780).

ATV (Abstract Timing Verifier)[77] was developed concurrently with **LEADOUT** and has the feature of plug-in delay models. **ATV** has an abstract notion of the circuit; thus it can be used for verification of micro-architectures as well as gate level net-lists. It uses loop unfolding to overcome the cyclic nature of sequential circuits. Hence, the analysis is true for only a user specified number of clock periods.

HUMMINGBIRD [78] was primarily designed as a timing analyzer operating in a logic synthesis environment. It incorporates sophisticated models for memory elements and permits multiple frequency clock signals. The algorithm resorts to an iterative technique, though no convergence properties are discussed. The motivation is to use timing analysis to provide feedback to the combinational resynthesis procedures.

Ishii *et al.*[24] present an algorithm for handling level-clocked circuitry, with arbitrary clock signals. The fixed delay model is used and only *set-up* constraints are checked. Sakallah *et al.*[54] present an elegant model which handles both kinds of memory elements in a simple manner. Using this model, they present an iterative approach for timing verification [55] called *CheckT_c*. Our approach is developed on these models. A formal definition of the models is given in Section 2.3.

All the timing verification techniques described so far are static, namely they are value-independent. A few of these possess the ability to do some data-dependent analysis, but only on a case by case basis. For sake of completeness, we briefly review the research in dynamic timing

analysis. The research so far has focussed on combinational logic only. Very often, paths in a circuit may not be exercised, *i.e.* no signal transitions propagate along the path. Such a path is termed a false path. The purpose of dynamic analysis is to disregard such paths and report the delay of only paths that can propagate transitions. The basic formulation of this problem can be found in [40, 13]. It involves capturing the sensitization criteria at every gate and solving the set of criteria efficiently. More recently, efficient techniques [41] based on solving Boolean SATISFIABILITY [67] have been reported. Although false paths are not encountered frequently in combinational circuits, they are extremely common in sequential circuits. Gated clocking and unreachable states in a state machine may contribute to paths that are conditionally active.

2.2.2 Timing optimization

Timing optimization encompasses all techniques that are used to improve the performance (clock period) of a circuit. This Section examines some techniques that are popular in literature and practice.

Combinational optimization

The problem of optimizing a combinational circuit has been well studied and numerous algorithms have been proposed. A representative sample of the optimization approaches may be classified into the following categories.

1. Circuit restructuring [66, 16, 17, 49]: A simple idea to decrease the delay of a circuit is to move late arriving signals in a cone of logic, closer towards the output. In practice, sophisticated techniques for selecting these “cones of logic” are required. Algorithms used for minimizing area, like logic decomposition are modified with a cost function that depends on some estimate of the final delay of the circuit.
2. Technology mapping and buffer optimization [65, 5, 73, 53, 36]: A critical step in logic synthesis is the binding of Boolean equations to an implementation technology. Standard technology mapping (such as [29]) yields a minimal area implementation. It can be tailored to yield circuits with less delay for an area penalty. Often signals need to be driven to several termini. The linear delay model reflects the delay due to the increased load. Buffer trees are constructed to ensure that signals arrive at destinations in time.

3. Rule based [9, 20]: These approaches use a variety of transformations to improve the performance of a circuit. Limitations inherent to all rule based systems are two-fold; the rules depend on the choice of technology and only local improvement can be gained.
4. Transistor sizing [64, 18, 57, 39]: This is a popular technique for performance optimization. The pioneering work done in TILOS demonstrated that the area and delay of a transistor netlist are representable as posynomial functions of transistor sizes. TILOS [18] uses a heuristic based on sensitivity computations. Shyu *et al.* [64] use the method of feasible directions (a standard non-linear optimization technique [51]) to solve the problem. iCONTRAST [57] uses a convex optimization technique proposed recently [76]. Marple [39] presents a solution to the transistor sizing problem based on Lagrangian multipliers.

Clock schedule optimization

The clock schedule optimization problem involves computing the minimum clock period and assigning instants of occurrence to the clock events (rise/fall of phases) in some time frame. The problem of computing the minimum clock period of a circuit with FEDFF's with a single phase clock is a relatively easy problem. The difficulty arises in circuits with level-sensitive latches using multi-phase clocking schemes. Data can stream through the active period of latches; this is called cycle stealing or retardation. It enables the circuit to operate at a shorter clock period, but also permits signal transitions to permeate across memory elements (see Figure 2.5). Thus latches are both a boon and a curse.

Various efforts at examining constraints for correct latch operation have hinted at possible algorithms to solve for the optimal clock problem. Unger *et al.* [75] provide an approach to solve the problem for 2 phase circuits. The pioneering work in optimal clocking can be traced to TAMIA [8]. The approach suggests starting with null retardation at a set of latches and successively updating the values (of retardation) as the iterations proceed. The iterations are used to "shave" time off in the different intervals that combine to yield the clocking scheme. Ishii *et al.* [25] present a polynomial algorithm to deal with 2 phase level-clocked circuitry, using the fixed delay model and considering only set-up constraints. Sakallah *et al.* [54, 55] propose an elegant model and use linear programming techniques to solve for the clock schedule. The optimality of the procedure [55] remains unanswered (most probably the algorithm is sub-optimal). The problem formulation in [55] is valid only in the steady state of circuit operation and leads to a non-convex solution set. A major contribution made by Szymanski [69] is to modify the constraints to ensure correct circuit

behavior from quiescence. This provides a formulation with a convex solution set. A drawback of the formulation in [55] is that several constraints in the linear programming formulation are redundant. Detecting a redundant constraint amongst a set of constraints is as hard as solving the linear program itself. Consequently general linear programming techniques are unable to eliminate constraints efficiently. Szymanski exploits the underlying graph structure of a circuit to obtain a reduced set of constraints that are necessary for the problem.

Sequential optimization: retiming

Retiming is a process of re-distributing memory elements in a circuit, to obtain a faster circuit with the same input-output behavior. The structure of logic gates remains unchanged. Leiserson *et al.* [33] were the first to provide an efficient algorithm to solve the retiming problem for single phase circuits with FEDFF's. Efforts have been made to extend retiming to deal with AHL's [25, 35], although the delay model (fixed delay) leaves something to be desired.

Sequential optimization: resynthesis

The constraints for a circuit to operate at a target clock period translate to a set of performance constraints on one or more "pieces" of combinational logic. In the single phase edge-triggered case, the pipeline performance optimization problem is equivalent to a combinational "speedup" problem [38]. The approach in [2] approximates level-sensitive latches by edge triggered flip-flops and handles arbitrary multi-phase circuits. Slack is used to direct logic resynthesis and logic movement across memory elements repeatedly, to find the best clock period at which the circuit can operate. The slack based approach is myopic in its optimization. To overcome this, simulated annealing is used to guide the optimization. However this may result in much larger circuits than necessary, especially when a target clock period is given. In order to help the combinational optimizers achieve their goal, it is important to expose large regions of combinational logic. Techniques described in [37, 11] make an effort to do so. The problem explored in [12] is to identify and eliminate circuit structures that prevent retiming from yielding a faster circuit. DeMicheli [10] uses the notion of synchronous logic operations, and combines combinational synthesis techniques with register movement to optimize the performance of single phase edge-triggered designs.

2.3 Definitions

This Section presents the mathematical models that form a basis for the rest of the thesis. These models were first proposed by Sakallah *et al.* in [55].

2.3.1 Clocking scheme

A clocking scheme, Φ is a collection of l periodic signals, ϕ_1, \dots, ϕ_l , each with a common period c , and is represented by $\Phi = (\phi_1, \phi_2, \dots, \phi_l)$. Associated with each phase ϕ_i are two real numbers s_i and e_i (short for start and end of the high interval of a phase), the time of occurrence of the rising and falling edges of ϕ_i ($0 \leq (s_i, e_i) \leq c$). Also associated with each phase ϕ_i is its local time frame, an interval of time of length c , such that the end of the active phase coincides with the end of the local time frame. A global time frame is chosen to coincide with the local time frame of a phase, henceforth called phase l . The phases are ordered so that $0 \leq e_1 \leq e_2 \dots \leq e_l = c$. We define a precedence relation (\prec) on phases as follows—

$$\phi_i \prec \phi_j \quad \text{if } e_i < e_j. \quad (2.1)$$

The precedence relation is

1. anti-reflexive $\phi_i \not\prec \phi_i$,
2. anti-symmetric $\phi_i \prec \phi_j$ implies $\phi_j \not\prec \phi_i$, and
3. transitive $\phi_i \prec \phi_j, \phi_j \prec \phi_k$ imply $\phi_i \prec \phi_k$.

The phase shift operator E_{ij} introduced in [55] is used to translate all measurements of time from the local frame of phase ϕ_i to the local frame of ϕ_j . The phase shift operator is defined for a combinational path between latches. For a path from a memory element clocked using phase ϕ_i to a memory element clocked using phase ϕ_j , the phase shift operator is defined as

$$E_{ij} = \begin{cases} e_j - e_i & \text{if } \phi_i \prec \phi_j \\ c + e_j - e_i & \text{otherwise} \end{cases} \quad (2.2)$$

The clocking scheme in Figure 2.6 shows a 2 phase clocking scheme. The values for E_{12} and E_{21} are also shown. Let an event be an upward or downward transition of a data signal. Consider an event at a ϕ_1 memory element occurring at time t_1 , given in terms of the local time frame for ϕ_1 . If this event causes another event at a ϕ_2 memory element with a delay say d , *i.e.*

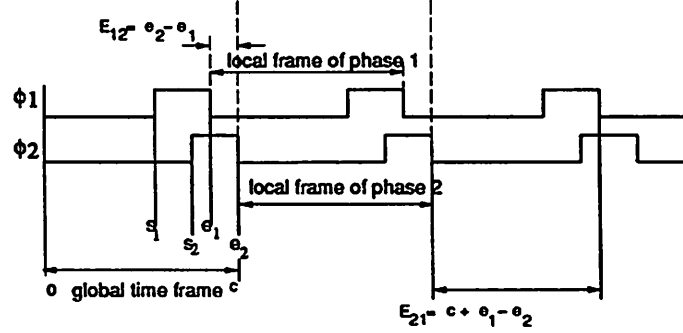


Figure 2.6: Two phase clocking scheme

time $t_1 + d$ in the local frame of ϕ_1 ; then E_{12} is the shift that must be subtracted from $t_1 + d$ to convert the event to the local time frame of ϕ_2 . To distinguish between variables in the local time frame from the global frame we use a superscript L . Thus in the local time frame of ϕ_i ,

$$e_i^L = c. \quad (2.3)$$

The local rise of a phase is

$$s_i^L = s_i + e_l - e_i = s_i + E_{il} \quad \text{if } s_i < e_i \quad (2.4)$$

and

$$s_i^L = s_i - e_i = s_i + E_{il} - e_l \quad \text{if } s_i > e_i. \quad (2.5)$$

Consequently, *a priori* information is needed on the relative occurrence of the rise and fall of each phase in the global frame and the relative occurrence of the fall of each of the phases for correct translation of events. This is specified by the clocking scheme. If all the values to $s_i, e_i, i = 1, \dots, l$ are known, the set Φ is called a clock schedule. Thus a clock schedule is an assignment to the variables that conforms to the clocking scheme. Note that an s_i or e_i without the superscript L refers to the instant of occurrence of the rise or fall of phase i in the global time frame (local time frame of ϕ_l). For the rest of the thesis we shall assume that $s_i < e_i$ for all the phases.

2.3.2 Combinational circuit

The thesis assumes all timing events are value-independent. In other words, the data signal can only be stable or changing at any instant. The actual stable value (high or low) is of no concern, nor is the direction of change (high to low or low to high). This simplifies the problems to make them tractable, albeit pessimistic.

A_i^L	=	latest that the signal is valid at the input of memory element i ,
a_i^L	=	earliest that the signal is valid at the input of memory element i ,
R_i^L	=	latest that the signal is valid at the output of memory element i ,
r_i^L	=	earliest that the signal is valid at the output of memory element i .

Table 2.1: Variables at a memory element

The simplified min-max delay model is used for each gate. The circuit \mathcal{C} is modeled as a finite, edge-bi-weighted, directed graph $G = (V, E, D, d)$. For every memory element $i \in \mathcal{C}$ there is a vertex $i \in V$; thus we use i for a memory element and the vertex representing it in G . G is called the **latch graph**. In addition for every primary input and primary output of the circuit there is a vertex in V . If there is a path of combinational logic from a memory element (or primary input), say i , to a memory element (or primary output), say j , we create an edge $e_{ij} : i \rightarrow j$ (directed from i to j), $e_{ij} \in E$. The weight D_{ij} (d_{ij}) is the maximum (minimum) sum of the gate delays along any combinational path from i to j : This is computed as explained in Section 2.2.1. We say i is a **fanin** of j (j is a **fanout** of i) if there is a directed edge from i to j in the graph. We denote the fanin set of i by $FI(i)$, the fanout set by $FO(i)$. A path $i_1 \rightsquigarrow i_p$ is a sequence of vertices $\{i_1, i_2, \dots, i_p\}$, such that every pair of successive vertices in the sequence have an edge between them, namely $\exists e_{i_k i_{k+1}} : i_k \rightarrow i_{k+1}$, for $k = 1, \dots, p-1$. A cycle is a path $i_1 \rightsquigarrow i_{p+1}$ whose first and last vertices coincide ($i_1 = i_{p+1}$). A cycle $i_1 \rightsquigarrow i_{p+1}$ is a **simple cycle** if i_1, \dots, i_p are all distinct and $i_1 = i_{p+1}$. For the rest of the thesis we will be concerned with simple cycles only. We denote the phase controlling latch i , as $\phi(i)$.

The symbol i is overloaded to mean a memory element $i \in V$ or a phase $i \in \{1, \dots, l\}$. The rise (fall) of the phase to memory element i is denoted by $s_{\phi(i)}$ ($e_{\phi(i)}$).

2.3.3 Memory elements and circuit clocking constraints

Four variables are associated with each memory element in the circuit; they are defined in Table 2.1. These variables are measured with respect to the local time frame of the phase of the latch. This subsection describes the constraints that model the circuit and the meaning of each constraint will be elucidated using signal waveforms. A word about the notation used in displaying

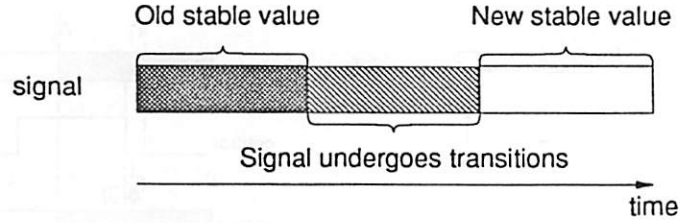


Figure 2.7: Key to waveforms

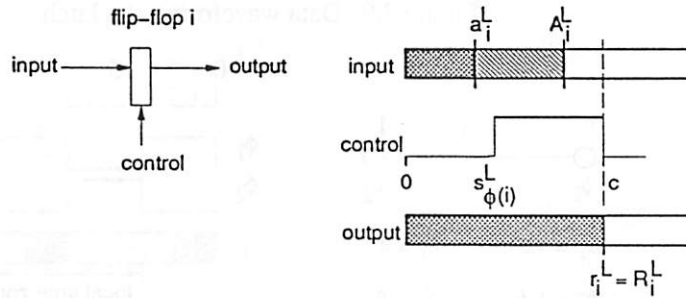


Figure 2.8: Data waveforms at a flip-flop

waveforms; a signal at the input/output of a memory element or a gate takes on 3 possible states during a clock period, displayed by different shades in a figure. Figure 2.7 provides a key to the signal waveforms. The signal has an old stable value to begin with, then some early event/s force a transition. This commences the signal changing interval. After the last possible transition has taken place the signal assumes the new stable value. The signals to the control ports of memory elements (*i.e.* the clock phases) are assumed to be clean (free of unexpected transitions).

The behavior of a flip-flop i can be represented by

$$R_i^L = e_{\phi(i)}^L = c, \tag{2.6}$$

and

$$r_i^L = e_{\phi(i)}^L = c. \tag{2.7}$$

The significance of these equations can be seen in Figure 2.8.

The equations for a latch are

$$R_i^L = \max(A_i^L, s_{\phi(i)}^L), \tag{2.8}$$

and

$$r_i^L = \max(a_i^L, s_{\phi(i)}^L). \tag{2.9}$$

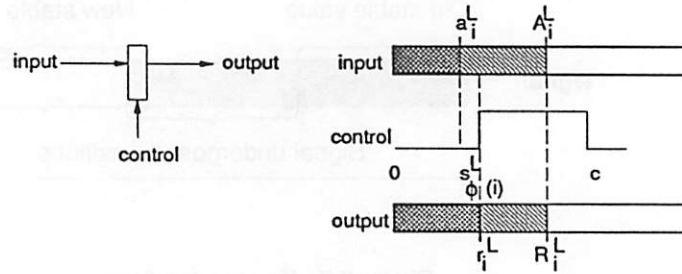


Figure 2.9: Data waveforms at a latch

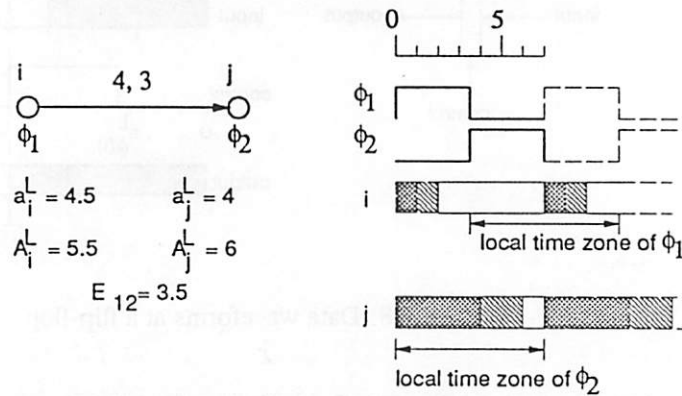


Figure 2.10: Propagation of data waveforms

Figure 2.9 portrays the implications of these equations.

The equations for propagation through combinational regions are

$$A_j^L = \max_{i \in FI(j)} (R_i^L + D_{ij} - E_{\phi(i)\phi(j)}), \quad (2.10)$$

and

$$a_j^L = \min_{i \in FI(j)} (r_i^L + d_{ij} - E_{\phi(i)\phi(j)}). \quad (2.11)$$

A pictorial interpretation of these constraints for an edge $i \rightarrow j$ in the latch graph is shown in Figure 2.10. Let the early (late) arrival at memory element i be 4.5 (5.5). $E_{12} = e_2 - e_1 = 7 - 3.5 = 3.5$ for the clocking scheme in Figure 2.10. The minimum (maximum) delay along $i \rightarrow j$ is 3 (4). Thus the early (late) arrival at memory element j is $4.5 + 3 - 3.5 = 4.0$ ($5.5 + 4 - 3.5 = 6$).

The constraints for correct data latching are

$$A_i^L \leq e_{\phi(i)}^L - S, \quad (2.12)$$

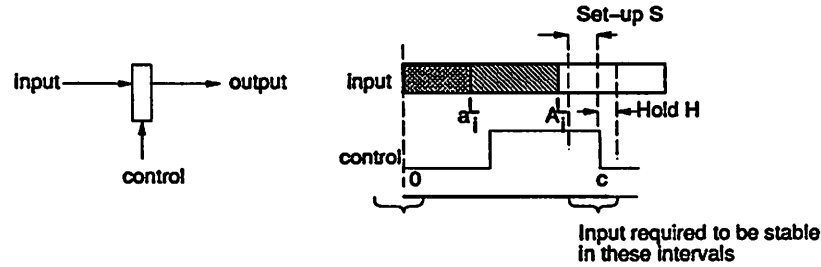


Figure 2.11: Set-up and hold constraints

and

$$a_i^L \geq H. \quad (2.13)$$

Equation 2.12 is the **set-up constraint**. It ensures that there is sufficient time for computation between (minimum separation) clock events. The constant S is known as the set-up time for memory element i . Equation 2.13 is known as the **hold constraint**. Its purpose is to prevent early signals from corrupting inputs to memory elements. The constant H is called the hold time of memory element i . Although the set-up and hold times can be different for different memory elements we assume (for sake of simplifying notation) that all memory elements have the same set-up and the same hold values. Figure 2.11 shows these constraints at a memory element.

Constraints 2.2-2.11 model the behavior of the circuit; constraints 2.12 and 2.13 enforce data stability during latching — together they are known as **circuit clocking constraints**. They are summarized in Table 2.2.

2.3.4 An example

Chapter 3 and Chapter 4 solve two closely related problems. We describe an example which will be used to clarify some aspects of the algorithms as we proceed. Consider the circuit shown in Figure 2.12. The circuit using flip-flops was first described in [82]. It has been modified so that level-sensitive latches are used instead. The circuit is a data-path of a video compression system and uses a delta PCM compression algorithm. The compression is achieved by a non-linear quantization operation Q . Its inverse D is used to maintain the prediction value. The rest of the components are **adders (+)** and a **subtractor (-)**. The delays of each component are shown alongside ([max, min] delays). The input signal is 9 bits and the compressed value is 6 bits. All 9 bit lines are shown as dark lines and the 6 bit lines are light. The circuit uses a 2-phase clocking scheme. For sake of analysis, assume that there is no delay through the memory elements and that the clock

Nature of constraint	Formulation
Data propagation	for $i = 1, \dots, V $
Long path	$A_i^L = \max_{j \in FI(i)} (R_j^L + D_{ji} - E_{\phi(j)\phi(i)})$
Short path	$a_i^L = \min_{j \in FI(i)} (r_j^L + d_{ji} - E_{\phi(j)\phi(i)})$
Memory elements	for $i = 1, \dots, V $
Late departure (latch)	$R_i^L = \max(A_i^L, s_{\phi(i)}^L)$
Late departure (flip-flop)	$R_i^L = c$
Early departure (latch)	$r_i^L = \max(a_i^L, s_{\phi(i)}^L)$
Early departure (flip-flop)	$r_i^L = c$
Correct latching	for $i = 1, \dots, V $
Set-up	$A_i^L \leq c - S$
Hold	$a_i^L \geq H$

Table 2.2: Clocking constraints

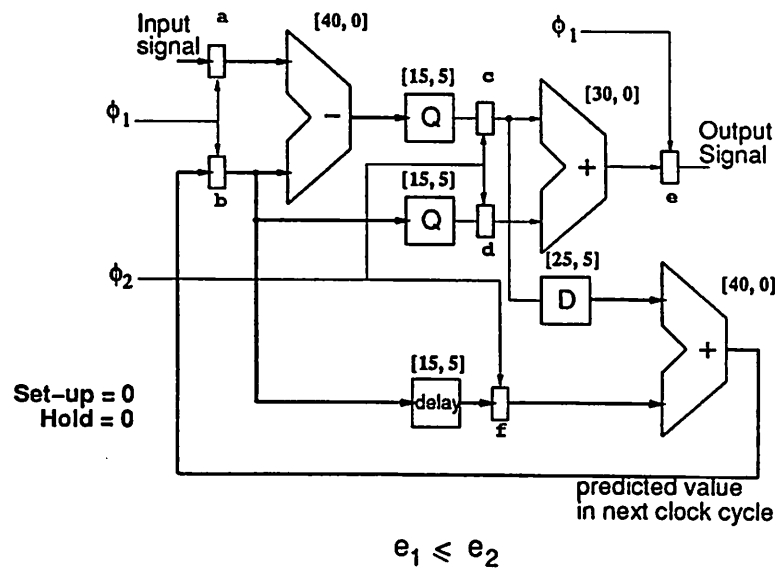


Figure 2.12: Example: video coder

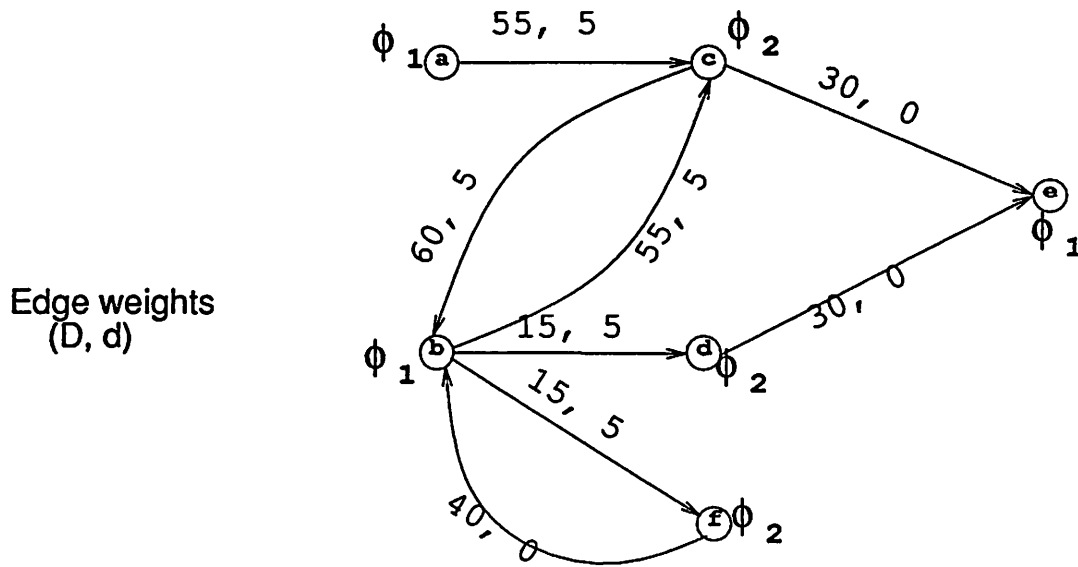


Figure 2.13: Latch graph for video coder

signals arrive at the memory elements without any skew. The set-up and hold times for all latches are assumed to be 0. The corresponding latch graph is shown in Figure 2.13. Each edge $i \rightarrow j$ has 2 weights, namely D_{ij} and d_{ij} .

2.4 Discussion

We shall ignore clock skew during the development of algorithms for ease of presentation. At the end of each Chapter, a brief note describes how clock skew is easily incorporated. To model clock skew, we partition the circuit into two sub-circuits; the first part consists of gates in the clock distribution circuitry and is called the skew network. The rest of the circuit forms the second part and is termed the data network. The data network is modeled using the latch graph described in Section 2.3.2. The skew network consists of paths from the clock signal pins to the memory elements. Let the phase signal to the control input of latch i undergo a skew along path P_i . P_i consists of buffers/inverters and is disjoint from the data network. A buffer/inverter k on P_i has maximum delay $B_k^{P_i}$ ($b_k^{P_i}$).

A natural approach to systematic investigation of a research area is to solve a series of related problems that are increasingly difficult. At the time of writing this thesis, a summary of the various problems in sequential clocking is given in Table 2.3 (in increasing order of difficulty). Of

<i>Problem</i>	<i>Comments</i>
Clock Verification	Given \mathcal{C} , c and $s_i, e; i = 1, \dots, l$, does circuit operate correctly ? See [24, 55].
Clock Optimization	Given \mathcal{C} , find smallest c and $s_i, e; i = 1, \dots, l$ so that circuit operates correctly. See [55, 69, 25].
Retiming	Given \mathcal{C} , reposition memory elements for smallest c . See [33, 25, 35].
Resynthesis	Given \mathcal{C} , resynthesize circuit for smallest c . See [38, 2].

Table 2.3: Clocking issues

these, we shall focus on the clock verification, clock optimization and pipeline resynthesis problems. The intricate link relating them will become clear over the next few Chapters.

Chapter 3

Clock Schedule Verification

Current VLSI designs have several thousand gates and memory elements. Ensuring that a clock schedule adheres to the *clocking constraints* is a difficult problem for the human designer, especially in the presence of level-sensitive memory elements. Another reason for investigating this problem is that previous approaches [55, 78] fail to provide an analysis on the complexity of their procedures. Lastly the development of sophisticated algorithms to solve other clocking issues is hampered without the ability to verify the resulting solution.

3.1 Overview

The *clocking constraints* (Table 2.2) are summarized in the Table 3.1. Since the clock period and clock events are known, the equations are simplified by combining the constant terms (column 3, Table 3.1). An important fact to note is that the A^L and R^L variables appear only in the long path propagation and late departure constraints (called the *late equation set*). Similarly the a^L and the r^L variables occur only in the short path propagation and early departure constraints (called the *early equation set*). The set-up and hold constraints impose bounds on A^L and a^L . The clock verification problem may be posed as:

Given a circuit $G(V, E, D, d)$ and a clock schedule, does the clock schedule satisfy the clocking constraints.

Note that the clocking constraints introduce auxiliary variables (A^L, a^L, R^L and r^L). Hence the verification procedure (if we are to use the clocking constraints in the form described so far) must ensure that the set-up and hold constraints are satisfied for all possible values of the auxiliary variables that are solutions to the set of constraints. It is this fact that leads to complications,

<i>Nature of constraint</i>	<i>Formulation</i>	<i>Simplification</i>
Long path propagation	for $i = 1, \dots, V $ $A_i^L = \max_{j \in FI(i)} (R_j^L + D_{ji} - E_{\phi(j)\phi(i)})$	$A_i^L = \max_{j \in FI(i)} (R_j^L + \Delta_{ji})$
Short path propagation	$a_i^L = \min_{j \in FI(i)} (r_j^L + d_{ji} - E_{\phi(j)\phi(i)})$	$a_i^L = \min_{j \in FI(i)} (r_j^L + \delta_{ji})$
Memory element	for $i = 1, \dots, V $	
Late departure (latch)	$R_i^L = \max(A_i^L, s_{\phi(i)}^L)$	same
Late departure (flip-flop)	$R_i^L = c$	same
Early departure (latch)	$r_i^L = \max(a_i^L, s_{\phi(i)}^L)$	same
Early departure (flip-flop)	$r_i^L = c$	same
Correct latching	for $i = 1, \dots, V $	
Set-up	$A_i^L \leq c - S$	same
Hold	$a_i^L \geq H$	same

Table 3.1: Simplified clocking constraints

hitherto unreported by previous approaches.

The current Chapter is organized as follows. Section 3.2 describes the theoretical results concerning the clock schedule verification problem. The existence of multiple solutions and conditions for unique solutions to exist form the contents of Section 3.3. Application of the algorithm on an example and the results on a set of benchmarks are provided in Section 3.4.

3.2 Theoretical issues

One method to solve the clocking constraints is to use iteration. The constraints form a map $\mathcal{M} : X \rightarrow X$, where X is a set containing the solutions. Since the variables appear on both sides of the max and min equations, a solution $x^* \in X$ must be a fixed point of the map \mathcal{M} , namely

$$x^* = \mathcal{M}(x^*). \quad (3.1)$$

An iterative method starts out with an initial guess x^0 for the solution to the problem and produces a sequence of iterates, such that $x^{i+1} = \mathcal{M}(x^i)$. The method succeeds in finding a solution if it can be shown that $x^{i+1} = x^i$ after a number of iterations. Note that this method yields *one* possible solution and is useful only when a unique solution is guaranteed to exist in X . In the case of the clocking constraints, uniqueness of a solution is guaranteed only in a restricted case. The solution in general will depend on the choice of x^0 . We shall henceforth drop the superscript L on the local

variables associated with each memory element. A superscript m will instead refer to the variable in the m^{th} iteration. We denote a solution to the clocking constraints as $(\hat{A}, \hat{R}, \hat{a}, \hat{r})$.

We begin by presenting a few lemmas that will be invoked repeatedly.

Lemma 3.2.1 *If $i_1 \rightsquigarrow i_p$ is a path in the circuit graph, then $\hat{A}_{i_p} \geq \hat{R}_{i_1} + \sum_{k=1}^{p-1} \Delta_{i_k i_{k+1}}$.*

Proof For each i_k , $\hat{R}_{i_k} \geq \hat{A}_{i_k}$ and for each edge $i_k \rightarrow i_{k+1}$, $\hat{A}_{i_{k+1}} \geq \hat{R}_{i_k} + \Delta_{i_k i_{k+1}}$. Summing over $k = 1, \dots, p-1$, we obtain $\hat{A}_{i_p} \geq \hat{R}_{i_1} + \sum_{k=1}^{p-1} \Delta_{i_k i_{k+1}}$. ■

Corollary 3.2.2 *If $i_1 \rightsquigarrow i_p$ is a path in the circuit graph, then $\hat{R}_{i_p} \geq \hat{R}_{i_1} + \sum_{k=1}^{p-1} \Delta_{i_k i_{k+1}}$.*

Lemma 3.2.3 *Let $(\hat{A}, \hat{R}, \hat{a}, \hat{r})$ be a solution to the clocking constraints. Then for any cycle $C : i_1 \rightsquigarrow i_{p+1}$ in the circuit graph, $\sum_{k=1}^p \Delta_{i_k i_{k+1}} \leq 0$.*

Proof From Lemma 3.2.2, we obtain for a cycle $\hat{R}_{i_{p+1}} \geq \hat{R}_{i_1} + \sum_{k=1}^p \Delta_{i_k i_{k+1}}$. Since $\hat{R}_{i_1} = \hat{R}_{i_{p+1}}$, the result follows. ■

Expanding $\Delta_{i_k i_{k+1}} = D_{i_k i_{k+1}} - E_{\phi(i_k)\phi(i_{k+1})}$ and using the definition of $E_{\phi(i_k)\phi(i_{k+1})}$ (from Equation 2.2), we see that $\sum_{k=1}^p E_{\phi(i_k)\phi(i_{k+1})}$ telescopes to $K_C c$; where K_C is an integer which indicates the number of clock periods available for computation and depends on the cycle C . Hence, for all cycles $C : i_1 \rightsquigarrow i_{p+1}$

$$\sum_{k=1}^p D_{i_k i_{k+1}} \leq K_C c. \quad (3.2)$$

A cycle for which $\sum_{k=1}^p \Delta_{i_k i_{k+1}} = 0$ is called a zero weight cycle in the latch graph. A cycle is said to be a positive weight cycle /negative weight cycle depending on the sign (positive/negative) of $\sum_{k=1}^p \Delta_{i_k i_{k+1}}$. Observe that the contrapositive to the statement of Lemma 3.2.3 implies that late equation set has no solution if there is a positive weight cycle.

Proposition 3.2.4 *For any edge $i \rightarrow j$, $\delta_{ij} \leq \Delta_{ij}$.*

This is a consequence of the fact that by definition $d_{ij} \leq D_{ij}$ and yields the following Corollary.

Corollary 3.2.5 *Let $(\hat{A}, \hat{R}, \hat{a}, \hat{r})$ be a solution to the clocking constraints. Then for any cycle $i_1 \rightsquigarrow i_{p+1}$ in the circuit graph, $\sum_{k=1}^p \delta_{i_k i_{k+1}} \leq 0$.*

3.2.1 Solving the late equation set

An iterative scheme for finding a solution to the late equation set is given in Procedure 3.2.1.

Procedure 3.2.1

1. Initialization-for all memory elements i

$$A_i^0 = -\infty$$

$$R_i^0 = \begin{cases} c & \text{if } i \text{ is a flip-flop} \\ s_{\phi(i)}^L & \text{if } i \text{ is a latch} \end{cases}$$

2. Iteration-for $m = 1, \dots, n$ {

• for each memory element i {

$$- A_i^m = \max_{j \in FI(i)} (R_j^{m-1} + \Delta_{ji})$$

$$- R_i^m = \begin{cases} c & \text{if } i \text{ is a flip-flop} \\ \max(A_i^m, s_{\phi(i)}^L) & \text{if } i \text{ is a latch} \end{cases}$$

}

}

Lemma 3.2.6 A_i^m and R_i^m are monotonically increasing with m at every memory element i

Proof Using induction.

1. Base case: $m = 1$

Quite clearly for all i , $A_i^1 \geq A_i^0$. Moreover,

$$R_i^1 \geq s_{\phi(i)}^L = R_i^0 \quad \text{if memory element } i \text{ is a latch, and}$$

$$= c = R_i^0 \quad \text{if memory element } i \text{ is a flip-flop.}$$

Thus $R_i^1 \geq R_i^0$.

2. Inductive case: Assume $A_i^m \geq A_i^{m-1}$ and $R_i^m \geq R_i^{m-1}$. We need to show, $A_i^{m+1} \geq A_i^m$ and $R_i^{m+1} \geq R_i^m$. Now $A_i^{m+1} = \max_{j \in FI(i)} (R_j^m + \Delta_{ji})$. For each term $j \in FI(i)$, $R_j^m + \Delta_{ji} \geq R_j^{m-1} + \Delta_{ji}$. Consequently $\max_{j \in FI(i)} (R_j^m + \Delta_{ji}) \geq \max_{j \in FI(i)} (R_j^{m-1} + \Delta_{ji})$, implying $A_i^{m+1} \geq A_i^m$.

If i is a flip-flop the $R_i^{m+1} = R_i^m = c$. If i is a latch, $A_i^{m+1} \geq A_i^m$, hence $\max(A_i^{m+1}, s_{\phi(i)}^L) \geq \max(A_i^m, s_{\phi(i)}^L)$. Thus $R_i^{m+1} \geq R_i^m$, for both cases.

For the sake of convergence it suffices to restrict all memory elements to latches. If any flip-flops are present, they can only abet the convergence of the algorithm. This will be clear after the following lemma.

Lemma 3.2.7 *Procedure 3.2.1 converges to a solution, if one exists in $n(= |V|)$ iterations.*

Proof The proof proceeds by contradiction. Assume that there exists a solution to the late equation set. Let $A_{i_n}^n > A_{i_n}^{n-1}$ for some latch i_n . $A_{i_n}^k$ and $R_{i_n}^k$ are monotonically increasing with k (for all i_n). By definition (Equation 2.10) of $A_{i_n}^n$, there must exist some $i_{n-1} \in FI(i_n)$, such that $A_{i_n}^n = R_{i_{n-1}}^{n-1} + \Delta_{i_{n-1}i_n}$. The fact that $A_{i_n}^n > A_{i_n}^{n-1}$, yields two key results.

1. $R_{i_{n-1}}^{n-1} + \Delta_{i_{n-1}i_n} > A_{i_n}^{n-1} \geq R_{i_{n-1}}^{n-2} + \Delta_{i_{n-1}i_n}$; in other words $R_{i_{n-1}}^{n-1} > R_{i_{n-1}}^{n-2}$.
2. $R_{i_{n-1}}^{n-1} > R_{i_{n-1}}^{n-2} \geq s_{\phi(i_{n-1})}^L$, gives us the additional information that (Equation 2.8) $R_{i_{n-1}}^{n-1} = A_{i_{n-1}}^{n-1}$.

As a consequence, $A_{i_{n-1}}^{n-1} > R_{i_{n-1}}^{n-2} \geq A_{i_{n-1}}^{n-2}$, i.e. $A_{i_{n-1}}^{n-1} > A_{i_{n-1}}^{n-2}$. Continuing this procedure, we trace a path in the latch graph with $n(= |V|)$ edges and $|V| + 1$ vertices. Hence some vertex must have been repeated more than once, implying that a cycle is contained in this path. Now for iterations p, q ($p > q$), and i_p, i_q on the path, $A_{i_p}^p = R_{i_q}^q + \sum_{k=q}^{p-1} \Delta_{i_k i_{k+1}}$. Now pick a vertex that has been repeated such that $i_p = i_q$. This results in

$$\begin{aligned} A_{i_p}^p &> A_{i_q}^q \\ \Rightarrow R_{i_q}^q + \sum_{k=q}^{p-1} \Delta_{i_k i_{k+1}} &> A_{i_q}^q \\ \Rightarrow \sum_{k=q}^{p-1} \Delta_{i_k i_{k+1}} &> A_{i_q}^q - R_{i_q}^q. \end{aligned}$$

Since, $R_{i_q}^q > R_{i_q}^{q-1} \geq s_{\phi(i_q)}^L$, we conclude $R_{i_q}^q = A_{i_q}^q$, implying $\sum_{k=q}^{p-1} \Delta_{i_k i_{k+1}} > 0$. Since by assumption the late equation set has a solution, we obtain a contradiction to Corollary 3.2.3. ■

The presence of a flip-flop ensures that the early and late departures from it are fixed for all iterations. Thus no flip-flop can occur on the path that prevents convergence in n iterations. The proof of convergence is identical to the proof of convergence for the longest path algorithm in a graph with no positive cycles [1].

Theorem 3.2.8 *The iterative procedure to obtain a solution to the late equation set runs in $O(|V||E|)$.*

Proof Each iteration requires $O(|E|)$ evaluations. (A^n, R^n) can be computed in $O(n|E|)$. ■

3.2.2 Solving the early equation set

There are several choices of initial values to start the iterations for the early equation set. The two natural choices for a memory element i are-

1-

$$\begin{aligned} a_i^0 &= A_i^0 \\ r_i^0 &= R_i^0 \quad \text{and} \end{aligned}$$

2-

$$\begin{aligned} a_i^0 &= A_i^n \\ r_i^0 &= R_i^n. \end{aligned}$$

Of these only the first leads to an algorithm that converges in polynomial time. The second choice for the initial guess can

- lead to an *erroneous* solution (this will be clarified in Section 3.3.3),
- take an arbitrary long time to converge.

The details of convergence of the iterations with the second choice of initial guess may be found in [61]. We shall focus only on the algorithm with the first choice of initial guess.

An algorithm to compute a solution to the early equation set is given in Procedure 3.2.2.

Procedure 3.2.2

1. *Initialization-* for all memory elements i

$$\begin{aligned} a_i^0 &= -\infty \\ r_i^0 &= \begin{cases} c & \text{if } i \text{ is a flip-flop} \\ s_{\phi(i)}^L & \text{if } i \text{ is a latch} \end{cases} \end{aligned}$$

2. *Iteration-* for $m = 1, \dots, n$ {

- for all memory elements i {
 - $a_i^m = \min_{j \in FI(i)} (r_j^{m-1} + \delta_{ji})$

$$\left. \begin{array}{l}
- r_i^m = \begin{cases} c & \text{if } i \text{ is a flip-flop} \\
\max(a_i^m, s_{\phi(i)}^L) & \text{if } i \text{ is a latch} \end{cases} \\
\end{array} \right\}$$

Lemma 3.2.9 a_i^m and r_i^m are monotonically increasing with m at every memory element i

Proof Using induction.

1. Base case: $m = 1$

Quite clearly for all i , $a_i^1 \geq a_i^0$. Moreover,

$$\begin{aligned}
r_i^1 &\geq s_{\phi(i)}^L = r_i^0 && \text{if memory element is a latch} \\
&= c = r_i^0 && \text{if memory element is a flip-flop.}
\end{aligned}$$

Thus $r_i^1 \geq r_i^0$.

2. Inductive case: Assume $a_i^m \geq a_i^{m-1}$ and $r_i^m \geq r_i^{m-1}$. We show, $a_i^{m+1} \geq a_i^m$ and $r_i^{m+1} \geq r_i^m$. Now $a_i^{m+1} = \min_{j \in FI(i)}(r_j^m + \delta_{ji})$. For each term $j \in FI(i)$, $r_j^m + \delta_{ji} \geq r_j^{m-1} + \delta_{ji}$. Hence $\min_{j \in FI(i)}(r_j^m + \delta_{ji}) \geq \min_{j \in FI(i)}(r_j^{m-1} + \delta_{ji})$, implying $a_i^{m+1} \geq a_i^m$.

If i is a flip-flop the $r_i^{m+1} = r_i^m = c$. If i is a latch, $a_i^{m+1} \geq a_i^m$, hence $\max(a_i^{m+1}, s_{\phi(i)}^L) \geq \max(a_i^m, s_{\phi(i)}^L)$. Thus $r_i^{m+1} \geq r_i^m$, for both cases. ■

Once again, for the sake of convergence it suffices to restrict all memory elements to latches.

Lemma 3.2.10 Procedure 3.2.2 converges to a solution, if one exists in $n(= |V|)$ iterations.

Proof The proof proceeds by contradiction. Assume that there exists a solution to the early equation set. Let $a_{i_n}^n > a_{i_n}^{n-1}$ for some latch i_n . Recall $a_{i_n}^k$ and $r_{i_n}^k$ are monotonically increasing with k (for all i_n). By definition (Equation 2.11) there must exist some $i_{n-1} \in FI(i_n)$, such that $a_{i_n}^{n-1} = r_{i_{n-1}}^{n-2} + \delta_{i_{n-1}i_n}$. Knowing that $a_{i_n}^n > a_{i_n}^{n-1}$, results in two interesting facts.

1. $r_{i_{n-1}}^{n-1} + \delta_{i_{n-1}i_n} \geq a_{i_n}^n > a_{i_n}^{n-1} = r_{i_{n-1}}^{n-2} + \delta_{i_{n-1}i_n}$; in other words $r_{i_{n-1}}^{n-1} > r_{i_{n-1}}^{n-2}$.
2. $r_{i_{n-1}}^{n-1} > r_{i_{n-1}}^{n-2} \geq s_{\phi(i_{n-1})}^L$, gives us the additional information that (Equation 2.9) $r_{i_{n-1}}^{n-1} = a_{i_{n-1}}^{n-1}$.

As a consequence, $a_{i_{n-1}}^{n-1} > r_{i_{n-1}}^{n-2} \geq a_{i_{n-1}}^{n-2}$, i.e. $a_{i_{n-1}}^{n-1} > a_{i_{n-1}}^{n-2}$. Continuing this procedure, we trace a path in the latch graph with $n (= |V|)$ edges and $|V| + 1$ vertices. Hence some vertex must have been repeated more than once, implying that a cycle is contained in this path. Now for iterations p, q ($p > q$), and i_p, i_q on the path, $a_{i_p}^p \leq r_{i_q}^q + \sum_{k=q}^{p-1} \delta_{i_k i_{k+1}}$. Now pick a vertex that has been repeated such that $i_p = i_q$. This results in

$$\sum_{k=q}^{p-1} \delta_{i_k i_{k+1}} > a_{i_p}^p - r_{i_q}^q.$$

But since, $r_{i_q}^q > r_{i_q}^{q-1} \geq s_{\phi(i_q)}^L$, we conclude $r_{i_q}^q = a_{i_q}^q$, implying

$$\sum_{k=q}^{p-1} \delta_{i_k i_{k+1}} > a_{i_p}^p - a_{i_q}^q.$$

Since $a_{i_p}^k = a_{i_q}^k$ is monotonically increasing with k , it follows $a_{i_p}^p - a_{i_q}^q \geq 0$, for $p > q$. Thus

$$\sum_{k=q}^{p-1} \delta_{i_k i_{k+1}} > 0.$$

This contradicts Corollary 3.2.5. ■

From Proposition 3.2.4 and Corollary 3.2.3, we conclude that if the early equation set fails to have a solution, the late equation set must also fail to have a solution. Thus the early equation set converges to a solution only if the late equation set converges to a solution.

Theorem 3.2.11 *The iterative procedure to obtain a solution to the early equation set runs in $O(|V||E|)$.*

Proof Similar to the proof of Theorem 3.2.8. ■

3.3 Uniqueness of solutions

The algorithms in Procedure 3.2.1 and Procedure 3.2.2 present an iterative approach to find a solution to the late and early equation sets if one exists. An example where multiple solutions exist is shown in Figure 3.1. The delays of the combinational regions (a and b) are given by the pair of numbers in brackets ($[\max, \min]$) adjacent to it. The set of feasible solutions (all instants are given in the local time zone of phase 2) to the late equation set, parameterized by $\alpha \in [0, 1]$ is given by -

$$\begin{aligned} A_1 &= R_1 = 3 + \alpha \\ A_2 &= R_2 = 8 + \alpha. \end{aligned}$$

<i>name</i>	<i>read-in</i> (sec.)	<i>time</i> (sec.)
2planet	12.99	0.01
2s1423	16.75	0.07
2s5378	32.04	0.06
2s9234	48.28	0.13
2s13207	64.15	0.26
2s38584	3672.93	1.77
2s38417	1014.82	4.17
2s35932	650.39	0.78

Table 3.6: Clock verification with library delay model

are devoid of any latching errors. Latching errors can arise due to one or more of the following reasons —

1. the clock period is so critical that at every iteration of the late equation set there exists some latch for which the late arrival does not converge, or
2. the late equation set converges and there is a set-up violation, or
3. the early equation set converges and there is a hold violation.

The first case may force all $n(= |V|)$ iterations in Procedure 3.2.1. The procedure, as described, cannot predict that a proposed clock schedule has too small a clock period *a priori* to completing the n iterations. A heuristic based on maintaining predecessor pointers can be used to detect a critical clock period during the relaxation of the long path equations. Although it cannot guarantee less than n iterations, in practice it is very efficient. This will be presented in Chapter 4 (close of Section 4.5.2) in a different context.

3.5 Discussion

Although the circuit model has ignored skew along the clock lines, it is easy to extend the algorithm to incorporate them. Note that the complexity of the algorithm is independent of the number of phases in the circuit. We can conceptually extend the number of phases in the circuit to include one phase for each latch. The opening and closing of a latch needs to be recomputed using the worst case skew. Since the computation of the late and early arrival times is de-coupled, the

clock events in the two equations can be independent. First, consider all clock events in the global time frame. Translate the clock events at a latch (early clock rise, late clock rise, early clock fall and late clock fall), to yield a separate phase for each latch. Order the $|V|$ phases according to the late clock falls. The phase shift operator is calculated for each pair.

The opening of a latch i will take on the early clock rise value in the early equation set and the late clock rise value in the late equation set. Similarly the closing of a latch will assume the early clock fall value in the late equation set and late clock fall value in the early equation set. The set-up is measured with respect to the early clock fall and the hold with respect to the late clock fall.

In conclusion, we have introduced the clock schedule verification problem and presented the mathematical framework for solving it efficiently. This leads us to be optimistic in our search for an algorithm to compute the clock schedule with a minimum clock period that meets the clocking constraints. We have demonstrated that there can be multiple solutions to the clocking constraints for the given model. An interpretation to the multiple solutions has also been provided. It was the paper by Sakallah *et al.* [55] that evoked our interest in this problem. Our first result was a pseudo-polynomial algorithm for clock schedule verification [61]. This approach recognized that the long path equation set was equivalent to a Bellman-Ford relaxation (a technique commonly used to solve the problem of computing the shortest path in a graph). However it ignored the possibility of multiple solutions and had a drawback that could lead to incorrect results for the short path equations if the clock period was optimal. This was duly pointed out by Szymanski [70]. In order to correct the deficiencies of the pseudo-polynomial algorithm, we came up with the a sufficient condition for uniqueness of the late and early equation set. Szymanski had reached the same conclusions independently. In addition he demonstrated that the sufficient condition for uniqueness of the late equation set is also a necessary condition for the late equation set. More importantly if this condition was met, there would be at least one solution to the late equation set that violated a set-up constraint (Lemma 3.3.3 to Lemma 3.3.11 including Theorem 3.3.10). At the same time, Burks *et al.* [6] provided a detailed analysis of the reasons for clock schedule verification failure and explored efficient approaches to examine circuits with potential errors.

Chapter 4

Clock Schedule Optimization

We turn our attention to the problem of optimizing the clock period of a synchronous circuit. The goal is to compute the smallest clock period and assign instants of time to the clock events (rise and fall of phases) in the global time frame. This problem bridges the gap between analysis and synthesis of sequential circuits.

4.1 Overview

The optimal clock period problem for a single phase circuit with edge-triggered memory elements is the easiest case. The clock period is determined by the longest path consisting of combinational logic gates between a pair of flip-flops. In the case of multi-phase circuits with level-sensitive memory elements, the problem is complicated by the fact that data can stream through during the active intervals. In addition, the designer may choose to provide constraints on the duty cycle of the phases and/or require clock events to be separated. These are called external clocking constraints. The optimal clock schedule computation problem may be stated as:

Given a circuit $G(V, E, D, d)$, find the minimum clock period c and rise and fall times for each of the phases, so that the clock schedule meets all circuit clocking constraints and external clocking constraints.

The formulation in Table 2.2 involves max and min operators, which interact with one another. The approach suggested by Sakallah *et al.* [55] is to relax the min and max operators to a set of inequalities. The clock period optimization problem reduces to a linear program whose objective is to minimize c . A solution to the linear program is a lower bound on the clock period. The solution to the linear program is perturbed to obtain a solution to the original problem.

Szymanski presents two major contributions in [69]. Note that the formulation in Table 2.2 involves auxiliary variables; more precisely $4|V|$ variables (namely A_i, R_i, a_i, r_i at each vertex i) are used, in addition to the $2l$ variables (s_i, e_i for $i = 1, \dots, l$ and $e_l = c$) that represent the clocking scheme. The presence of auxiliary variables A_i, R_i, a_i and r_i serves little purpose, since they have to be assigned columns in a tableau (assuming the simplex [45] method is used to solve the linear program). The first cut is to eliminate these variables and pose the optimal clock formulation only in terms of the variables arising from the clocking scheme. A further refinement is to use the structure of the constraints to efficiently eliminate redundant constraints.

The organization of Chapter 4 is as follows. The clocking constraints are presented in an alternative form in Section 4.2. A compact representation for the constraints is the goal of Section 4.3. Section 4.4 presents two approaches to solve the clock schedule optimization problem. An example is given in Section 4.5 to illustrate the proposed techniques. Results on a set of benchmarks are also provided in the same Section.

4.2 Clocking constraints: a new form

Szymanski terms the original constraints in [54] as aggressive. He suggests modifying the early equation set by letting $r_i = s_{\phi(i)}^L$ for a level-sensitive memory element. This is called the conservative set of constraints. The motivation is two-fold. First, it has the effect of resulting in a feasible space that is convex, thus making it easier to minimize the clock period. The constraints can be reduced to a set of linear inequalities avoiding any need for perturbation of the linear program solution. Secondly, note that equations in Table 2.2 are true only in the steady state of circuit operation. A clocking scheme must be correct even in the start-up phase. The early arrivals at a memory element monotonically increase to the steady state values (see discussion in Section 3.3.3). Hold constraints enforce a lower bound on the early arrivals. Thus to ensure that the hold constraints are satisfied at all times it suffices to ensure that the early arrivals in the first period of operation meet the bounds. It is possible that the aggressive constraints lead to a smaller clock period but result in a clocking scheme with an error during the start-up period.

The idea of eliminating the auxiliary variables is conceptually elegant but leads to the question of formulating the clocking constraints without having to explicitly generate equations in Table 2.2. The conservative constraints described in [69] are shown in Table 4.1.

Denote a flip-flop by F and a latch by L. There are 4 types of paths in a latch graph G . A path $p : i_1 \rightsquigarrow i_q$ can be

Nature of constraint	Formulation
Data propagation Long path Short path	for $i = 1, \dots, V $ $A_i^L = \max_{j \in FI(i)} (R_j^L + D_{ji} - E_{\phi(j)\phi(i)})$ $a_i^L = \min_{j \in FI(i)} (r_j^L + d_{ji} - E_{\phi(j)\phi(i)})$
Memory element Late departure (latch) Late departure (flip-flop) Early departure (latch) Early departure (flip-flop)	for $i = 1, \dots, V $ $R_i^L = \max(A_i^L, s_{\phi(i)}^L)$ $R_i^L = c$ $r_i^L = s_{\phi(i)}^L$ $r_i^L = c$
Correct latching Set-up Hold	for $i = 1, \dots, V $ $A_i^L \leq c - S$ $a_i^L \geq H$

Table 4.1: Conservative clocking constraints

1. from a latch i_1 to a latch i_q (LL)
2. from a latch i_1 to a flip-flop i_q (LF)
3. from a flip-flop i_1 to a latch i_q (FL) or
4. from a flip-flop i_1 to a flip-flop i_q (FF).

A path *necessarily* terminates if it encounters a flip-flop. Consequently a flip-flop can only occur at the start or end of a path. Let the set of all such paths be denoted by $\mathcal{P}(G)$. A path is allowed to have repeated vertices; hence the set $\mathcal{P}(G)$ can be infinite. $\mathcal{P}(G)$ captures all possible paths along which data can (possibly) propagate unhindered by the opening and closing of memory elements.

For each edge $e_{ij} : i \rightarrow j$ define,

$$K_{ij} = \begin{cases} 0 & \text{if } \phi(i) \prec \phi(j) \\ 1 & \text{otherwise.} \end{cases} \quad (4.1)$$

Note that from Equation 2.2, we obtain

$$E_{ij} = e_j - e_i + K_{ij}c. \quad (4.2)$$

For a path $p : i_1 \rightsquigarrow i_q$ define,

$$K_{i_1 i_q}^p = \sum_{k=1}^{q-1} K_{i_k i_{k+1}} \quad (4.3)$$

and

$$D_{i_1 i_q}^p = \sum_{k=1}^{q-1} D_{i_k i_{k+1}}. \quad (4.4)$$

Equation 4.2 and Equation 4.3 yield the following fact.

Property 4.2.1 *Let $p : i_1 \rightsquigarrow i_q$ be a path in the latch graph, then $\sum_{k=1}^{q-1} E_{\phi(i_k)\phi(i_{k+1})} = e_{\phi(i_q)} + K_{i_1 i_q}^p c - e_{\phi(i_1)}$.*

For a path consisting of a single edge $i_1 \rightarrow i_2$, we drop the superscript p on $K_{i_1 i_2}^p$. For a path $p : i_1 \rightsquigarrow i_n$, $K_{i_1 i_n}^p$ keeps track of the number of clock periods available for computation along p .

Let path $p \in \mathcal{P}(G)$, $p : i_1 \rightsquigarrow i_q$. We introduce one inequality per path that depends on the nature of the path, and one inequality per edge (note that each edge is a valid path in $\mathcal{P}(G)$) as shown in Table 4.2. Constraints 1-2 are called the set-up (or long path) constraints and 3-4

Path type	Constraint
1: LL/LF	$e_{\phi(i_q)} \geq s_{\phi(i_1)} + D_{i_1 i_q}^p - K_{i_1 i_q}^p c + S$
2: FL/FF	$e_{\phi(i_q)} \geq e_{\phi(i_1)} + D_{i_1 i_q}^p - K_{i_1 i_q}^p c + S$
Edge type	Constraint
3: LL/LF	$e_{\phi(i_2)} \leq s_{\phi(i_1)} + d_{i_1 i_2} + (1 - K_{i_1 i_2})c - H$
4: FL/FF	$e_{\phi(i_2)} \leq e_{\phi(i_1)} + d_{i_1 i_2} + (1 - K_{i_1 i_2})c - H$

Table 4.2: Inequalities for correct clocking

are called the hold (or short path) constraints. The rest of the section is devoted to proving the *equivalence* of the sets of constraints in Table 4.1 and Table 4.2.

The following Lemma has been used by several researchers [24, 25, 69, 62]; the proof is borrowed from [69].

Lemma 4.2.1 *Let c be a feasible value of the clock period to constraints 1 and 2 in Table 4.2 over all paths $p \in \mathcal{P}(G)$. Let $C : i_1 \rightsquigarrow i_{p+1}$, ($i_1 = i_{p+1}$) be a cycle in the latch graph, then $D_{i_1 i_{p+1}}^C \leq K_{i_1 i_{p+1}}^C c$.*

Proof If there are any flip-flops lying on the cycle, a repeated application of constraint 2 in Table 4.2, will ensure $D_{i_1 i_{p+1}}^C - K_{i_1 i_{p+1}}^C c \leq 0$. So let us assume that C contains only level-sensitive latches. Let us construct a set of cycles, which are valid paths in $\mathcal{P}(G)$; define C^r to be a path which loops around

the cycle C , r times. Inequality 1 in Table 4.2 requires, $e_{\phi(i_1)} \geq s_{\phi(i_1)} + rD_{i_1 i_{p+1}}^C - rK_{i_1 i_{p+1}}^C c + S$. In other words,

$$c \geq \frac{s_{\phi(i_1)} - e_{\phi(i_1)} + S + rD_{i_1 i_{p+1}}^C}{rK_{i_1 i_{p+1}}^C}. \quad (4.5)$$

Letting $r \rightarrow \infty$, we see $c \geq \frac{D_{i_1 i_{p+1}}^C}{K_{i_1 i_{p+1}}^C}$. ■

Corollary 4.2.2 *Let c be a feasible value of the clock period to the constraint 1 and 2 in Table 4.2 over all paths $p \in \mathcal{P}(G)$, then the latch graph has no positive weight cycles.*

Proof Let $C : i_1 \rightsquigarrow i_{p+1}, (i_1 = i_{p+1})$ be a cycle in the latch graph, then $\sum_{k=1}^p \Delta_{i_k i_{k+1}} = \sum_{k=1}^p (D_{i_k i_{k+1}} - E_{\phi(i_k)\phi(i_{k+1})}) = \sum_{k=1}^p D_{i_k i_{k+1}} - \sum_{k=1}^p E_{\phi(i_k)\phi(i_{k+1})} = D_{i_1 i_{p+1}}^C - K_{i_1 i_{p+1}}^C c \leq 0$. ■

The ensuing theorem relates the clocking constraints in Table 4.1 to the inequalities in Table 4.2. Szymanski demonstrated[69] that the inequalities in Table 4.1 implied the inequalities in Table 4.2. We will provide a proof for the converse too.

Theorem 4.2.3 *The constraints in Table 4.1 have a solution if and only if the inequalities in Table 4.2 have a solution.*

Proof The proof consists of two parts.

- \Rightarrow) Let (A^L, R^L, a^L, r^L) and $(s_1, e_1, \dots, s_l, e_l)$ be a solution to Table 4.1. We show that for all $p : i_1 \rightsquigarrow i_q$ the constraints in Table 4.2 are satisfied.

- p is of type LL or LF.

Since the late equation set is unchanged, we use Lemma 3.2.1 to obtain

$$A_{i_q}^L \geq R_{i_1}^L + \sum_{k=1}^{q-1} (D_{i_k i_{k+1}} - E_{\phi(i_k)\phi(i_{k+1})}). \quad (4.6)$$

Together with $e_{\phi(i_q)}^L - S \geq A_{i_q}^L$, $R_{i_1}^L \geq s_{\phi(i_1)}^L$, Property 4.2.1, Equation 2.3 and Equation 2.4, we get

$$\begin{aligned} e_{\phi(i_q)}^L - S &\geq s_{\phi(i_1)}^L + D_{i_1 i_q}^p - e_{\phi(i_q)} - K_{i_1 i_q}^p c + e_{\phi(i_1)} \\ \Rightarrow c + e_{\phi(i_q)} &\geq s_{\phi(i_1)}^L + e_{\phi(i_1)} + D_{i_1 i_q}^p - K_{i_1 i_q}^p c + S \\ \Rightarrow e_l + e_{\phi(i_q)} &\geq s_{\phi(i_1)}^L + e_{\phi(i_1)} + D_{i_1 i_q}^p - K_{i_1 i_q}^p c + S \\ \Rightarrow e_{\phi(i_q)} &\geq (s_{\phi(i_1)}^L - e_l + e_{\phi(i_1)}) + D_{i_1 i_q}^p - K_{i_1 i_q}^p c + S \\ \Rightarrow e_{\phi(i_q)} &\geq s_{\phi(i_1)} + D_{i_1 i_q}^p - K_{i_1 i_q}^p c + S. \end{aligned}$$

This equivalent to constraint 1 in Table 4.2.

– p is of type FL or FF. Once again using Lemma 3.2.1

$$A_{i_q}^L \geq R_{i_1}^L + \sum_{k=1}^{q-1} (D_{i_k i_{k+1}} - E_{\phi(i_k)\phi(i_{k+1})}). \quad (4.7)$$

Together with $e_{\phi(i_q)}^L - S \geq A_{\phi(i_q)}^L$, $R_{\phi(i_1)}^L = e_{\phi(i_1)}^L$ (since i_1 is a flip-flop), Property 4.2.1 and Equation 2.3 we get

$$\begin{aligned} e_{\phi(i_q)}^L - S &\geq e_{\phi(i_1)}^L + D_{i_1 i_q}^p - e_{\phi(i_q)} - K_{i_1 i_q}^p c + e_{\phi(i_1)} \\ \Rightarrow c + e_{\phi(i_q)} &\geq c + e_{\phi(i_1)} + D_{i_1 i_q}^p - K_{i_1 i_q}^p c + S \\ \Rightarrow e_{\phi(i_q)} &\geq e_{\phi(i_1)} + D_{i_1 i_q}^p - K_{i_1 i_q}^p c + S \end{aligned}$$

This is equivalent to constraint 2 in Table 4.2.

It remains to prove the hold inequalities in Table 4.2 for each path consisting of a single edge $i_1 \rightarrow i_2$.

– LL or LF: For any edge $i_1 \rightarrow i_2$ we know the following constraints are true:

$$\begin{aligned} a_{i_2}^L &\leq r_{i_1}^L + d_{i_1 i_2} - E_{\phi(i_1)\phi(i_2)} \\ r_{i_1}^L &= s_{\phi(i_1)}^L \\ a_{i_2}^L &\geq H. \end{aligned}$$

Hence

$$\begin{aligned} H &\leq s_{\phi(i_1)}^L + d_{i_1 i_2} - (e_{\phi(i_2)} - e_{\phi(i_1)} + K_{i_1 i_2} c) \\ \Rightarrow H &\leq (s_{\phi(i_1)}^L + e_{\phi(i_1)} - e_{\phi(i_2)}) + d_{i_1 i_2} - K_{i_1 i_2} c \\ \Rightarrow e_{\phi(i_2)} &\leq (s_{\phi(i_1)}^L + e_{\phi(i_1)} - e_l) + d_{i_1 i_2} + e_l - K_{i_1 i_2} c - H \\ \Rightarrow e_{\phi(i_2)} &\leq s_{\phi(i_1)}^L + d_{i_1 i_2} + (1 - K_{i_1 i_2})c - H. \end{aligned}$$

This is equivalent to constraint 3 in Table 4.2.

– FL or FF: For any edge $i_1 \rightarrow i_2$ we know the following constraints are true:

$$\begin{aligned} a_{i_2}^L &\leq r_{i_1}^L + d_{i_1 i_2} - E_{\phi(i_1)\phi(i_2)} \\ r_{i_1}^L &= e_{\phi(i_1)}^L \\ a_{i_2}^L &\geq H. \end{aligned}$$

Hence

$$\begin{aligned}
H &\leq e_{\phi(i_1)}^L + d_{i_1 i_2} - (e_{\phi(i_2)} - e_{\phi(i_1)} + K_{i_1 i_2} c) \\
\Rightarrow H &\leq (e_{\phi(i_1)}^L + e_{\phi(i_1)}) + d_{i_1 i_2} - e_{\phi(i_2)} - K_{i_1 i_2} c \\
\Rightarrow e_{\phi(i_2)} &\leq (e_{\phi(i_1)}^L + e_{\phi(i_1)} - e_l) + d_{i_1 i_2} + e_l - K_{i_1 i_2} c - H \\
\Rightarrow e_{\phi(i_2)} &\leq e_{\phi(i_1)} + d_{i_1 i_2} + (1 - K_{i_1 i_2})c - H.
\end{aligned}$$

This is equivalent to constraint 4 in Table 4.2.

- \Leftrightarrow Conversely let $(s_1, e_1, \dots, s_l, e_l)$ be a solution to the inequalities in Table 4.2. We show that there exists a solution to the constraints in Table 4.1 by constructing an algorithm to calculate a set of values that satisfy them. The proof proceeds along similar lines to the proof of convergence of Procedure 3.2.1. An additional burden of demonstrating that the converged iterates indeed satisfy set-up and hold constraints lies with us.

Consider the iterative algorithm in Procedure 4.2.1. For notational convenience, we shall drop the superscript L on the local variables at a memory element and instead let the superscript m on each variable reflect the iteration number.

Procedure 4.2.1

1. Initialization-for all memory elements i

$$\begin{aligned}
A_i^0 &= -\infty \\
R_i^0 &= \begin{cases} c & \text{if } i \text{ is a flip-flop} \\ s_{\phi(i)}^L & \text{if } i \text{ is a latch} \end{cases}
\end{aligned}$$

2. Iteration-for $m = 1, \dots, n$ {

– for all memory elements i {

$$\begin{aligned}
* A_i^m &= \max_{j \in FI(i)} (R_j^{m-1} + \Delta_{ji}) \\
* R_i^m &= \begin{cases} c & \text{if } i \text{ is a flip-flop} \\ \max(A_i^m, s_{\phi(i)}^L) & \text{if } i \text{ is a latch} \end{cases}
\end{aligned}$$

}

}

3. Initialization-for all memory elements i

$$\begin{aligned}
a_i &= -\infty \\
r_i &= \begin{cases} c & \text{if } i \text{ is a flip-flop} \\ s_{\phi(i)}^L & \text{if } i \text{ is a latch} \end{cases}
\end{aligned}$$

4. for all memory elements i {
 - $a_i = \min_{j \in FI(i)} (\tau_j + \delta_{ji})$
 }

Steps 1 and 2 in Procedure 4.2.1 are identical to Procedure 3.2.1, and hence must converge in $n(= |V|)$ iterations, as long as there are no positive weight cycles in the latch graph (a true fact by virtue of corollary 4.2.2).

Steps 3 and 4 define the early arrivals and early departures at the memory elements. Note that the only iterations are in step 2. Let us denote the solution obtained from Procedure 4.2.1 by (A^n, R^n, a, τ) . It remains to check if (A^n, R^n, a, τ) meets the set-up and hold constraints (inequalities 2.12 and 2.13).

- Assume for sake for contradiction that there exists some memory element i_1 which has a set-up violation, $A_{i_1}^n > e_{\phi(i_1)}^L - S$. Trace a supporting path to i_1 as follows. Find $i_2 \in FI(i_1)$, such that $A_{i_1}^n = R_{i_2}^{n-1} + \Delta_{i_2 i_1}$. If $R_{i_2}^{n-1} = s_{\phi(i_2)}^L$ or if $R_{i_2}^{n-1} = c$, terminate the path, else continue tracing the path. Since the iterations are guaranteed to converge in $|V|$ iterations, a supporting path $p : i_q \rightsquigarrow i_1$ must be found, with $q \leq |V|$. If i_q is a latch we have $A_{i_1}^n = R_{i_q}^{n-q+1} + \sum_{k=1}^{q-1} \Delta_{i_{k+1} i_k}$ and $R_{i_q}^{n-q+1} = s_{\phi(i_q)}^L$ (if i_q is flip-flop we obtain $R_{i_q}^{n-q+1} = c$ and the proof proceeds along similar lines). So we have a path for which, $s_{\phi(i_q)}^L + \sum_{k=1}^{q-1} \Delta_{i_{k+1} i_k} > e_{\phi(i_1)}^L - S$. In other words,

$$\begin{aligned} s_{\phi(i_q)}^L + D_{i_q i_1}^p - K_{i_q i_1}^p c + e_{\phi(i_q)} - e_{\phi(i_1)} &> e_{\phi(i_1)}^L - S \\ \Rightarrow (s_{\phi(i_q)}^L - e_l + e_{\phi(i_q)}) + D_{i_q i_1}^p - K_{i_q i_1}^p c &> (e_{\phi(i_1)}^L - e_l + e_{\phi(i_1)}) - S \\ \Rightarrow s_{\phi(i_q)}^L + D_{i_q i_1}^p - K_{i_q i_1}^p c &> e_{\phi(i_1)} - S, \end{aligned}$$

thus contradicting constraint 1 in Table 4.2. If i_q is a flip-flop, a contradiction to constraint 2 in Table 4.2 is obtained.

- Assume for sake for contradiction that there exists some memory element i_1 which has a hold violation, $a_{i_1} < H$. Then from step 4, we conclude that there exists some $i_2 \in FI(i_1)$, such that $a_{i_1} = \tau_{i_2} + \delta_{i_2 i_1}$. Assume i_2 is a latch (the argument if i_2 is a flip-flop is similar). This implies

$$\begin{aligned} H &> \tau_{i_2} + \delta_{i_2 i_1} \\ \Rightarrow H &> s_{\phi(i_2)}^L + \delta_{i_2 i_1} \\ \Rightarrow H &> s_{\phi(i_2)}^L + d_{i_2 i_1} - e_{\phi(i_1)} + e_{\phi(i_2)} - K_{i_2 i_1} c. \end{aligned}$$

Rearranging terms, we obtain

$$\begin{aligned} e_{\phi(i_1)} &> (s_{\phi(i_2)}^L + e_{\phi(i_2)} - c) + (1 - K_{i_2 i_1})c + d_{i_2 i_1} - H \\ \Rightarrow e_{\phi(i_1)} &> s_{\phi(i_2)} + (1 - K_{i_2 i_1})c + d_{i_2 i_1} - H. \end{aligned}$$

Contradicting constraint 3 in Table 4.2. If i_2 is a flip-flop, a contradiction to constraint 4 in Table 4.2 is found. ■

4.3 Eliminating redundant constraints

This section is basically a reproduction of the theorems from [69], that are used to reduce the number of constraints in Table 4.2. Constraints 3 and 4 total up to $|E|$ constraints (one inequality per edge). Constraints 1 and 2, on the other hand, must hold for all paths, and possibly are infinite in number. Hence we must find a more compact representation. Consider constraints 1 and 2 in Table 4.2 for a path $p : u_1 \rightsquigarrow u_q$, ($\phi(u_1) = j$, $\phi(u_q) = i$). These can be represented in the form

$$x_i - x_j \geq \alpha_{u_1 u_q}^p - \beta_{u_1 u_q}^p c, \quad (4.8)$$

where $\alpha_{u_1 u_q}^p$ is defined as

$$\alpha_{u_1 u_q}^p = \sum_{k=1}^{q-1} D_{u_k u_{k+1}} + S \quad (4.9)$$

and

$$\beta_{u_1 u_q}^p = K_{u_1 u_q}^p. \quad (4.10)$$

x_j represents a clock event associated with the late departure from u_1 and x_i represents a clock event associated with the latching at u_q . Note $K_{u_1 u_q}^p \geq 0$.

Upto now, the dependence of constraints lay on a path from memory element u_1 to memory element u_q . However we are interested in the greatest lower bound on the difference $x_i - x_j$. To compute this we modify this dependence to include all paths that use the same clock events as p for late departure and latching. Let

$$U_{x_j x_i}^c = \max \left\{ \left(\alpha_{u_1 u_q}^p - \beta_{u_1 u_q}^p c \right) \left| \begin{array}{l} p : u_1 \rightsquigarrow u_q, \\ x_j \text{ triggers departure from } u_1 \\ x_i \text{ latches signal at } u_q. \end{array} \right. \right\} \quad (4.11)$$

For a given clock period c , $U_{x_j x_i}^c$ is the relevant bound on the minimum value $x_i - x_j$ can take.

Let Γ be the set of cycles in $\mathcal{P}(G)$. Recall that there must be at least one memory element in every cycle in the circuit. Consequently for any cycle $C : u_1 \rightsquigarrow u_{p+1}$, $K_{u_1 u_{p+1}}^C > 0$. A weak lower bound on the clock period c (from Lemma 4.2.1) is

$$c \geq \psi \equiv \max_{C \in \Gamma} \left(\frac{D_{u_1 u_{p+1}}^C}{K_{u_1 u_{p+1}}^C} \right). \quad (4.12)$$

Let $p' : u_1 \rightsquigarrow u_q$ be a path that contains a simple cycle $C : u_k \rightsquigarrow u_{k+v}$ ($1 < k \leq k+v < q$, and $u_k = u_{k+v}$), and $c \geq \psi$. Let us decompose p' into three paths:

1. $p^1 : u_1 \rightsquigarrow u_k$
2. $C : u_k \rightsquigarrow u_{k+v}$
3. $p^2 : u_{k+v}(=u_k) \rightsquigarrow u_q$

The contribution of p' to the term $U_{x_{\phi(u_1)} x_{\phi(u_q)}}^c$ is

$$(\alpha_{u_1 u_q}^{p'} - \beta_{u_1 u_q}^{p'} c) = (D_{u_1 u_k}^{p^1} - K_{u_1 u_k}^{p^1} c) + (D_{u_k u_{k+v}}^C - K_{u_k u_{k+v}}^C c) + (D_{u_{k+v} u_q}^{p^2} - K_{u_{k+v} u_q}^{p^2} c) + S.$$

Rearranging terms we get

$$(\alpha_{u_1 u_q}^{p'} - \beta_{u_1 u_q}^{p'} c) = [(D_{u_1 u_k}^{p^1} - K_{u_1 u_k}^{p^1} c) + (D_{u_{k+v} u_q}^{p^2} - K_{u_{k+v} u_q}^{p^2} c)] + (D_{u_k u_{k+v}}^C - K_{u_k u_{k+v}}^C c) + S.$$

Since $(D_{u_k u_{k+v}}^C - K_{u_k u_{k+v}}^C c) \leq 0$ (from Lemma 4.2.1),

$$\begin{aligned} (\alpha_{u_1 u_q}^{p'} - \beta_{u_1 u_q}^{p'} c) &\leq [(D_{u_1 u_k}^{p^1} - K_{u_1 u_k}^{p^1} c) + (D_{u_{k+v} u_q}^{p^2} - K_{u_{k+v} u_q}^{p^2} c)] + S \\ &\leq \alpha_{u_1 u_q}^{p^1 \cup p^2} - \beta_{u_1 u_q}^{p^1 \cup p^2} c. \end{aligned}$$

Thus the bound given by the cycle-disjoint path $p^1 \cup p^2$ dominates the bound provided by p' . So a simple way to compute $U_{x_j x_i}^c$ is

$$U_{x_j x_i}^c = \max \left\{ \left(\alpha_{u_1 u_q}^p - \beta_{u_1 u_q}^p c \right) \begin{array}{l} p : u_1 \rightsquigarrow u_q, p \text{ is cycle-disjoint} \\ x_j \text{ triggers departure from } u_1 \\ x_i \text{ latches signal at } u_q. \end{array} \right\} \quad (4.13)$$

Intuitively, if we encounter any cycle on a path from u_1 to u_q , we can only decrease $U_{x_{\phi(u_1)} x_{\phi(u_q)}}^c$. Also note that as c increases, $U_{x_j x_i}^c$ decreases. Thus any bound that is present at a clock period

$c > \psi$ must also be a bound at $c = \psi$. Using the lower bound ψ for the clock period, we obtain a sufficient set of constraints that need to be considered.

For a given clock period c , the relevant constraints may be obtained as follows. Each edge $i \rightarrow j$ in the graph G is weighted with $D_{ij} - K_{ij}c$. Solve an all pairs longest path problem (with the constraint that a path cannot continue if it encounters a flip-flop), given that the graph has no positive cycles. The length of the longest path from a memory element on phase j to a memory element on phase i gives us the constant on the right hand side of the inequality, denoted by $U_{x_j x_i}^c$. Since we have l phases, we can have at most $2l$ variables (rise and fall times). The value of $K_{u_1 u_q}^p$, for a cycle disjoint path $u_1 \rightsquigarrow u_q$, can range from 0 to $|V| - 1$. Hence there can be at most $(|V| - 1)2l(2l - 1) \sim O(|V|l^2)$ relevant constraints for the circuit. If we are given a clock period c this reduces to $O(l^2)$ constraints. To compute all relevant long path constraints for all valid clock periods, we use the approach suggested by Szymanski [69]. This has a complexity of $O(l|E||V|)$. This approach also requires the computation of ψ (the algorithm to do so has a complexity of $O(|V||E|b)$, where b is the number of bits of accuracy required in computing ψ). Henceforth, the set of relevant long path constraints will be denoted by $x_i - x_j \geq \alpha_{ji} - \beta_{ji}c$, since the dependence on paths in the graph has been captured.

The constraint 3 and 4 in Table 4.2 can be written as

$$x_i - x_j \leq \gamma_{u_1 u_2} + \eta_{u_1 u_2} c, \quad (4.14)$$

where

$$\gamma_{u_1 u_2} = d_{u_1 u_2} - H \quad (4.15)$$

and

$$\eta_{u_1 u_2} = 1 - K_{u_1 u_2}. \quad (4.16)$$

Once again we modify this dependence to include all edges that use the same clock events as $u_1 \rightarrow u_2$ for departure and latching. Recall that for an edge $(u_1 \rightarrow u_2) \in E$, $K_{u_1 u_2} \in \{0, 1\}$, implying $\eta_{u_1 u_2} \in \{0, 1\}$. Let

$$L_{x_j x_i}^c = \min \left\{ \left(\gamma_{u_1 u_2} + \eta_{u_1 u_2} c \right) \begin{array}{l} (u_1 \rightarrow u_2) \in E, \\ x_j \text{ triggers departure from } u_1 \\ x_i \text{ latches signal at } u_q. \end{array} \right\} \quad (4.17)$$

$L_{x_j x_i}^c$ decreases with decreasing c , since $\eta_{u_1 u_2} \geq 0$. With a lower bound on c , we get $L_{x_j x_i}^c \geq L_{x_j x_i}^\psi$ for all feasible clock periods. There are E constraints and $O(l^2)$ variables. Eliminating redundant

constraints will yield a set of $O(l^2)$ constraints. The set of relevant short path constraints will be denoted by $x_i - x_j \leq \gamma_{ji} + \eta_{ji}c$, since the dependence on edges in the graph has been captured. In all there are $O(|V|l^2)$ set-up and hold constraints, that are germane to the clock period optimization problem.

4.4 Solving the optimization problem

The problem formulation for clock period optimization may be posed in several equivalent forms. The linear program formulation is conceptually the simplest.

$$\begin{array}{ll}
 P : & \min(c) \\
 \text{such that} & x_j - x_i \leq -\alpha_{ji} + \beta_{ji}c \quad O(|V|l^2) \text{ long path constraints} \\
 & x_i - x_j \leq \gamma_{ji} + \delta_{ji}c \quad O(2l^2) \text{ short path constraints} \\
 & \left. \begin{array}{l} 0 \leq x_i \leq c \\ \psi \leq x_0 (= e_l) = c \end{array} \right\} \quad \text{upper and lower bounds}
 \end{array}$$

In this section we show that the linear program for clock period optimization has a special structure that makes it possible to solve it efficiently. The linear program formulation will be denoted by P .

We could solve this as a linear programming problem; instead we propose two approaches based on graph algorithms to solve the problem. If we fix the clock period to a value say $c = \pi$, then the size of the constraint set reduces to $O(l^2)$. This can be done by just evaluating the right hand sides with $c = \pi$ and picking the dominating bound for each inequality. This takes $O(|V|l^2)$. The constraints are then of the form

$$\begin{array}{ll}
 x_i - x_j \leq k_{ji}^{\pi} & O(l^2) \text{ long and short path constraints} \\
 x_i - x_0 \leq 0 & \left. \vphantom{x_i - x_j} \right\} \\
 0 \leq x_i & \text{upper and lower bounds} \\
 \psi \leq x_0 (= e_l) = \pi &
 \end{array}$$

Note that x_0 is a special variable representing the end of the global time frame, namely $e_l = c$.

4.4.1 A simple algorithm

We append the clock event separation constraints and ordering restrictions (if any) at this time. It should be pointed out that duty cycle constraints are of the form $mc \leq x_i - x_j \leq Mc$,

where $0 \leq m \leq M \leq 1$. Only constraints of the form $x_i - x_j \leq \mu c$, where μ is non-negative are permitted. The reason for excluding the negative μ will be clear towards the end of this section. Thus only a restricted set of duty cycle constraints (maximum duty cycle) are permitted. A feasibility check at $c = \pi$ can be done in polynomial time as explained below.

Theorem 4.4.1 *Given a clock period $\pi \geq \psi$, it is possible to check if there is a valid clocking scheme, and if so to find values for the rise and fall times of the l phases in $O(l^3)$ time.*

Proof It is well known that the feasibility of a set of constraints of the form $x_i - x_j \leq k_{ji}^\pi, k_{ji}^\pi \in R$, can be related to the shortest path on a graph problem. To check for feasibility we construct a constraint graph $G_p(V_p, E_p)$ as follows. For each variable x_i construct a vertex v_{p_i} . For each constraint $x_i - x_j \leq k_{ji}^\pi$, construct an edge from v_{p_j} to v_{p_i} with weight k_{ji}^π . Henceforth when we say “add an edge of weight w ”, we mean the following—if a previous edge exists we simply change its weight to be the minimum of the original weight and w . If no such edge exists we create a new edge of weight w in the graph. Add edges to all vertices $v_{p_i} (i \neq 0)$, from v_{p_0} with weight 0 (representing constraint $x_i \leq x_0$). Construct a zero vertex v_z (not to be confused with v_{p_0}) which has edges from all vertices other than v_{p_0} with weight zero (reflecting $x_i \geq 0$). Weigh the edge from v_{p_0} to v_z with $(-\pi)$. Add an edge from v_z to v_{p_0} of weight π ; thus ensuring $x_0 = \pi$. This construction makes the graph G_p strongly connected. From every vertex there is an edge to v_z . There is an edge from v_z to v_{p_0} , and there is an edge from v_{p_0} to all other vertices.

Initialize the potential of v_z to 0 and the potentials of all other nodes to $+\infty$. Now do a Bellman-Ford iteration for the shortest paths. If there is a negative cycle in G_p , it will be detected and such a cycle implies a set of inconsistent inequalities, implying infeasibility. Else, the algorithm will terminate with a set of consistent potentials for all vertices. The complexity is $O(l^3)$. If there are any upper bounds on variables, we initialize the potential of the vertex that represents that variable to the upper bound instead of $+\infty$.

If there is a negative cycle in the graph G_p , it implies that the constraints are infeasible. Let C_- be a negative cycle with weight $-W$ through, vertex v_{p_i} . This implies we have a constraint (after elimination from the set) $x_i \leq x_i - W$, i.e. $0 \leq -W$; clearly infeasible. ■

In order to do guarantee that binary search will find the optimum clock period, we need to prove two facts:

1. Convexity of the problem: If x^q, x^r are feasible solutions to the problem P with clock periods q and $r (q > r)$ respectively, then there exists a solution to all clock periods between r and q .

2. There is a tight upper bound on the clock period. Note that this implies that the upper bound is actually attained. So we have to show the existence of an upper bound Π and a feasible solution x^Π for $c = \Pi$.

Theorem 4.4.2 *Let x^q, x^r be feasible solutions to the constraints in P with clock periods q and r respectively ($q > r$). Let $c = \lambda q + (1 - \lambda)r$ ($\lambda \in [0, 1]$). Then $x = \lambda x^q + (1 - \lambda)x^r$ is a feasible solution to the constraints in P with clock period c .*

Proof Consider any constraint of the form $x_i - x_j \leq a + bc$, where a, b are real constants. We know

$$x_i^q - x_j^q \leq a + bq \text{ and}$$

$$x_i^r - x_j^r \leq a + br.$$

This implies, $\lambda x_i^q - \lambda x_j^q \leq \lambda a + \lambda bq$ and

$$(1 - \lambda)x_i^r - (1 - \lambda)x_j^r \leq (1 - \lambda)a + (1 - \lambda)br.$$

Adding the last two inequalities we get,

$$(\lambda x_i^q + (1 - \lambda)x_i^r) - (\lambda x_j^q + (1 - \lambda)x_j^r) \leq a + b(\lambda q + (1 - \lambda)r)$$

Hence $x = \lambda x^q + (1 - \lambda)x^r$ is feasible for clock period $c (= \lambda q + (1 - \lambda)r)$. ■

Theorem 4.4.3 $\Pi = |V|2l \max_{u_1 \rightarrow u_2 \in E} \{D_{u_1 u_2}\}$, is a tight upper bound on the clock period.

Proof We will give an algorithm to find Π . The short path constraints have a non-negative right hand side always. The long path constraints have a right hand side of the form $-\alpha_{ji} + \beta_{ji}c$ with β_{ji} , possibly equal to zero. We will first show that for every cycle in the constraint graph, the sum of the coefficients of c along the cycle must be a strictly positive value. We know that there are no negative coefficients of c on any of the edge weights. For sake of contradiction assume that there exists a cycle $C : v_1 \rightsquigarrow v_{n+1}$ in the constraint graph with zero coefficient sum for c . Let $e_i : v_i \rightarrow v_{i+1}$, $i = 1, \dots, n - 1$, denote the set of edges in C . Since the sum of coefficients of c , each of which is non-negative is zero, each edge must have a zero coefficient. Let $phase(v_i)$ denote the phase associated with the variable represented by vertex v_i (e.g. if vertex v_r represents the rise of phase k , namely s_k , then $phase(v_r) = \phi_k$). Now consider the nature of constraints that give rise to edges in G_p with zero coefficients of c .

- A long path constraint with a zero coefficient for c gives rise to an edge in the constraint graph e_i such that $phase(v_{i+1}) \prec phase(v_i)$ (Equations 1 and 2 in Table 4.2).
- A short path constraint with a zero coefficient for c gives rise to an edge in the constraint graph e_i such that $phase(v_{i+1}) \prec phase(v_i)$ (Equations 3 and 4 in Table 4.2).

This implies that as we traverse the cycle $phase(v_n) \prec phase(v_{n-1}) \cdots phase(v_2) \prec phase(v_1)$, irrespective of the nature of the constraint. Since the relation \prec (Chapter 2, Equation 2.1) is transitive, we conclude $phase(v_n) \prec phase(v_1)$. But there is an edge from v_n to v_1 , implying $phase(v_1) \prec phase(v_n)$. Also recall from definition that \prec is anti-reflexive, thus yielding a contradiction to the assumption.

The smallest value for the sum of coefficients of c in any cycle is 1. The largest value α_{ji} can have for an edge is $(|V| \max_{u_1 \rightarrow u_2 \in E} \{D_{u_1 u_2}\})$. A cycle can have at most $2l$ edges. So for a clock period $\Pi \geq 2l|V| \max_{u_1 \rightarrow u_2 \in E} \{D_{u_1 u_2}\}$ there will be no negative cycles in the graph G_p . Now carry out the Bellman-Ford iteration for $c = \Pi$. Since there are no negative cycles, we are guaranteed that the algorithm will converge to a valid solution. ■

Note that we never have to compute Π , only justify the existence of Π that depends on the delays of the gates in the circuit.

Lemma 4.4.4 *The complexity of binary search is $O((|V|l^2 + l^3) \log \Pi) \sim O(|V|l^2 \log |V|)$*

Proof The first term ($O(|V|l^2)$) is the complexity of selecting the minimum value of the right hand side k_{ji}^r for a given clock period π . The second term ($O(l^3)$) is due to the Bellman-Ford iteration in Theorem 4.4.1. If we normalize all numbers by $\max_{u_1 \rightarrow u_2 \in E} D_{u_1 u_2}$, then for $|V| \gg l$ we get $\log \Pi = \log(2l|V|) \sim O(\log |V|)$. Hence $O((|V|l^2 + l^3) \log \Pi) \sim O(|V|l^2 \log |V|)$. ■

A minimum duty cycle constraint will cause the coefficient for c (in the \leq inequality) to be negative, *i.e.* $(-m)$. Theorem 4.4.3 excludes the presence of such constraints. Consequently the bound in Lemma 4.4.4 does not hold in the presence of minimum phase separation constraints. In fact, the addition of these constraints may render the problem infeasible. Intuitively, a long path may force the clock period c to be of at least value π_1 ; this forces the on-time of a phase to be at least $m\pi_1$, and a short path may cause a violation.

We are guaranteed that the binary search will work only if there is no cycle for which the sum of the coefficients of c is negative. A simple case is when $m \in [0, \frac{1}{2l-1})$. The next section will present a generalized algorithm which will handle arbitrary duty cycle constraints.

4.4.2 A general algorithm

This algorithm is motivated by a technique used in linear programming which adds a constraint to the active set only when needed. We take advantage of the special structure of the constraints to find a feasible solution for a given clock period, if one exists. The general problem (denoted by GP) is of the form

$$\begin{aligned}
 GP: \quad & \min(c) \\
 \text{such that } & x_i - x_j \leq \min_{k=1, \dots, N}(a_{ji}^k + b_{ji}^k c) \\
 & \psi \leq c.
 \end{aligned}$$

The variables a_{ji}^k and b_{ji}^k are real numbers. We construct a constraint graph. $G_p(V_p, E_p)$ as described in Theorem 4.4.1. The general algorithm is given in Procedure 4.4.1.

Procedure 4.4.1

$c = \text{clock period}$

$\psi = \text{lower bound on } c$

$G_p(V_p, E_p) = \text{constraint graph}$

$c_0 = \psi$

$k = 0$

- *while (TRUE) {*
 1. $c = c_k$
 $\text{flag} = \text{check_constraints}(G_p, c)$
 2. *if (flag == ALL POSITIVE CYCLES) return TRUE (c is the optimum clock period)*
 3. *if (flag == NEGATIVE CYCLE)*
 $k = k + 1, c_k = \text{new_lower_bound}$
 4. *if (flag == INFEASIBLE) return FALSE (problem is infeasible)*
- }*

The routine *check_constraints()* for a given clock period can return one of three values:

1. *ALL POSITIVE CYCLES*: The set of constraints for the current clock period is feasible.

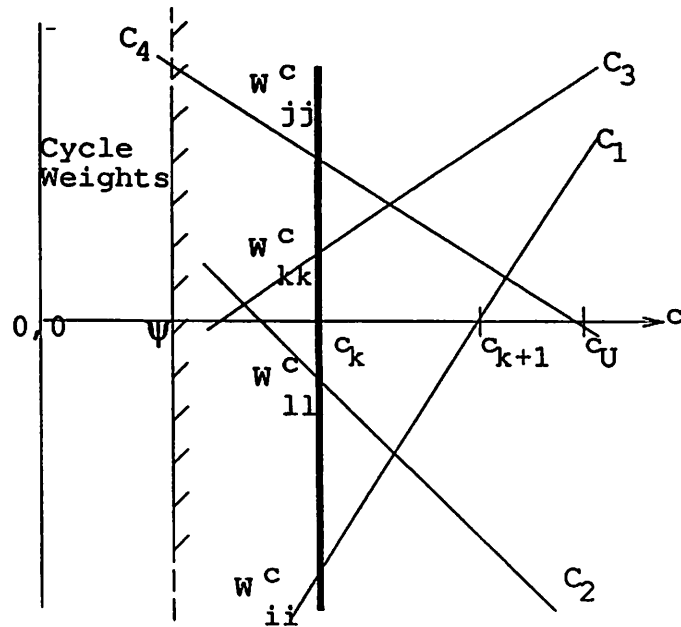


Figure 4.1: Graphical interpretation of the cycle weights

2. **NEGATIVE CYCLE:** The set of constraints for the current clock period is infeasible because at least one negative cycle exists in G_p . If a negative cycle is found, the routine also computes the value for the clock period in the next iteration.
3. **INFEASIBLE:** The problem is infeasible.

The search starts at the lower bound of the clock period. The routine *check-constraints()* evaluates the dominating constraint for each edge in E_p i.e. it computes $\min_{k=1, \dots, N}(a_{ji}^k + b_{ji}^k;c)$ and sets it as the edge weight. Floyd-Warshall [32] is used to detect the shortest path from x_p to x_q , keeping track of the sum of the b_{ij} 's for the shortest path. During the Floyd-Warshall iterations, we keep track of the diagonal entries of the Floyd-Warshall matrix. As soon as one of these entries becomes negative, we analyze all the cycles detected so far for each vertex. Let W_{ii}^c denote the minimum weight of a cycle if one exists from x_i to x_i at this time. Let B_{ii} denote the sum of the b_{ij} 's for the same cycle. There arise four cases as shown in Figure 4.1.

1. $W_{ii}^c < 0$ and $B_{ii} > 0$: Feasible clock periods must be greater than or equal to $c_k + \frac{-W_{ii}^c}{B_{ii}} = c_{k+1}$ (= *new_lower_bound*)(cycle C_1 in figure).

2. $W_{ii}^c < 0$ and $B_{ii} \leq 0$: The problem is infeasible because, for every clock period greater than c this cycle will have a negative value (cycle C_2 in figure).
3. $W_{ii}^c \geq 0$ and $B_{ii} \geq 0$: c is possibly feasible if $W_{jj}^c \geq 0$ holds for all vertices j and the Floyd-Warshall algorithm terminates with no negative cycles (cycle C_3 in figure).
4. $W_{ii}^c \geq 0$ and $B_{ii} < 0$: Feasible clock periods must be less than or equal to $c_k + \frac{W_{ii}^c}{-B_{ii}} = c_U$ (cycle C_4 in figure).

Note that the clock period being tested (c) is monotonically increasing. If we encounter a vertex i satisfying case 1, then we get a lower bound on the clock period. If vertex i satisfies case 2 then the problem is infeasible. Cases 3 and 4 do not give us any information regarding infeasibility because we are examining just one of the cycles in the graph (possibly prior to termination of the Floyd-Warshall iterations). The last case gives an upper bound on the clock period, which can be used to detect infeasibility early. The ensuing lemmas provide insight into the problem.

Lemma 4.4.5 *The number of cycles is of order $O((N + 1)^{|E_p|})$*

Proof Consider the different possibilities that an edge can partake in a cycle. Each edge is

1. not in the cycle,
2. or is in the cycle with one of the N possible weights on it;

$N + 1$ choices in all. Thus we have $O((N + 1)^{|E_p|})$ cycles. ■

For the optimal clocking problem, $|V_p| = 2l + 1$, $|E_p| \leq 2l(2l + 1)$, $N \leq |V| - 1$; thus the number of cycles is $O(|V|^{4l^2 + 2l})$. For the rest of the section we use the notation $\sum_C a_{ij}^k$ to represent the sum of the a_{ij} 's along a cycle $C \in G_p$, obtained from some value of k for each edge along C . Note that each edge can have a different value of k ; $1 \leq k \leq N$. Similarly, $\sum_C b_{ij}^k$ represents the sum of the b_{ij} 's for a cycle.

Lemma 4.4.6 *If for any $c (\geq \psi)$ there is a negative cycle through vertex i , such that $W_{ii}^c < 0$, and $B_{ii} \leq 0$ (for that cycle), then for all $c' \geq c$, there is a negative cycle through i .*

Proof Let us denote the negative cycle through i at clock period c as C_- . Let \hat{k} be the dominating constraint for each edge $i \rightarrow j$ in C_- at clock period c (i.e. $a_{ij}^{\hat{k}} + b_{ij}^{\hat{k}}c = \min_{k=1, \dots, N} (a_{ij}^k + b_{ij}^k c)$). Then

$$\min_{k=1, \dots, N} (a_{ij}^k + b_{ij}^k c') \leq a_{ij}^{\hat{k}} + b_{ij}^{\hat{k}} c'$$

$$\begin{aligned}
\sum_{C_-} (\min_{k=1, \dots, N} (a_{ij}^k + b_{ij}^k c')) &\leq \sum_{C_-} (a_{ij}^k + b_{ij}^k c') \\
&\leq \sum_{C_-} (a_{ij}^k + b_{ij}^k c) + b_{ij}^k (c' - c) \\
&< W_{ii}^c + B_{ii}(c' - c) \\
&< 0
\end{aligned}$$

Thus the weight of C_- for a clock period c' is negative. ■

Lemma 4.4.7 *If the problem GP is infeasible then \exists a cycle C in G_p , such that*

1. $\sum_C (b_{ij}^k)$ is strictly negative, or
2. $\sum_C (a_{ij}^k) < 0$ and $\sum_C (b_{ij}^k) = 0$.

Proof By contradiction. Suppose the above condition does not hold *i.e.*, for all cycles if $\sum_C (a_{ij}^k) < 0$, then $\sum_C (b_{ij}^k) > 0$, else $\sum_C (b_{ij}^k) \geq 0$. Then for a sufficiently large c , it is possible to make all cycles in the graph have strictly positive weight and a feasible solution to P can be found using Theorem 4.4.1. ■

Theorem 4.4.8 *The algorithm in Procedure 4.4.1 is complete, i.e. it finds an optimum solution if one exists, else it reports the problem as infeasible.*

Proof The proof is given graphically. We break the proof into two parts; in the first part we prove that the algorithm converges to a solution in a finite number of iterations, second we prove that the clock period is optimum.

Let c_k denote the value of the c in the k^{th} iteration, $k \geq 0$, $c_0 = \psi$. The proof relies on the fact that the number of cycles in the constraint graph is finite, though exponential in the number of constraints (see Lemma 4.4.5). Without loss of generality, assume that there are negative cycles (C_0, C_1, \dots, C_k) in the constraint graph for clock periods c_0, c_1, \dots, c_k . Note that cycle C_p is negative for all values of $c \in [c_p, c_{p+1})$. When the algorithm reports the existence of a negative cycle in iteration k , it either gives a value for c_{k+1} in the next iteration, or reports the problem to be infeasible. The proof for infeasibility is provided by a set of cycles such that the minimum weight of these cycles for all $c \geq \psi$ is strictly negative.

To show that the clock period reported in iteration n (if the problem is feasible) is optimum, we note that for all $c \in [\psi, c_k)$, we have a proof of infeasibility, namely a set of negative cycles. For $c = c_k$, there are no negative cycles, hence there is a solution to the x_i 's from Theorem 2. Intuitively

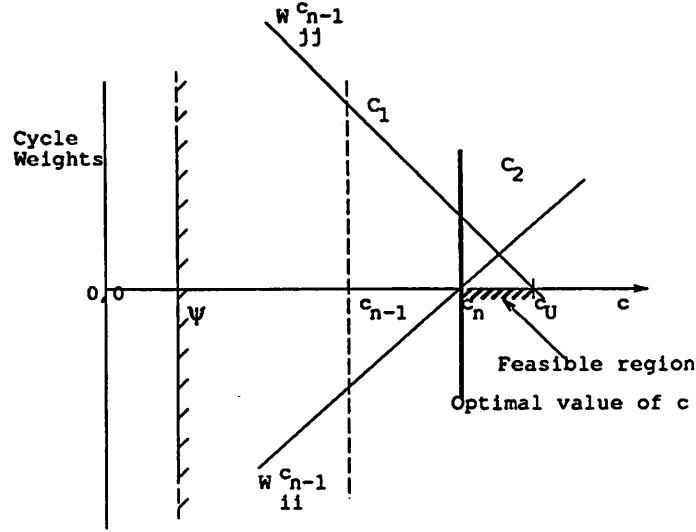


Figure 4.2: Graphical interpretation of optimality

the feasible region is the interval defined by the cycles determining the new upper and lower bounds on c (C_1 and C_2 in Figure 4.2). ■

Corollary 4.4.9 *Procedure 4.4.1 has a complexity of $O((l^2|V|^{4l^2+2l+1}))$*

Proof In each iteration there are two steps.

- Evaluating the dominating constraint of each edge (complexity $O(|E_p|N) \sim O(|V|l^2)$).
- Floyd-Warshall on the resulting graph requires $O(l^3)$ time.

Assuming $|V| \gg l$, the first process dominates the computation. In the worst case we examine every cycle in G_p leading to a complexity of $O(|E_p|N(N+1)^{|E_p|}) \sim O((l^2|V|^{4l^2+2l+1}))$ (see Lemma 3.2.9). ■

So for circuits with fixed number of phases, the complexity is polynomial in the size of the circuit. The value of N (β_{u_1, u_q}^p in Equation 4.8) reflects the depth of cycle stealing along paths in the circuit. For most circuits, $N \ll |V|$ (typically $N \sim 5$), leading to a fast algorithm for the linear program. Megiddo [44] presents an interesting approach to solve a similar problem. Note that we permit only those external constraints that can be put in the form required by GP . For example constraints of the form $e_i - s_i = e_j - s_j$, for all $i \neq j, i, j \in \{1, \dots, l\}$, cannot be expressed in the required manner. This constrains all phases to have active intervals of identical duration. Such constraints may require using a general linear program for the optimal clock schedule computation.

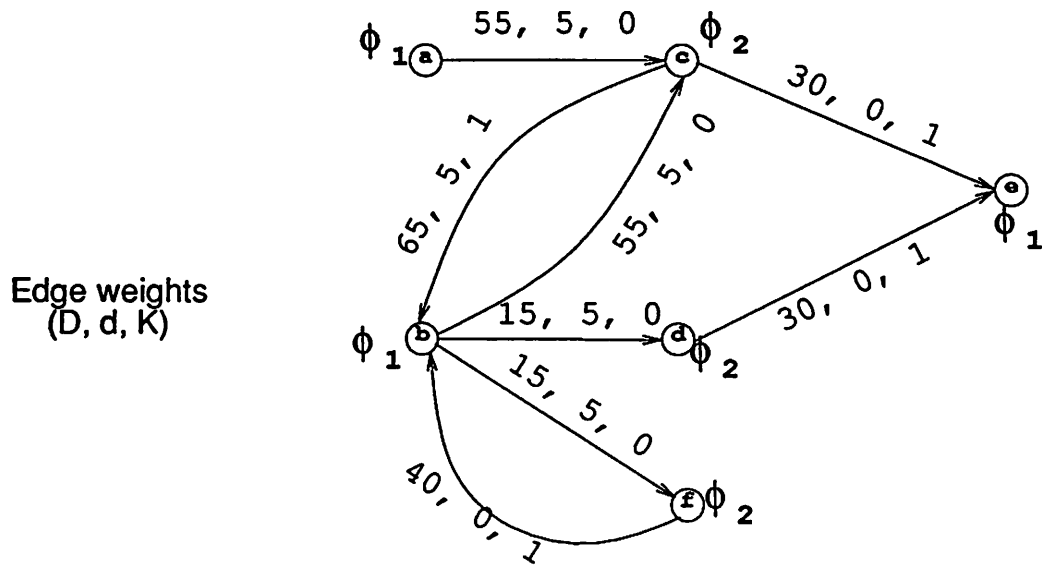


Figure 4.3: Latch graph for video coder

4.5 Results

Let us focus on the implications of the theoretical results obtained so far. In this section, we will discuss an example from [82] in detail and present the results of an implementation of the benchmarks described in Chapter 3 (Section 3.4.1).

4.5.1 An example

Consider the video coder circuit described in Section 2.3.4. The latch graph for the circuit is shown in Figure 4.3. There are two cycles in the latch graph, which yield the following lower bound on the clock period;

$$\begin{aligned} \psi &= \max \left(\frac{65 + 55}{1 + 0}, \frac{15 + 40}{0 + 1} \right) \\ &= 120. \end{aligned}$$

Let us first enumerate the long path constraints (for all cycle disjoint paths);

$$\begin{aligned} e_2 - s_1 &\geq 55 && \text{path } a \rightarrow c \\ &\geq 55 && \text{path } b \rightarrow c \\ &\geq 15 && \text{path } b \rightarrow d \end{aligned}$$

$$\begin{aligned}
& \geq 15 \quad \text{path } b \rightarrow f \\
& \geq 135 - c \quad \text{path } a \rightarrow c \rightarrow b \rightarrow d \\
& \geq 135 - c \quad \text{path } a \rightarrow c \rightarrow b \rightarrow f \\
e_1 - s_1 & \geq 120 - c \quad \text{path } a \rightarrow c \rightarrow b \\
& \geq 85 - c \quad \text{path } a \rightarrow c \rightarrow e \\
& \geq 85 - c \quad \text{path } b \rightarrow c \rightarrow e \\
& \geq 45 - c \quad \text{path } b \rightarrow d \rightarrow e \\
& \geq 165 - 2c \quad \text{path } a \rightarrow c \rightarrow b \rightarrow d \rightarrow e \\
e_1 - s_2 & \geq 65 - c \quad \text{path } c \rightarrow b \\
& \geq 40 - c \quad \text{path } f \rightarrow b \\
& \geq 30 - c \quad \text{path } d \rightarrow e \\
& \geq 30 - c \quad \text{path } c \rightarrow e \\
& \geq 110 - 2c \quad \text{path } c \rightarrow b \rightarrow d \rightarrow e \\
e_2 - s_2 & \geq 80 - c \quad \text{path } c \rightarrow b \rightarrow d \\
& \geq 80 - c \quad \text{path } c \rightarrow b \rightarrow f.
\end{aligned}$$

Quite clearly some paths dominate others, e.g. path $a \rightarrow c$ provides a bound (55) on the right hand side to the difference $e_2 - s_1$ that exceeds the bound provided by $b \rightarrow d$ (15). Eliminating such dominant constraints we get

$$\begin{aligned}
e_2 - s_1 & \geq 55 \quad \text{path } a \rightarrow c \\
& \geq 55 \quad \text{path } b \rightarrow c \\
& \geq 135 - c \quad \text{path } a \rightarrow c \rightarrow b \rightarrow d \\
& \geq 135 - c \quad \text{path } a \rightarrow c \rightarrow b \rightarrow f \\
e_1 - s_1 & \geq 120 - c \quad \text{path } a \rightarrow c \rightarrow b \\
& \geq 165 - 2c \quad \text{path } a \rightarrow c \rightarrow b \rightarrow d \rightarrow e \\
e_1 - s_2 & \geq 65 - c \quad \text{path } c \rightarrow b \\
& \geq 110 - 2c \quad \text{path } c \rightarrow b \rightarrow d \rightarrow e \\
e_2 - s_2 & \geq 80 - c \quad \text{path } c \rightarrow b \rightarrow d \\
& \geq 80 - c \quad \text{path } c \rightarrow b \rightarrow f.
\end{aligned}$$

Together with the constraint $c \geq \psi = 120$, we are able to eliminate some more constraints, e.g. $c \geq 120 \Rightarrow 135 - c \leq 15$ for all feasible c . Hence path $a \rightarrow c$, which provides a bound of 55 dominates $a \rightarrow c \rightarrow b \rightarrow d$ which gives a bound that has value no greater than 15. Keeping only the paths (a path when several provide the same bound) that dominate, the long path constraints are

$$\begin{aligned} e_2 - s_1 &\geq 55 \\ e_1 - s_1 &\geq 120 - c \\ e_1 - s_2 &\geq 65 - c \\ e_2 - s_2 &\geq 80 - c. \end{aligned}$$

The short path constraints are defined for each edge in the graph. They are

$$\begin{aligned} e_2 - s_1 &\leq 5 + c && \text{paths } a \rightarrow c, b \rightarrow d, b \rightarrow f, b \rightarrow c \\ e_1 - s_2 &\leq 0 && \text{paths } c \rightarrow e, d \rightarrow e, f \rightarrow b \\ &\leq 5 && \text{path } c \rightarrow b. \end{aligned}$$

The dominating paths are

$$\begin{aligned} e_2 - s_1 &\leq 5 + c \\ e_1 - s_2 &\leq 0. \end{aligned}$$

In addition let us assume that the minimum pulse width requirement on each clock phase is 20 units; yielding

$$\begin{aligned} e_1 &\geq s_1 + 20 \\ e_2 &\geq s_2 + 20. \end{aligned}$$

Furthermore, the designer requires that at least 60 units of time elapse after the fall of phase 1 and before the fall of phase 2. This gives

$$e_2 \geq e_1 + 60.$$

The model constraints require

$$e_1, s_1, s_2 \geq 0.$$

Consider the procedure described in Section 4.4.1 (the simple algorithm). The constraint graph is shown in Figure 4.4. Figure 4.5 shows the constraint graph with edge-weights evaluated at

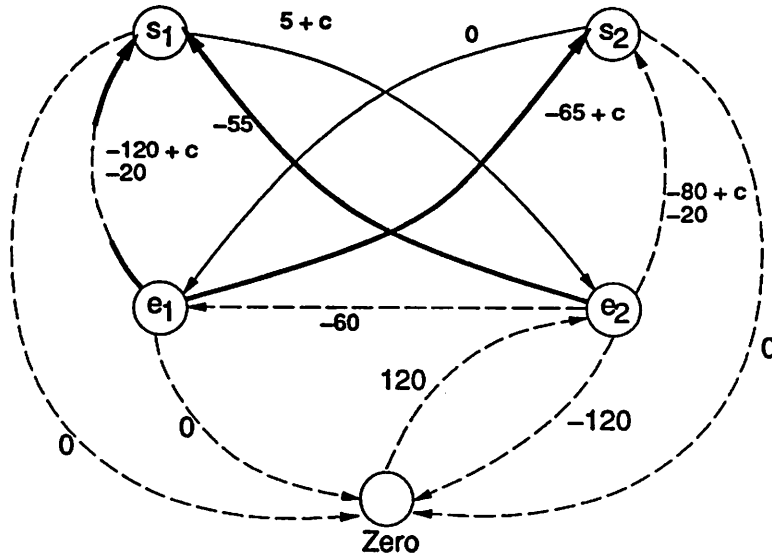


Figure 4.4: Constraint graph for video coder

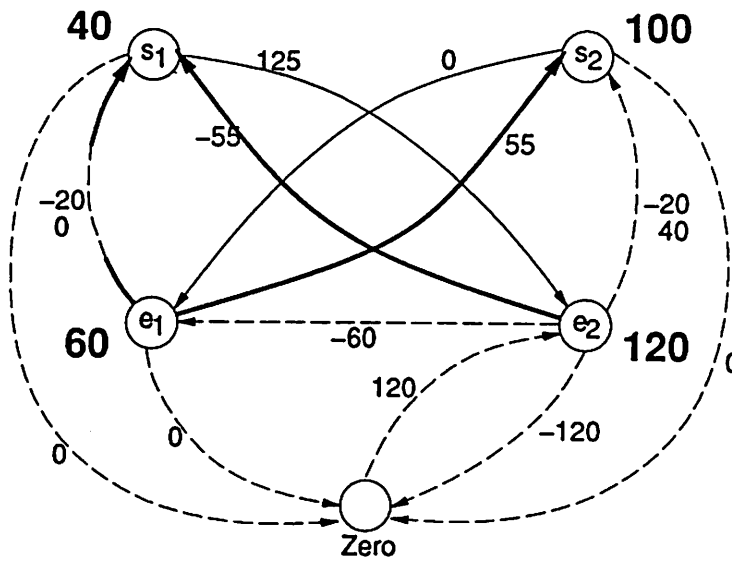


Figure 4.5: Constraint graph with edge weights evaluated at $c = 120$

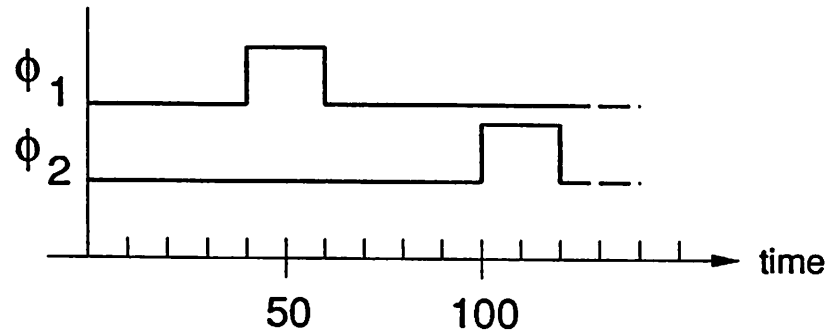


Figure 4.6: Clocking scheme for video coder

$c = \psi = 120$. Note that an edge can have more than one weight, in particular edge $e_1 \rightarrow s_1$ in the figure has 2 weights — the minimum weight is used for the shortest path algorithm. Initializing the zero vertex to 0 potential, the shortest path algorithm yields a solution (see Figure 4.6)

$$\begin{aligned} s_1 &= 40 \\ e_1 &= 60 \\ s_2 &= 100 \\ e_2 &= 120. \end{aligned}$$

To portray the complication caused by minimum duty cycle constraints, let us enforce a minimum active interval width for each phase to be $0.6c$. The constraints to do so are

$$\begin{aligned} s_1 - e_1 &\leq -0.6c \\ s_2 - e_2 &\leq -0.6c. \end{aligned}$$

Now consider the following set of constraints

$$\begin{aligned} s_1 - e_1 &\leq -0.6c && \text{minimum duty cycle for phase 1} \\ s_2 - e_2 &\leq -0.6c && \text{minimum duty cycle for phase 2} \\ e_2 - s_1 &\leq 5 + c && \text{short path constraint for edge } a \rightarrow c \\ e_1 - s_2 &\leq 0 && \text{short path constraint for edge } c \rightarrow e. \end{aligned}$$

Summing up, we get $0 \leq 5 - 0.2c$; in other words $c \leq 25$. This is inconsistent with the lower bound ψ . To see how the general algorithm (presented in Section 4.4.2) would detect this, consider the constraint graph modified to include the minimum duty cycle in Figure 4.7. A negative cycle is

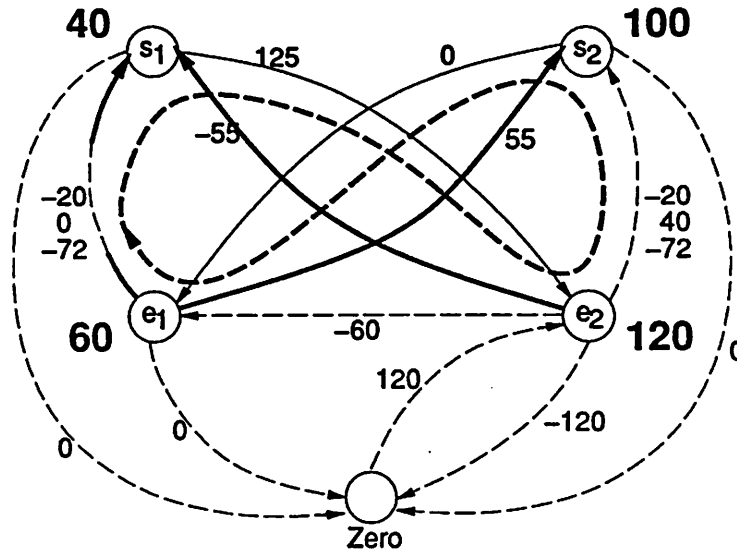


Figure 4.7: Constraint graph for video coder with negative cycle

encircled in bold dashes; its weight is $(-72 + 125 - 72 + 0) = -19$. The sum of the b_{ij} 's is -0.2 . Thus we have encountered the case where $W_{ii}^c < 0$ and $B_{ii} \leq 0$ (see Section 4.4.2, Para. 3).

4.5.2 Experiments

The construction of the benchmark circuits is outlined in Chapter 3 (Section 3.4.1). The results of the optimal clocking algorithms on the benchmarks are computed using two delay models. The first set of experiments is done using the unit delay fanout model (Table 4.3). Column 2 gives the time taken for reading the circuit and setting up the data structures. The lower bound on the clock period (ψ) and the time required for its computation (together with constraint generation) form the contents of columns 3 and 4. The optimum clock period is given in column 5 and the time taken for minimizing the linear program are shown in column 6 (using the simplified algorithm in Section 4.4.1) and column 7 (using the general algorithm in Section 4.4.2). The last column gives the total time taken by the implementation described by Szymanski [69], using a unit delay model. The first entry in the column is the time taken to process the circuit and the second entry reports the time for computing ψ , extracting the constraints and the best time required for solving the resulting linear program (amongst 3 linear program solvers). Szymanski[69] reports that no one linear program solver (of the 3) was uniformly the fastest; and that for occasional instances there was a factor of five between the best and worst linear program solvers. The circuits used

by Szymanski differ slightly (in terms of latch and gate count) since we choose to eliminate some redundant latches using a sequential optimizer. These results are for an implementation on a Silicon Graphics 4D/440 machine using a single processor (the specmark rating for the 4D/440 is 31.5 and for the DEC5000/125 is around 17). A “-” indicates that the results are not available.

name	read-in	ψ	time	optimal clock	time		time Szymanski [69]
	(sec.)		(sec.)		A1 (sec.)	A2 (sec.)	
2planet	1.29	56.00	0.01	56.00	0.01	0.01	-
2s1423	4.19	129.40	0.42	129.40	0.13	0.14	-
2s5378	5.25	32.60	0.51	32.60	0.12	0.14	-
2s9234	9.87	55.20	1.30	55.20	0.25	0.30	-
2s13207	14.31	54.60	1.94	54.60	0.61	0.65	-
2s38584	115.96	861.80	15.47	861.80	2.81	2.9	15.5 + 6.0
2s38417	148.90	240.40	24.27	240.40	5.80	5.93	8.6 + 4.1
2s35932	72.64	641.80	1.00	641.80	0.65	0.68	-

Table 4.3: Optimal clock computation with unit delay fanout model

A1 - simple algorithm

A2 - general algorithm

The second set of experiments uses the linear delay model prescribed by an industrial library (Table 4.4). The library has realistic gate delays and set-up hold times for the memory elements. The run times are similar to the results in Table 4.3. The duty cycle of each phase was constrained to lie between 0.3 and 0.5. Note that entries of column 2 includes the time taken to bind the circuit to the library.

The optimum clock period achieves the lower bound in all cases with the unit fanout delay model; whereas under the library delay model, there is a gap between the lower bound and the optimum clock period. The Floyd-Warshall skeleton (in column A2) detects a negative cycle early, if one exists. Consequently the iterations are never completed for lower (*i.e.* infeasible) clock periods. As a side-note, the implementation to compute the lower bound ψ makes use of a cycle-detection heuristic proposed by Szymanski[71]. The detailed procedure to compute ψ may be found in [69]. Briefly, a guess ψ' for the value of ψ is made and each edge in the latch graph is assigned a weight $D_{ij} - K_{ij}\psi'$. A positive cycle in the latch graph implies that the guess was an under-estimate for the bound. If all cycles are negative, the guess over-estimated the bound. A binary search will yield the value of ψ , whose accuracy is controlled by the number of searches,

name	read-in	ψ	time	optimal	time	
	(sec.)		(sec.)		clock	A1 (sec.)
2planet	12.99	32.67	0.03	35.77	0.01	0.01
2s1423	16.75	53.24	0.48	60.34	0.17	0.15
2s5378	32.04	19.21	0.75	27.86	0.20	0.17
2s9234	48.28	28.24	1.51	31.71	0.31	0.25
2s13207	64.15	29.26	0.22	34.74	0.77	0.67
2s38584	3672.93	210.76	16.23	218.41	3.29	3.21
2s38417	1014.82	97.78	2.25	97.79	6.11	5.93
2s35932	650.39	132.92	16.28	140.01	0.84	0.79

Table 4.4: Optimal clock computation with library delay model

that we are willing to undertake. If the value of ψ' is much larger than ψ , the longest path algorithm converges rapidly. At each iteration for the longest path, a priority queue is used to keep track of vertices that were affected in the previous iteration. A priority queue gives an improvement of a factor of 2 over the naive implementation. The problem arises when ψ' is smaller than ψ ; the failure to converge implies a positive cycle and this will require all iterations to be performed. Instead a predecessor pointer is maintained at every vertex, *i.e.* a parent of the vertex which caused an update of the vertex potential. By repeated traversal of the predecessor pointers, it is possible to detect a positive cycle cheaply. This heuristic alone yielded a speed-up of an order of magnitude on the large examples.

4.6 Discussion

So far the discussion has ignored the effect of clock skew on the clock schedule optimization problem. To incorporate clock skew, recall that the maximum (minimum) delay along a skew path P_i to latch i is given by $\sum_{k \in P_i} B_k^{P_i}$ ($\sum_{k \in P_i} b_k^{P_i}$). The clocking constraints translate as-

1. long path constraints

$$(a) \forall p : i_1 \rightsquigarrow i_q$$

$$e_{\phi(i_q)} + \left(\sum_{k \in P_{i_q}} b_k^{P_{i_q}} \right) \geq s_{\phi(i_1)} + \left(\sum_{k \in P_{i_1}} B_k^{P_{i_1}} \right) + \left(\sum_{k=1}^{q-1} (D_{kk+1}^p - K_{kk+1}^p c) \right) + S. \quad (4.18)$$

(b) \forall cycles $C : i_1 \rightarrow i_{q+1}$

$$c \geq \frac{\sum_{k=1}^q D_{kk+1}^C}{\sum_{k=1}^q K_{kk+1}^C} \quad (4.19)$$

2. *short path constraints* $\forall p : i_1 \rightarrow i_2$

$$e_{\phi(i_2)} + \left(\sum_{k \in P_{i_2}} B_k^{P_{i_2}} \right) + H \leq s_{\phi(i_1)} + \left(\sum_{k \in P_{i_1}} b_k^{P_{i_1}} \right) + d_{i_1 i_2} + (1 - K_{i_1 i_2})c. \quad (4.20)$$

These constraints can be put in the form described by *GP*.

This Chapter has focussed on computing the optimal clock schedule. Its fundamental contribution is the unraveling of the structure underlying the linear programming formulation of the problem. This enables us to develop an algorithm which is insensitive to the problem instance, *e.g.* we do not experience the non-uniform run times as reported by Szymanski. The mathematical relation between the complexity of solving the linear program and the size of circuit evolves as an artifact of understanding the problem structure. As we shall see (in the next Chapter), this also provides an insight into a technique to optimize a special class of sequential circuits (called pipelines) for high performance.

Several discussions with Szymanski on the clock schedule verification problem led to an exposure to the optimal clocking problem; in particular our curiosity was aroused by the non-convexity of the original model proposed by Sakallah *et al.* [55]. We chose the effort by Szymanski [72] that provides the linear programming formulation (whose merits have been summarized earlier), as the basis for our approach. By a curious co-incidence, a paper by Megiddo[44] triggered some ideas that led to exploring the structure of the linear program for the optimal clock schedule computation problem. Ishii *et al.* had reached similar conclusions for two-phase level-clocked circuits [25] independently. In contrast, our approach does not restrict the circuits to have a two-phase clocking methodology with only AHLs. In addition, we permit user defined duty cycle constraints and clock event separation constraints to be introduced in the formulation.

Chapter 5

Resynthesis of Multi-Phase Pipelines

The thesis has so far focused on analyzing a synchronous sequential circuit. Techniques of analysis are limited to drawing a designer's attention to potential violations of circuit specifications. The next logical question that begs to be answered is, whether synthesis techniques can be used to correct the design to meet specifications. A consequence of designing in the sequential domain is that constraints on clock period, duty cycle, and clock event separation specified by the user need to be satisfied. We call them external timing constraints; in contrast to the clocking constraints which arise from the circuit.

5.1 Overview

The external timing constraints translate to a set of performance constraints on one or more "pieces" of combinational logic. For circuits with single phase edge-triggered memory elements, it has been shown by Malik *et al.* [38] that the pipeline performance optimization problem is equivalent to a combinational performance problem; namely, the first problem has a solution if and only if the second problem has a solution. Bartlett *et al.* [2] propose an approach based on approximating level-sensitive latches by edge triggered flip-flops. This algorithm can handle arbitrary multi-phase circuits. A slack based algorithm is used to direct resynthesis and logic movement across memory elements repeatedly to find the best clock period at which the circuit can operate. The slack based approach algorithm is myopic in its optimization. To overcome this, simulated annealing is used to direct the optimization. However this may result in much larger circuits than necessary, especially when a target clock period is given.

Our approach differs from [2] in several ways.

1. We focus on the problem of satisfying a target clock period.
2. Only combinational regions are optimized, with no movement of logic across memory elements.
3. Flip-flops and latches are handled without any approximations.
4. We restrict the algorithm to acyclic pipeline circuits.

Section 5.2 provides some definitions that are specific to pipelines. In Section 5.3, we examine necessary and sufficient conditions for a circuit to operate at a specified clock period. An underlying theme of this research is to provide a means of directing traditional combinational delay optimizers to achieve the required performance. This is also termed combinational resynthesis since the circuit structure is changed without affecting the functionality. The effects of combinational resynthesis are studied in Section 5.4. In order to provide maximum flexibility for a combinational delay optimizer, steps are taken to obtain a minimum “perturbation” solution. This forms the contents of Section 5.5. We demonstrate how cycle stealing may be achieved in Section 5.6. The results of our discussion are presented in Section 5.7; the proposed algorithm is demonstrated on two simple examples in Section 5.7.1, and the implementation details are given in Section 5.7.2.

5.2 Definitions

We characterize a gate in our circuit with delays from an input pin to an output pin. This is a fixed delay model (see Section 2.1.1 for details). A multi-phase pipeline (P) consists of n stages of combinational logic separated by latches (see Figure 5.1). Each stage of combinational block has inputs from the previous stage (*i.e.* from the latches) and perhaps some inputs from the external world (called *primary inputs*). It has outputs feeding the next stage and outputs to the external world (called *primary outputs*). To simplify the discussion we make the following restrictions:

1. Stage S_1 has primary inputs through latches.
2. Stage S_n provides primary outputs from latches.
3. All other stages to have no primary inputs/outputs. The extension to handle primary inputs/outputs at intermediate stages is straightforward.

The u^{th} stage denoted by S_u has its inputs latched by a bank of latches denoted by u and its outputs latched by a bank of latches denoted by $u + 1$. The delay of each stage S_u is characterized by two

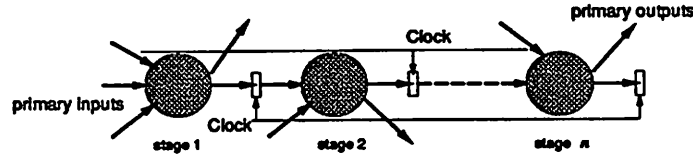


Figure 5.1: Multi-phase pipeline circuit

parameters, the longest delay (D_{uu+1} is the maximum sum of gate delays along a path from an input of stage u to an output of stage u) and the shortest delay (d_{uu+1} is the minimum sum of gate delays along a path from an input of stage u to an output of stage u) from an input to an output. We assume an implicit map $IM : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, l\}$ which maps an input of stage S_k to the phase that the input is latched. Similarly $OM : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, l\}$ maps the output of stage S_k to the phase that the output is latched. All inputs (outputs) of a stage are latched on the same phase. Also note $OM(k) = IM(k + 1)$.

The pipeline resynthesis problem for a target clock period can be stated as:

Given a pipeline P with n stages, using a clocking scheme with l phases, find an implementation that meets a given clock period constraint c .

The problem implicitly expects the rise and fall times for the phases to be determined consistent with the clocking constraints and the external timing constraints.

5.3 Theoretical results

We are interested in necessary and sufficient conditions for correct clocking of an arbitrary multi-phase circuit. The topological structure of a circuit and the distribution of delays within it give rise to the *clocking constraints*. A more detailed discussion on these constraints can be found in Chapter 4 (Section 4.2). The clocking constraints may be classified into two categories.

1. Long path constraints: Let $p : u_1 \rightsquigarrow u_q$ be a path with distinct latches u_1, \dots, u_q on it. Then we require $e_{\phi(u_q)} - s_{\phi(u_1)} \geq \sum_{k=1}^{q-1} (D_{u_k u_{k+1}} - K_{u_k u_{k+1}} c) + S$. These are also called the set-up constraints.
2. Short path constraints: For every path $p : u_1 \rightarrow u_2$, with latches u_1, u_2 and only combinational elements in between, we require $e_{\phi(u_2)} - s_{\phi(u_1)} \leq d_{u_1 u_2} + (1 - K_{u_1 u_2})c - H$. These are also called the hold constraints.

The set of constraints can be put in the form (see Section 4.4)

$$x_j - x_i \leq -\alpha_{ji} + \beta_{ji}c$$

$$x_i - x_j \leq \gamma_{ji} + \delta_{ji}c$$

$$0 \leq x_i \leq c$$

$$\psi \leq x_0 (= e_1) = c.$$

These constraints can be represented as a constraint graph, which leads to an efficient algorithm to compute the optimum clock period (c), given values to α_{ji} , β_{ji} , γ_{ji} and δ_{ji} . In this chapter we focus on the dual problem. We are given a target value of c ; the problem is to determine bounds on α 's (note that a bound on the α 's relates to bounds on D 's and d 's) for which the inequalities have a solution. There is an implicit constraint hidden in this formulation, namely $D_{kk+1} \geq d_{kk+1}$.

Let us make the following assumptions for ease of presentation :

1. The longest path from an input of stage S_i to an output of stage S_{j-1} , $j > i$, has a delay $\sum_{k=i}^{j-1} (D_{kk+1})$
2. All gates have identical rise and fall times.
3. The clock skew is negligible and all memory elements have identical set-up and identical hold times.
4. The shortest path from an input of stage to an output of the same stage (say S_k) is larger than the hold time H , i.e. $d_{kk+1} \geq H$. This is true because the hold time for most of the latches in present day technology is 0. Arguably, even if the hold time is a finite value, the combinational regions in most designs have short paths that satisfy this constraint.
5. The target clock period is greater than $S + H$.

The first assumption will be relaxed in section 5.6. The extension to include clock skew and different set-up/hold times is easy and will not be detailed.

We now present a graph structure that combines the circuit structure and the clock events. Recall that the latch graph G (Section 2.3.2) represents the paths between latches in a circuit. The constraint graph G_p (Section 4.4.2) captures the relevant constraints between clock events that arise from the latch graph. To compute bounds on path delays, a graph that has properties of both—the

latch graph and the constraint graph, is needed. The construction of this graph, denoted by \mathcal{G} , is now outlined.

The graph \mathcal{G} is a finite, edge-weighted, directed multi-graph $\mathcal{G} = (V, E)$. For every phase i , there are two vertices $s_i, e_i \in V$. Henceforth, whenever we say “add an edge of weight w from u to v ”, we mean the following: if no edge exists from u to v create an edge with weight w ; if such an edge exists, append w to the list of weights on the edge. An edge weight can be either a constant or a linear function of the clock period c . Stage S_k is split into two vertices $O(k)$ and $I(k)$ and an edge from the former to the latter (called a long path edge) with weight $-D_{kk+1} + K_{kk+1}c$, is added. There is an edge from $e_{\phi(OM(k))}$ to $O(k)$ with weight $-S$ and an edge from $I(k)$ to $s_{\phi(IM(k))}$ with weight 0. Place an edge (called a short path edge) from $s_{\phi(IM(k))}$ to $e_{\phi(OM(k))}$ with weight $d_{kk+1} - H + (1 - K_{kk+1})c$. In addition place edges of 0 weight from $I(k+1)$ to $O(k)$ ($k = 1, \dots, n-1$). To force the rise and fall times to be consistent with the clocking scheme, we construct a zero vertex (z). An edge from z to e_l (with weight c) and an edge from e_l to z (with weight $-c$) are added. Also edges from s_i ($i = 1, \dots, l-1$) and e_i ($i = 1, \dots, l-1$) to z (with weight 0) are added. Edges from e_i to s_i ($i = 1, \dots, l$) and from e_{i+1} to e_i ($i = 1, \dots, l-1$) with weight 0 are added. A portion of the graph for the k^{th} stage is shown in Figure 5.2.

Theorem 5.3.1 *The specified clock period c is feasible if and only if there is no negative cycle in \mathcal{G} .*

Proof The proof is similar to the proof of Theorem 4.4.1. The weight of an edge is defined as the minimum of its list of weights evaluated at c .

1. \Leftarrow) The if part relies on the manner in which \mathcal{G} was constructed. Each path from e_i (or s_i) to s_j (or e_j) represents a separation constraint. If there are no negative cycles, then the Bellman-Ford shortest path algorithm (with z initialized to 0), is guaranteed to converge to a solution. The values assigned by the Bellman-Ford algorithm to the e_i 's and s_i 's will yield a solution for the clocking scheme.
2. \Rightarrow) Conversely, if there is a negative cycle (of weight $-W$) in \mathcal{G} , say through vertex i . This implies we have a constraint of the form $0 \leq -W$, implying an inconsistent set of inequalities.

■

The condition for the non-existence of negative cycles leads to a set of linear inequalities between the D 's, d 's and c . In fact this is equivalent to eliminating the x_i 's (*i.e.* e_i, s_i) from the system of inequalities that need to be satisfied for correct clocking. Define a cycle to be a relevant

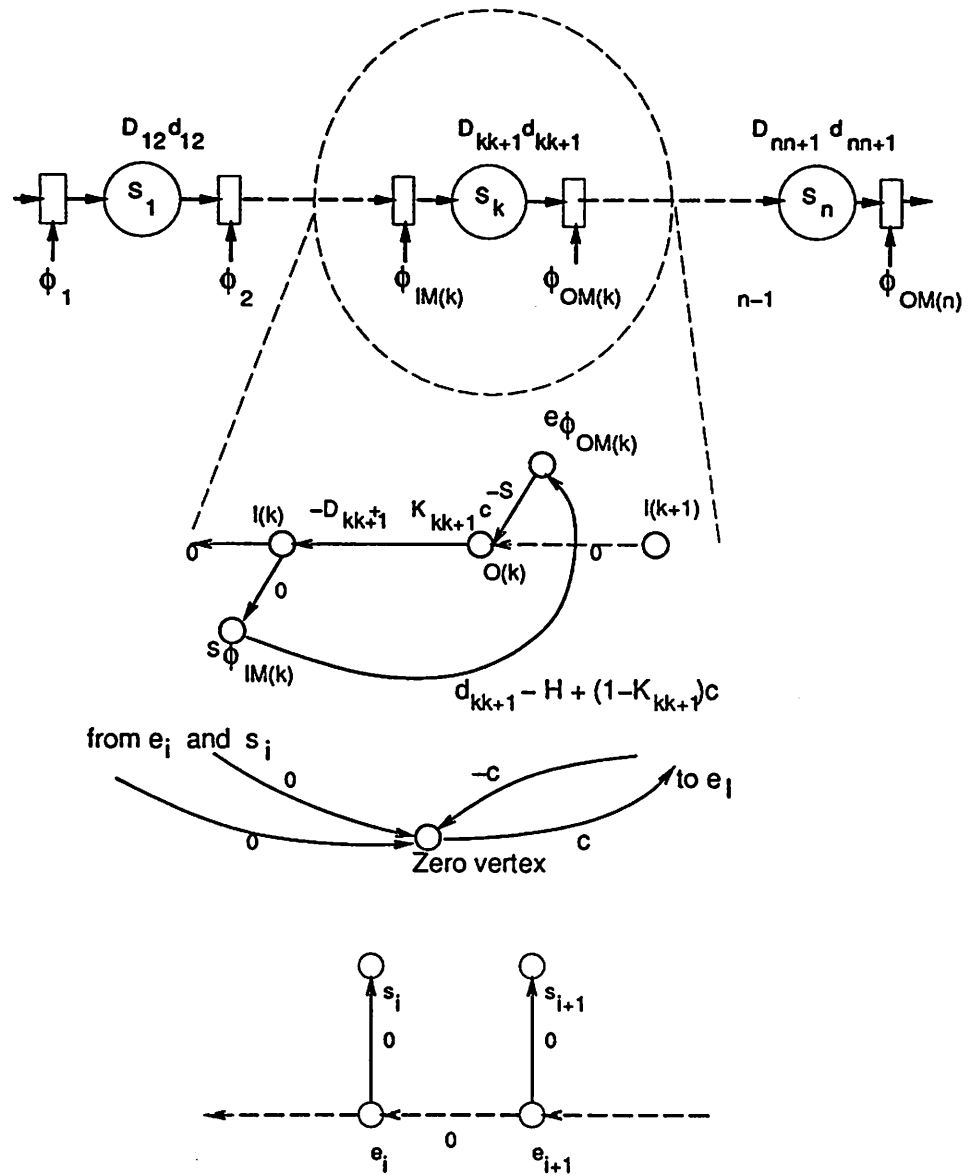


Figure 5.2: Graph construction

if its weight can be negative. An edge is called a **candidate** if it has a negative weight in its list of weights. Every relevant cycle must contain at least one candidate edge. Note that a *short path edge* can never be a candidate edge¹.

Lemma 5.3.2 *For an n stage pipeline, there are $O(n^2)$ relevant cycles.*

Proof The key to counting the number of cycles is realizing that there are two kinds of candidate edges ; from $O(u)$ to $I(u)$ and from $e_{\phi(u)}$ to $O(u)$. But any path containing a latter edge must pass through one or more edges of the former type. Since there is a zero weight path from $I(v)$ to $O(v - 1)$, all negative cycles must contain a path from $O(u)$ to $I(v)$ ($u \geq v$). Since we can choose u, v in $\frac{n(n+1)}{2}$ ways there are $O(n^2)$ cycles that need to be considered. ■

5.4 Resynthesis

To understand the implications of resynthesis on regions of the pipeline, we model the algorithm for combinational delay optimization as follows. Let \mathcal{R} denote the abstract algorithm which operates as follows.

1. \mathcal{R} takes as input, a piece of combinational logic \mathcal{C} , with arrival times specified at the inputs and required times specified at the outputs. The slack at each output is the difference between the required and the arrival time. If the slack is positive for all outputs then \mathcal{C} satisfies the performance constraint. If the slack is negative for an output then it is a critical output.
2. The output of \mathcal{R} , denoted by $\mathcal{R}(\mathcal{C})$, is a combinational circuit, logically equivalent to \mathcal{C} . \mathcal{R} will never cause an output with positive slack in \mathcal{C} to have a negative slack in $\mathcal{R}(\mathcal{C})$, *i.e.* it does a careful restructuring to guarantee that non-critical paths do not become critical.

Assuming equal rise and fall times for gates in the library, we seek to understand the effect of \mathcal{R} on long and short paths of a combinational region. Let D and d be the long and short paths in \mathcal{C} . We denote the long and short paths after resynthesis by primes (refer to Figure 5.3). The algorithm \mathcal{R} ensures that $D' < D$. If $d' \geq d$, there is no cause for concern because all the short path lengths appear as positive weights in \mathcal{G} , so increasing d can only help eliminate negative cycles. On the other hand if $d' < d$, we could possibly be introducing negative cycles. We make the assumption that delay insertion (by discrete amounts) is permitted so that the shortest path in $\mathcal{C}(\mathcal{C})$

¹Assumption 4 in section 5.3

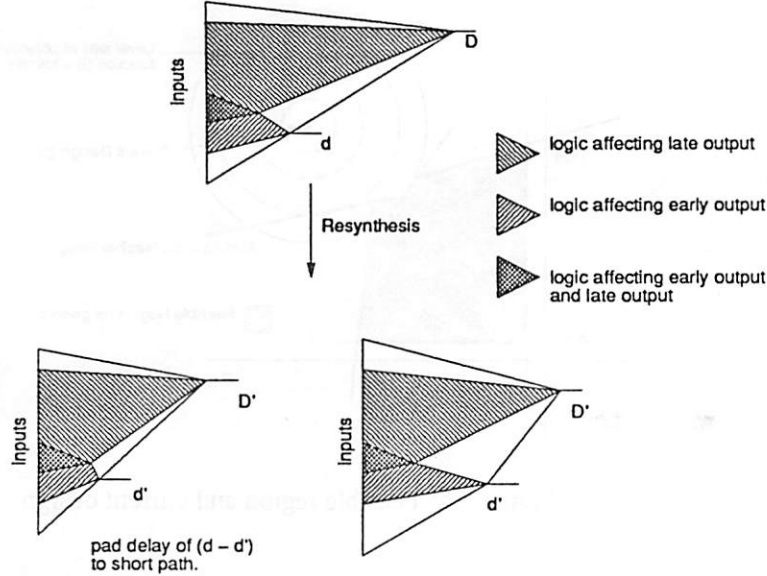


Figure 5.3: Effect of combinational optimization on long/short paths

is just greater than d . The problem of delay insertion to satisfy lower bounds on paths is the topic of concern in Chapter 6. However, this requires $d \leq D'$ to be consistent with our model. Note that this requirement only simplifies the discussion, *i.e.* we assume that the d for each stage is fixed to the value of the current implementation. We need to append the above constraint ($D'_{kk+1} \geq d_{kk+1}$ for each stage S_k)² to the set of constraints that force all cycles in \mathcal{G} to have non-negative weight. This implies that the target clock is never so small as to require speeding up short paths (if so we must let the short path delays be variables).

We have the following set of constraints

1. those arising from cycles in \mathcal{G} , and
2. $D'_{kk+1} \geq d_{kk+1}$, for $k = 1, \dots, n$.

This gives a polytope P in R^n , which lies in the positive orthant, and represents all feasible delay assignments that meet the target period. The given design is represented by the point $p = (D'_{12}, D'_{23}, \dots, D'_{nn+1})$ (see Figure 5.4). If the design is feasible, then the point $p \in P$; and there is no resynthesis to be done. Since $c > S + H$, we find that setting $D'_{ii+1} = d_{ii+1}$ is a feasible solution. This is a consequence of the fact that the feasible region is a cone with its vertex

²We can choose to let d'_{kk+1} be a variable rather than fix it to d_{kk+1} , and add the constraint $d'_{kk+1} \geq H$. After combinational optimization we need to pad delays so that all short paths are greater than d'_{kk+1} and less than D'_{kk+1} .

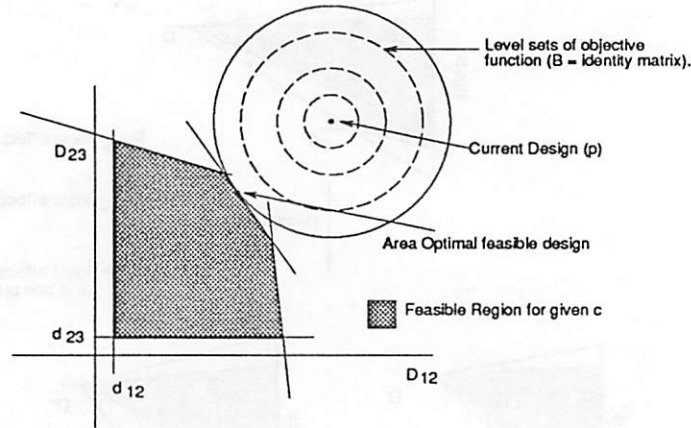


Figure 5.4: Feasible region and current design

at $(d_{12}, d_{23}, \dots, d_{nn+1})$. Let us assume that there is an area penalty associated with each speedup, which is directly proportional to the deviation from the current point p .

This suggests the formulation of the problem as an optimization problem. Let $D = (D_{12}, \dots, D_{nn+1})$ be a vector of known delays of the n stages. Let $D' = (D'_{11}, \dots, D'_{nn+1})$ be the vector of unknown target delays. We choose to find a D' which is a solution to

$$\begin{aligned} & \min (D' - D)^T (D' - D) \\ & \text{subject to } D' \in P \end{aligned}$$

This is an example of constrained optimization with a positive definite (convex) objective function over a convex region P . There are well known algorithms to solve this problem. We develop an algorithm tailored for our problem in Appendix B. Note that we could have chosen some other objective function like —

1.

$$\min_{D' \in P} \max_{k=1, \dots, n-1} |D'_{kk+1} - D_{kk+1}|$$

2.

$$\min_{D' \in P} \sum_{k=1}^{n-1} |D'_{kk+1} - D_{kk+1}|.$$

The reasons for choosing a quadratic objective function are twofold; such an objective function is easy to minimize and a unique optimum is guaranteed if the problem is feasible. If we had chosen to

let d'_{kk+1} 's be variables then they too must be entered in the objective function in a manner similar to the D'_{kk+1} 's.

5.5 The optimization problem

The optimization problem that we need to solve is of the form

$$QOPT : \min(x - x_0)^T B(x - x_0)$$

$$Ax \leq b.$$

The matrix B in the simplest form is the identity matrix. We may choose to weigh the combinational resynthesis of each stage S_k with a positive factor b_k . In such a case B is a diagonal matrix with $[B]_{kk} = b_k$. We point out that if the set P is feasible, then the problem has a unique optimum.

Theorem 5.5.1 *Any optimum solution to QOPT will have $x^i \leq x_0^i$ for $i = 1, \dots, n$.*

Proof By contradiction. Assume that there exists an optimum solution such that for some i , that $x^i > x_0^i$, i.e. $D'_{ii+1} > D_{ii+1}$. All constraints arising from the set of cycles in \mathcal{G} are of the form $\sum_{S_i \in C_-} (D'_{ii+1}) \leq w_{C_-}(c)$, where C_- is a cycle in \mathcal{G} that could have negative weight, and $w_{C_-}(c)$ is a linear function of c with positive slope. Since $D_{ii+1} \geq d_{ii+1}$, decreasing D'_{ii+1} to D_{ii+1} will decrease the objective function, while maintaining feasibility. ■

Once the optimization problem has been solved, we have $x^i < x_0^i$ or $x^i = x_0^i$. We need to resynthesize the the stages S_i for which $D'_{ii+1} < D_{ii+1}$. We set the arrival times at the inputs to stage S_i to be 0, and the required times at all the outputs to be D'_{ii+1} . After combinational resynthesis, we pad delays if there are any short path violations. We can have n such resynthesis steps (one for each stage). The resynthesized regions are mutually disjoint.

The simplest choice for B is the identity matrix. However, we may choose to weigh different stages of the pipeline depending on their resynthesis "potential". We define the potential of a stage as a figure which represents the ease of resynthesizing a stage to meet an arbitrary target delay. This number is computed as a heuristic function of the following.

1. The difference between the longest and shortest paths of the stage. For a stage with small difference, resynthesis for a target delay of the stage may cause all outputs to be critical.
2. The size of the stage. If the stage has a large size then the resynthesis algorithm has a better chance of identifying nodes on critical cuts [66].

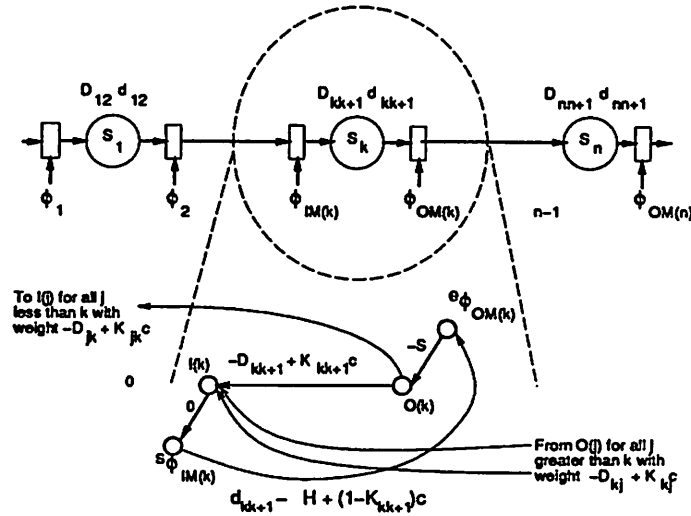


Figure 5.5: Graph modified for cycle stealing

The actual function used to compute this figure will depend to a large extent on the algorithm used for combinational resynthesis, and the factors that aid the particular algorithm. The resynthesis algorithm in our implementation is the path restructuring approach presented in [66].

5.6 Cycle stealing

So far it was assumed that the longest path in the circuit from (an input of) stage S_i to (an output of) stage S_{j-1} ($j > i$), was the sum of the longest paths in each of the stages. This need not be true, and in fact relaxing this assumption will permit resynthesis to take account of cycle stealing across several latches. However, we must be willing to undertake a complicated resynthesis procedure. Let D_{ij} denote the longest path from an input of stage S_i to an output of stage S_{j-1} . Let d_{ij} denote the shortest path from an input of stage S_i to an output of stage S_{j-1} . Note

$$\sum_{k=i}^{j-1} d_{kk+1} \leq d_{ij}$$

The construction of the graph \mathcal{G} is modified as follows (Figure 5.5). Note that the edges from $I(k)$ to $O(k-1)$ are deleted. Instead of n variables previously, we now have $\frac{n(n+1)}{2}$ variables. The objective function also changes appropriately, *i.e.*, the vector x in $QOPT$ is

$(D'_{12}, D'_{13}, \dots, D'_{jk}, \dots, D'_{nn+1})$, where $j < k$. We have additional constraints of the form

$$D'_{ij} \leq D'_{ik} + D'_{kj} \begin{cases} i = 1, \dots, n \\ i + 2 \leq j \leq n + 1 \\ i + 1 \leq k \leq j - 1 \end{cases} \quad (5.1)$$

The number of edges in the graph increases but the number of cycles remains unchanged.

Lemma 5.6.1 $D'_{ij} \leq \sum_{k=i}^{j-1} D'_{kk+1}$, $j > i + 1$.

Proof By induction on j .

1. Base case: $j = i + 2$. Quite clearly $D'_{ii+2} \leq D'_{ii+1} + D'_{i+1i+2}$, from Equation 5.1.
2. Induction case: Assume the statement to be true for $j = p$, i.e. $D'_{ip} \leq \sum_{k=i}^{p-1} D'_{kk+1}$. Now from Equation 5.1 we have

$$\begin{aligned} D'_{ip+1} &\leq D'_{ip} + D'_{pp+1} \\ \Rightarrow D'_{ip+1} &\leq \sum_{k=i}^{p-1} D'_{kk+1} + D'_{pp+1} \\ \Rightarrow D'_{ip+1} &\leq \sum_{k=i}^p D'_{kk+1}. \end{aligned}$$

■

In order to guarantee that short path delays remain consistent after padding delays if any, we need to add the constraints

$$D'_{ij} \geq D'_{ik} + d_{kj} \begin{cases} i = 1, \dots, n \\ i + 2 \leq j \leq n + 1 \\ i + 1 \leq k \leq j - 1 \end{cases} \quad (5.2)$$

$$D'_{ii+1} \geq d_{ii+1} \quad i = 1, \dots, n \quad (5.3)$$

The feasible region is a polytope in the positive orthant of the D'_{ij} 's, but it is no longer a cone, e.g. consider the set of constraints—

$$\begin{aligned} d_{12} &= d_{23} = 0.2 \\ D'_{13} &\geq D'_{12} + 0.2, D'_{12} + 0.2 \end{aligned}$$

$$\begin{aligned}
D'_{13} &\leq D'_{12} + D'_{13} \\
D'_{12}, D'_{23} &\geq 0.2 \\
d_{13} &= 0.6 \\
D'_{13} &\geq 0.6.
\end{aligned}$$

If the last 2 constraints are ignored, we have a cone with the vertex at $(D'_{12} = 0.2, D'_{23} = 0.2, D'_{13} = 0.4)$. In order to make the feasibility check easier we expand the feasible region as follows. Let

$$\hat{d}_{ij} = \begin{cases} d_{ii+1} & \text{if } j = i + 1 \\ \sum_{k=i}^{j-1} d_{kk+1} & \text{otherwise} \end{cases} \quad (5.4)$$

and replace the constraints in 5.2 by

$$D'_{ij} \geq D'_{ik} + \hat{d}_{kj}.$$

Note that $\hat{d}_{ij} \leq d_{ij}$, so we have enlarged the feasible region to be a cone, with its vertex defined by the point $(\hat{d}_{12}, \hat{d}_{23}, \dots, \hat{d}_{nn+1})$. This serves the purpose as $D'_{kk+1} \geq d_{kk+1}$ in the simplified model. As before, we could choose to let d_{ij} 's be variables and add the constraints

$$d'_{ij} \geq (j - i)H$$

The resynthesis is now done for each stage with arrival and required times placed on its inputs and outputs. The resynthesis is done in a "forward" manner, *i.e.* stage S_k is resynthesized before stage S_{k+1} . Let D''_{ij} denote the longest path from an input of stage S_i to an output of stage S_{j-1} , after stage S_{j-1} (and all stages $S_i, i < j$) has been resynthesized. We set the arrival times at the inputs to stage S_k to

$$a_x = 0 \quad \forall x \in I(k) \quad (5.5)$$

Assume that stage S_k has a single output y . The extension to multiple outputs is easy and will be shown at the end of this section. We shall specify a pair of required times for each output; a hard required time (r_y^h) and a soft (r_y^s) required time. The latter is less than the former. If the circuit is to operate correctly it *must* meet the hard required time. However we would like the current output of stage S_k to attain the soft required time, in order to help the resynthesis of stages that follow it. The required times at the output of stage k are set to

$$r_y^h = \min(D'_{kk+1}, \min_{i < k} (D'_{ik+1} - D''_{ik})) \quad (5.6)$$

$$r_y^s = \min_{j > k+1} (D'_{kj} - \sum_{i=k+1}^{j-1} \min(\hat{D}_{ii+1}, D'_{ii+1})) \quad (5.7)$$

The hard required time is a function of

1. the maximum delay permitted through stage k (first term), and
2. the effect of the stages that have been resynthesized (second term)

The soft required time uses an *as fast as possible* heuristic to ease the resynthesis of future stages. The required time to the delay optimizer is r_y^s . The output of the optimizer is accepted if the hard constraint is met. For the last stage S_n , r_y^s is undefined, since there is no $j > n + 1$ and consequently, there are only hard constraints. This guarantees the following after resynthesis:

1. From the first term for r_y^h we get $D''_{kk+1} \leq D'_{kk+1}$.
2. The second term in the required time constraint for r_y^h yields

$$\begin{aligned} D''_{kk+1} &\leq (D'_{ik+1} - D''_{ik}) \quad \forall i < k \\ \Rightarrow D''_{kk+1} + D''_{ik} &\leq D'_{ik+1} \\ \Rightarrow D''_{ik+1} &\leq D'_{ik+1}. \end{aligned}$$

3. The intuitive reason for defining r_y^s is now explained. This term distributes slacks over the regions that have not yet been resynthesized and hence is a heuristic.

$$\begin{aligned} D''_{kk+1} &\leq (D'_{kj} - \sum_{i=k+1}^{j-1} \min(\bar{D}_{ii+1}, D'_{ii+1})), \quad \forall j > k + 1 \\ \Rightarrow D''_{kk+1} + \sum_{i=k+1}^{j-1} \min(\bar{D}_{ii+1}, D'_{ii+1}) &\leq D'_{kj}. \end{aligned}$$

The term \bar{D}_{ii+1} in the min constraint is the longest path in stage S_i for any path from an input of stage S_k to an output of stage S_{j-1} . If $\bar{D}_{ii+1} < D'_{ii+1}$ then this stage is not critical for the constraint during the resynthesis of stage S_k . We assume (optimistically) that resynthesis of stage S_i will not make it critical. Should it become critical later, the second term will reflect it. If $D'_{ii+1} < \bar{D}_{ii+1}$ then the first term during resynthesis of stage S_i will guarantee that its final delay is less than D'_{ii+1} .

$$\begin{aligned} D''_{kk+1} + D''_{k+1j} &\leq D'_{kj} \\ \Downarrow \\ D''_{kj} &\leq D'_{kj}. \end{aligned}$$

We get the following proposition—

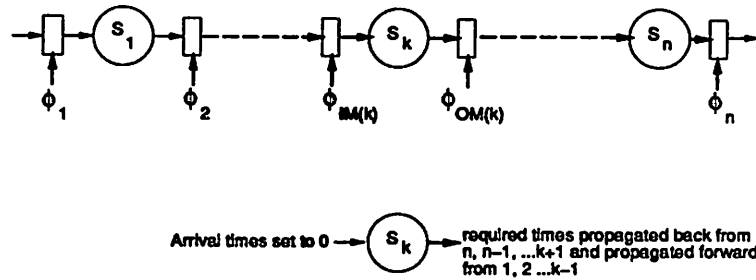


Figure 5.6: k^{th} resynthesis Region

Proposition 5.6.2 *If the required time constraints (Equation 5.6 and Equation 5.7) are met for all stages, then $D''_{kj} \leq D'_{kj}$ and $D''_{ik+1} \leq D'_{ik+1}$, for all $k, j > k + 1$ and $i < k$.*

The resulting circuit (after resynthesis) will have a feasible clock period c . As a side note, should the combinational re-synthesis fail at any stage, say S_j , we can repeat the optimization problem, giving the cost of resynthesizing stage S_j a large weight. This ability to restart may be used repeatedly to find a good final solution.

To extend the discussion above to stages with multiple outputs, we must take care to ensure that D''_{ik} only includes the delay along paths from an input of previous stages S_i that reach the output of S_k in question. Similarly \bar{D}_{ii+1} must be interpreted as the longest path in stage S_i from the output of S_k to an output of S_{j-1} .

5.7 Results

5.7.1 An example

Let us consider an example shown in Figure 5.7. Set-up and hold times are assumed to be zero. Let x and y be bounds for stage 1 and stage 2 respectively. All un-weighted edges have a weight of 0 by default. Let the target clock period be 3. Enumerating all cycles that could have negative weight we obtain

$$\begin{aligned} x &\leq 3 \\ x + y &\leq 6 \\ y &\leq 6 \\ x &\leq 4 \end{aligned}$$

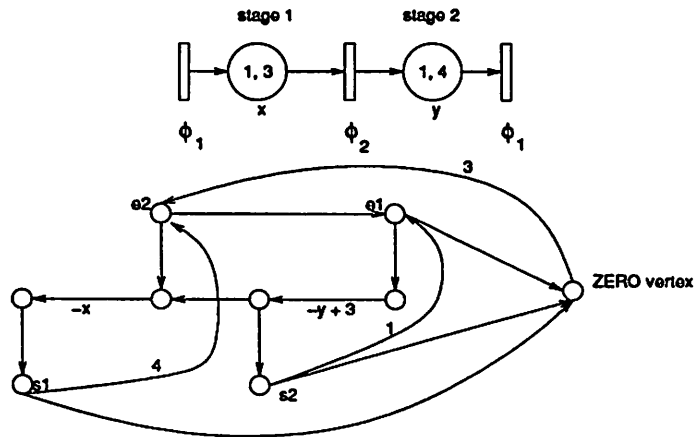


Figure 5.7: Pipeline example and associated graph

$$y \leq 4.$$

The constraint $D'_i \geq d_i$ leads to

$$x \geq 1$$

$$y \geq 1.$$

The objective function is $\min((x - 3)^2 + (y - 4)^2)$. This problem can be graphically solved (see Figure 5.8) to obtain a solution of $x = 2.5, y = 3.5$. If stage 1 has a cost which is twice the cost of stage 2, the objective function gets modified to $\min(2(x - 3)^2 + (y - 4)^2)$. The minimum of this function is obtained at $x = \frac{8}{3}, y = \frac{10}{3}$.

To see the need for the extended model, consider the pipeline with multiple outputs (Lines 1 and 2) for each stage as shown in Figure 5.9. We introduce a new variable z (longest delay from stage 1 to stage 2) and the constraint graph gets modified as shown in Figure 5.9. Note that D_{23} has a value 4 if we are along Line 2 and value 2 if we are along Line 1. Though we need to use the value of 4 for the quadratic optimization, we must take care to use the correct value (of 2) when we compute the required time at the output of Stage 1 in Line 1. If the current longest path from stage 1 to stage 2 is 5 (instead of 7, as in the simplified model), the constraints for a clock period of 3 become,

$$x \leq 3$$

$$z \leq 6$$

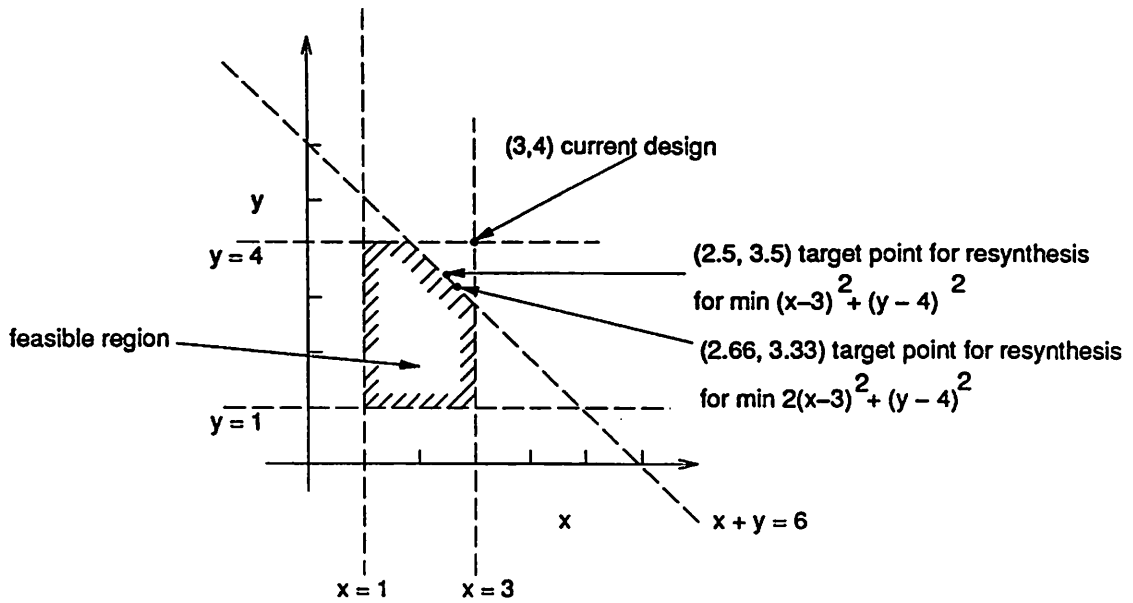


Figure 5.8: Graphical solution for example

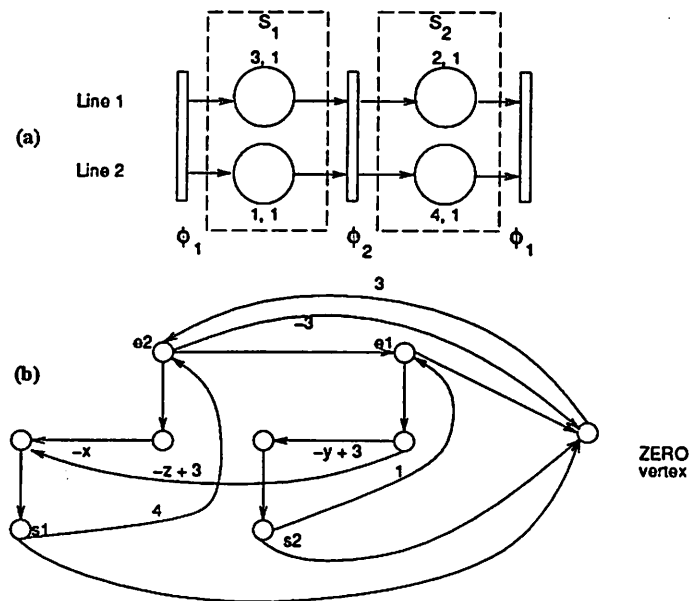


Figure 5.9: Example for extended model

$$\begin{aligned}
 y &\leq 6 \\
 x &\leq 4 \\
 y &\leq 4 \\
 z - x - y &\leq 0 \\
 x &\geq 1 \\
 y &\geq 1 \\
 z &\geq x + 1.
 \end{aligned}$$

The objective function is $\min((x - 3)^2 + (y - 4)^2 + (z - 5)^2)$. The optimum value is 0 and the solution point is (3, 4, 5), *i.e.* the current design is feasible. For a circuit with the longest delay from stage 1 to stage 2 less than the sum of the longest individual stage delays in the simplified model, the target delays would have been unnecessarily computed. Let us use a target clock period of 2.5; the constraints become

$$\begin{aligned}
 x &\leq 2.5 \\
 z &\leq 5 \\
 y &\leq 5 \\
 x &\leq 3.5 \\
 y &\leq 3.5 \\
 z - x - y &\leq 0 \\
 x &\geq 1 \\
 y &\geq 1 \\
 z &\geq x + 1.
 \end{aligned}$$

The optimum value is attained at $x = 2.5, y = 3.5, z = 5$. The arrival and required times for the stages are shown in Table 5.1. So only the regions in Line 1 - Stage 1 and Line 2 - Stage 2 need to be resynthesized. In addition the regions in Line 1 - Stage 2 and Line 2 - Stage 1 can be optimized for area since they can each be slowed by 0.5 units.

5.7.2 Experiments

The results of the resynthesis approach proposed in Section 5.6 are now described. Since there are no pipeline benchmark circuits we designed a set of circuits described in Section 5.7.2.

<i>stage #</i>	<i>Line</i>	<i>input arrival</i>	<i>output required</i>
1	1	0	$\min(2.5, 5 - 2) = 2.5$
	2	0	$\min(2.5, 5 - 3.5) = 1.5$
2	1	0	$\min(3.5, 5 - 2.5) = 2.5$
	2	0	$\min(3.5, 5 - 1) = 3.5$

• Table 5.1: Arrival and required times for resynthesis

The results of the experiments are summarized in Section 5.7.2

Benchmarks

All circuits use level-sensitive memory elements clocked using 2 phases. The outputs of successive stages are clocked on alternate phases. The five pipeline circuits are

1. *adder*: A 10 bit adder with 1 stage.
2. *mcnc*: This is a 2 stage cascade of 2 benchmarks from the MCNC suite of examples. The first stage is *alu2* and the second stage is *cm138*.
3. *parity*: Computes the parity of a 8 bit input, the parity of the even inputs and the parity of the odd inputs. It has 2 stages.
4. *addTi*: Described later in this section.
5. *population*: Counts the number of ones in a 32 bit input using 4 stages.

The table below summarizes some results of the algorithm to the pipeline circuits described above. The circuit is first decomposed into 2 input and/or gates. The unit delay fanout model is used. The optimal clock for these circuits (column 2 in Table 5.2) is computed using the algorithms described in Chapter 4.

Experiments

The implementation is studied from three differing viewpoints. The area of each circuit is measured in terms of the number of 2 input *and* and *or* gates and *inverters*.

1. In the first set of experiments, we seek to examine the effectiveness of the algorithm for area-clock period trade-off. In Table 5.2 we resynthesize each circuit for upto 4 target clock periods (in decreasing order), until the resynthesis algorithm is unable to produce a faster clock. Options to the combinational resynthesis algorithm are set to the default settings. Column 1 lists the examples. The optimum clock period to the initial circuit and the area of the initial circuit form the contents of column 2. Column 3 to 6 give the target clock period and area for the corresponding circuit. A "-" means that the combinational resynthesis algorithm failed to produce any improvement for lower target clock periods with the default settings. The table demonstrates that the area overhead can be controlled by specifying a target clock period. The only anomaly in the table is the circuit named *parity* (row 7), where setting a faster target clock resulted in an area saving. We attribute this to the fact that decreasing the target clock period increases the ϵ critical network (see [66] for details) which allows the optimizer to explore a large space for delay optimization.

<i>name</i>	<i>initial</i>	<i>target</i>			
	(clock/area)	(clock/area)			
adder	21.4/103	20/104	19/104	18/123	17/127
addT1	10.0/26	9/28	8/29	-	-
addT2	12.1/89	11/95	10/101	9/101	-
addT3	21.3/226	20/238	-	-	-
addT4	26.2/511	25/551	24/563	22/569	-
mcnc	54.6/384	52/391	50/413	48/412	-
parity	16.0/39	15/54	14/45	-	-
population	27.3/249	26/252	-	-	-

Table 5.2: Area-clock period trade-off

2. It is possible to minimize the clock period by repeatedly decreasing the clock period and using different options to the combinational resynthesis algorithm. These results are summarized in Table 5.3. The target clock period given in column 3 is the best result after a few iterations. Column 4 gives the results of using the optimal clock algorithm on the resynthesized pipeline. Columns 5 and 6 give the initial and final area. Column 7 gives the total time taken by the procedure to compute the best clock period. The last column gives the number of memory elements in the circuit. For the first 6 circuits the algorithm is able to take advantage of cycle stealing and a substantial reduction in the clock period is observed for a small area penalty.

name	clock			area		time (sec.)	latches
	initial	target	final	initial	final		
adder	21.4	15.3	15.2	103	135	6.0	45
addT1	10.0	7.2	7.2	26	29	1.1	10
addT2	12.1	8.2	8.0	89	106	20.5	25
addT3	21.3	19.2	19.2	226	240	10.9	56
addT4	26.2	22.0	22.0	511	535	43.0	119
mcnc	54.6	47.7	47.6	384	412	12.8	24
parity	16.0	13	12.2	39	54	6.7	11
population	27.3	25.4	25.4	249	320	44.1	59

Table 5.3: Pipeline resynthesis for minimum clock period using unit fanout model

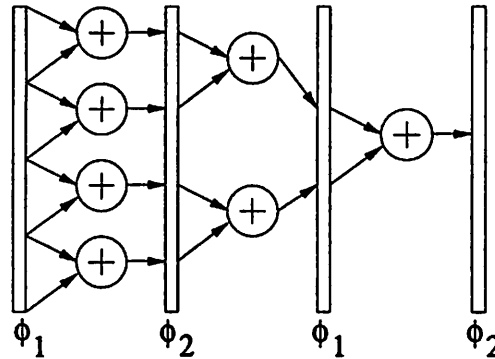


Figure 5.10: addT3- a 3 stage pipeline

- The complexity of the quadratic program grows as the number of stages. Recall that the number of variables and the number of constraints for a pipeline grow as the square of the number of stages. In order to study the efficiency of the quadratic program we constructed a pipeline (addT n) with n stages as follows: the pipeline computes the sum of 2^n inputs (each of width 3 bits). Stage S_i has 2^{n-i+1} inputs and 2^{n-i} outputs. Each stage has 2^{n-i} adders which compute the sum in parallel. A 3 stage pipeline is shown in Figure 5.10.

The time taken by the algorithm to analyze the circuit and come up with the stage delays (D'_{kk+1}) is summarized in the table below. The second and third columns give the initial and final clock periods. The next two columns give the time required to solve the quadratic program (QP) and the time taken for resynthesis (RSY). The size of the quadratic program, *i.e.* number of variables and number of constraints is shown alongside the entry in column

n	clock		time(sec.)	
	initial	target	QP	RSY
1	10.00	7.2	0.44 (3, 8)	0.6
2	12.10	8.72	1.15 (6, 20)	7.8
3	21.30	19.2	2.87 (10, 43)	8.0
4	26.20	22.0	11.44 (15, 76)	31

Table 5.4: Pipeline resynthesis for addTn

QP in brackets. The quadratic program seems to display a behavior which is nearly cubic in the number of stages.

5.8 Discussion

Phase separation and duty cycle constraints may be expressed in the form $x_i - x_j \leq \sigma_{ij} + \delta_{ij}c$, where the x 's may be the rise or fall of phases: σ_{ij} is a real number representing a fixed separation and δ_{ij} is a real $\in (-1, 1)$ representing the separation as a ratio of the target clock period. The addition of edges representing these constraints increases the number of constraints that force all cycles in \mathcal{G} to have non-negative weight.

When we consider cyclic circuits, we need to add the following condition for all cycles in the circuit. Let C be any simple cycle in the circuit. We need

$$\sum_{p \in C} D_p \leq \sum_{p \in C} K_p c, \quad (5.8)$$

where p is a combinational path from a latch to another, and D_p is the maximum delay along p . If we have an exponential set of cycles in the circuit, we will have an exponential number of constraints. If we are willing to restrict the regions to be resynthesized, the problem becomes more amenable and the size of the constraint matrix A can be pruned. However the result is a set of constraints on several interacting path lengths. The resynthesis algorithm will have to optimize these paths so that they simultaneously meet all the path constraints. The delay optimizer, in use currently, uses block oriented constraints on arrival and required times and is unable to handle path based constraints efficiently. Techniques that use path based delay optimization are still an issue of research.

The results obtained by Malik *et al.* [38] for pipeline circuits with FEDFFs, prompted us to investigate pipelines with AHLs and multi-phase clocking schemes. The goal was to

relate the clock period constraints for a target clock period to performance constraints on regions of combinational logic. The proposed approach gives a set of sufficient conditions for resynthesis, which if satisfied guarantee the target clock period to be achievable.

Chapter 6

Delay Insertion for Short Paths

Sequential synthesis of circuits requires that output signals arrive in a specified interval. During resynthesis of pipelines for a target clock period (see Chapter 5) it is required that every combinational path have a delay that lies between a maximum and a minimum value. Traditional delay optimization approaches consider only a part of the problem; namely to ensure that the delay of each path is less than the upper bound. This Chapter considers the problem of delaying outputs to meet the lower bound without violating upper bounds that have been satisfied.

6.1 Overview

For the rest of this Chapter, we shall restrict attention to combinational circuits. Providing a circuit that achieves the upper bound constraints at all outputs, given primary input arrival times, has long been a focus of research. Singh *et al.* [66] present an algorithm that uses the movement of critical signals closer to the output of a cone of logic to speed up circuits. Fishburn presents [16] an approach to decrease the depth of a circuit and in [17] presents an iterative algorithm that combines several known methods in a heuristic manner. We are concerned with the problem of satisfying lower bounds on delays of paths. De Micheli *et al.* [81] in an effort concerning wave pipelining of circuits, solve a problem which is related to the problem we describe in this Chapter.

The definitions of terms used in this Chapter are described in Section 6.2. Section 6.3 introduces the “padding” problem and presents associated theoretical results (including a naive approach to solve the problem). A linear programming technique for an optimum solution to the problem is given in Section 6.4. Section 6.5 describes two extensions to the formulation. Section 6.6 deals with an application to wave pipelining of circuits. Experimental results form the contents of

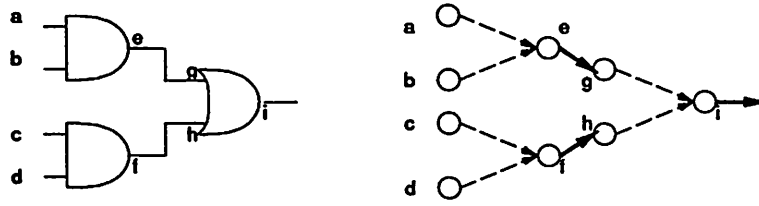


Figure 6.1: Graph for a simple circuit

Section 6.7.

6.2 Definitions

A combinational circuit \mathcal{C} is represented as a directed acyclic graph. The terms circuit and graph are used interchangeably. A circuit is an interconnection of gates. Each gate has one output pin and one or more input pins. For every pin in the circuit there is a vertex in \mathcal{C} . The set of vertices is denoted by V . There are two kinds of edges, internal and external. The edge sets are denoted by E^I and E^E respectively. An internal edge is directed from an input pin of a gate to its output pin. The external edges represent interconnections or wiring in the circuit. Every connection from an output of a gate to an input of another gate is represented by an external edge. Figure 6.1 shows a simple and-or circuit with external edges shown in bold and internal edges shown in dashes. The circuit reads data through leads termed as primary inputs and provides data on primary outputs. The set of primary inputs/outputs is denoted by PI/PO . There is a vertex for every primary input and every primary output.

An edge e_{ij} from i to j is denoted by $i \rightarrow j$; i is called a fanin of j and j is called a fanout of i . We use the notation $FI(i)$ to denote the set of fanins of i and $FO(i)$ to denote the set of fanouts of i . A vertex representing a primary input has no fanins and a vertex representing a primary output has no fanouts. A weight w_{ij} is associated with each edge e_{ij} —

1. if $e_{ij} \in E^I$, then the weight is the delay incurred by a signal propagating from the input represented by i to the output of the gate, or
2. if $e_{ij} \in E^E$, then the weight is a variable whose value is to be determined.

Delays can be inserted only on external edges.

There are two arrival times associated with each vertex i —

1. an early arrival time a_i , and
2. a late arrival time A_i .

The arrival times at vertex i are computed as follows —

1. if i is a primary input, then a_i and A_i are specified by the user or an algorithm at the higher level (often $a_i = A_i = 0$), or
2. if i is an input/output pin of a gate,

$$a_i = \min_{j \in FI(i)} (a_j + w_{ji}), \quad (6.1)$$

$$A_i = \max_{j \in FI(i)} (A_j + w_{ji}). \quad (6.2)$$

A path p in the graph is a sequence of vertices i_1, i_2, \dots, i_n such that each vertex is a fanin to the next vertex in the sequence. It is denoted by $p : i_1 \rightsquigarrow i_n$. The delay of a path $p : i_1 \rightsquigarrow i_n$ is $\sum_{j=1}^{n-1} w_{i_j i_{j+1}}$ and is denoted by $d(p)$.

At every primary output i , data is required to be available no earlier than r_i and no later than R_i , namely $r_i \leq a_i \leq A_i \leq R_i$. All previous delay optimization techniques try to guarantee $A_i \leq R_i$ on termination of the algorithm. However no attempt is made to ensure $a_i \geq r_i$ for each primary output i . It is unclear whether a naive insertion of delays will suffice to meet this constraint, because paths can interact one another. Consequently, inserting delay on a sub-path to slow an early arriving signal can also slow some late arriving signals, offsetting any gain made by the delay optimizer.

We say that a path $p : i_1 \rightsquigarrow i_n$, $i_1 \in PI, i_n \in PO$ is a **critical long path** if \forall edges $e_{i_k i_{k+1}} \in p$, we have $A_{i_{k+1}} = A_{i_k} + w_{i_k i_{k+1}}$, where $k = 1, \dots, n - 1$. The set of long paths P is the set of all critical long paths which have $A_{i_n} = R_{i_n}$. We say that p is a **critical short path** if $a_{i_1} + d(p) < r_{i_n}$. The short path set S is the set of all critical short paths.

We make the following assumptions —

1. Gates have a unique direction of signal propagation, namely from inputs to outputs.
2. The amount of delay that can be padded is continuous. In reality, only discrete amounts of delay can be padded. We use the continuous padding problem as an approximation to the discrete padding problem. We find an optimum solution to the relaxed problem, and derive a discrete solution from the continuous solution.
3. We also assume that all gates provide the same input load.

6.3 Is padding always possible?

The problem definition for the delay insertion problem is as follows:

Given a circuit $C(V, E^E \cup E^I)$, edge-weights on the internal edges, arrival times (early and late) at all primary inputs and the required bounds (early and late) at all primary outputs (R_i and r_i at primary output i) assign a set of delays (real numbers) to the external edges so that all paths meet the required upper and lower bounds.

We shall call this the padding problem. We assume that all paths from a primary input to a primary output respect (are \leq) the upper bound at the output. We shall insert delay on the external edges to satisfy the lower bounds. The first issue we need to examine is whether it is possible (if at all) to always insert delays to meet lower bounds, without affecting upper bounds of paths. The goal of this Section is to demonstrate that it is always possible to do so under mild conditions on arrival and required times at the inputs and outputs.

The following Lemma gives simple bounds on early and late arrival times at a vertex.

Lemma 6.3.1 *Let $p : i_1 \rightsquigarrow i_n$ be a path in the graph. Then*

- $A_{i_n} \geq A_{i_1} + d(p)$, and
- $a_{i_n} \leq a_{i_1} + d(p)$.

Proof By induction on the path.

- Base case: The path has one edge $p : i_1 \rightsquigarrow i_2$. From 6.1 we get

$$\begin{aligned} A_{i_2} &\geq A_{i_1} + w_{i_1 i_2} \\ \Rightarrow A_{i_2} &\geq A_{i_1} + d(p). \end{aligned}$$

- Induction case: Let $p : i_1 \rightsquigarrow i_{k+1}$ and assume that $A_{i_k} \geq A_{i_1} + d(i_1 \rightsquigarrow i_k)$. We need to show that $A_{i_{k+1}} \geq A_{i_1} + d(i_1 \rightsquigarrow i_{k+1})$. We have from Equation 6.2

$$\begin{aligned} A_{i_{k+1}} &\geq A_{i_k} + w_{i_k i_{k+1}} \\ \Rightarrow A_{i_{k+1}} &\geq A_{i_1} + d(i_1 \rightsquigarrow i_k) + w_{i_k i_{k+1}} \\ \Rightarrow A_{i_{k+1}} &\geq A_{i_1} + d(i_1 \rightsquigarrow i_{k+1}) \\ \Rightarrow A_{i_{k+1}} &\geq A_{i_1} + d(p). \end{aligned}$$

The proof for $a_{i_n} \leq a_{i_1} + d(p)$ proceeds along similar lines. ■

The ensuing theorem gives necessary and sufficient conditions for the padding problem to have a solution.

Theorem 6.3.2 *In any circuit, let $i_1 \in PI$, $i_n \in PO$, such that \exists a path $p : i_1 \rightsquigarrow i_n$. The padding problem has a feasible solution if and only if $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}, \forall$ such p .*

Proof The proof is divided into two parts.

- \Rightarrow) We will first show the necessity of the condition. Let p be any path from $i_1 \in PI$ to $i_n \in PO$. If the padding problem has a solution then we know that,
 - the upper bound on path p is met, hence from Lemma 6.3.1

$$A_{i_1} + d(p) \leq R_{i_n}, \quad (6.3)$$

- the lower bound on path p is met, hence from Lemma 6.3.1

$$a_{i_1} + d(p) \geq r_{i_n}. \quad (6.4)$$

Subtracting Equation 6.4 from Equation 6.3, we get $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}$.

- \Leftarrow) The sufficiency of the condition will be proved by a constructive argument. This will in turn provide insight to a naive algorithm for padding delays. Let s be a critical short path, *i.e.* $s \in S$ and s violates the lower bound on the path delay. Recall that P is the set of critical long paths. The following Lemma provides a property of a critical short path that is crucial for the proof.

Lemma 6.3.3 *Let $s : i_1 \rightsquigarrow i_n$ be a critical short path, namely $s \in S$, and let $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}$ then \exists an edge e on s , such that e does not lie on any critical long path ($e \notin q, \forall q \in P$).*

Proof For sake of contradiction assume that each edge $e_{i_k i_{k+1}}$ (denoted by e_k for brevity¹) of s is a part of some long path in P . Consider the construction described below.

¹ $k = 1, \dots, n - 1$

- **Construction C:** Restrict attention to only those paths in P that contain an edge of s . Without loss of generality replace P by this subset. Several edges of s (not necessarily consecutive) may lie on the same $q \in P$. In addition each edge e_k of s may lie on several long paths in P — denote this set by Q_k . Each edge e_k is given a label l_k , and every such label is assigned a path from P (denoted by Q_{l_k}) as follows:

$$l_k = \begin{cases} 1, & Q_1 = q, q \in Q_1 & \text{if } k = 1 \text{ (initialization)} \\ l_{k-1}, & & \text{if } e_k \in Q_{l_{k-1}} \\ l_{k-1} + 1, & Q_{l_k} = q, q \in Q_k & \text{otherwise.} \end{cases} \quad (6.5)$$

Let L be the largest label so assigned. We now break s into a set of L disjoint sub-paths. Each sub-path, denoted by $s_k, k = 1, \dots, L$, is a set of successive edges with the same label. Let $h(s_k)$ be the vertex where s_k starts and $t(s_k)$ be the vertex where s_k ends. Note that $h(s_{k+1}) = t(s_k)$. Note that all vertices representing input pins of gates have a single fanin and a single fanout. Consequently an internal edge (a fanout edge from an input pin) must have the same label as the fanin edge to the input pin. A change of labels on s can take place only on an external edge. Each $h(s_k)$ (and $t(s_k)$) can only be an output pin of a gate. Let us divide each Q_k into 3 parts

1. Q_k^1 - from a primary input to $h(s_k)$ (inclusive)
2. Q_k^2 - s_k not including $h(s_k)$ but including $t(s_k)$ and
3. Q_k^3 - rest of Q_k .

The construction process yields a covering of the edges of the short path s with at most L long paths from P . Let $A(Q_k)$ denote the late arrival at the primary input on the path Q_k . Let $R(Q_k)$ denote the upper bound on the delay to the primary output on path Q_k .

Now consider the following paths in the graph

- Q_k and
- the path obtained by the concatenation of Q_k^1 and Q_{k-1}^3

for $k = 2, \dots, L - 1$. Since $Q_k \in P$ (the set of long paths) we know (see Figure 6.2)

$$A(Q_k) + d(Q_k) = R(Q_k),$$

implying

$$A(Q_k) + d(Q_k^1) + d(Q_k^2) + d(Q_k^3) = R(Q_k). \quad (6.6)$$

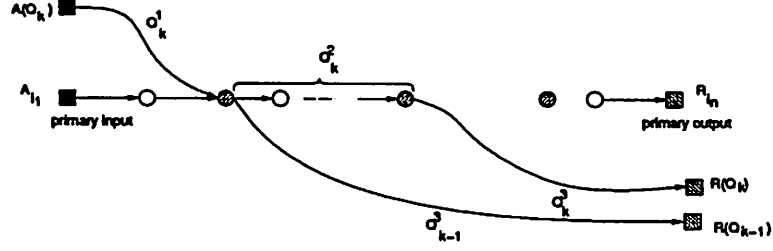


Figure 6.2: Short and long path interactions

Consider the second path ($Q_k^1 Q_{k-1}^3$):

$$A(Q_k) + d(Q_k^1) + d(Q_{k-1}^3) \leq R(Q_{k-1}). \quad (6.7)$$

Subtracting Equation 6.7 from Equation 6.6 we get

$$d(Q_k^2) + d(Q_k^3) - d(Q_{k-1}^3) \geq R(Q_k) - R(Q_{k-1}). \quad (6.8)$$

The terminal cases are now described.

– $k = 1$: Since Q_1^1 is empty and $A(Q_1) = A_{i_1}$

$$A_{i_1} + d(Q_1) = R(Q_1),$$

implying

$$A_{i_1} + d(Q_1^2) + d(Q_1^3) = R(Q_1). \quad (6.9)$$

– $k = L$: In this case Q_L^3 is empty and $R(Q_L) = R_{i_n}$. Consider the path Q_L and the path obtained by concatenating Q_L^1 and Q_{L-1}^3 . For the first path we obtain

$$A(Q_L) + d(Q_L^1) + d(Q_L^2) = R_{i_n}, \quad (6.10)$$

and for the second path we obtain

$$A(Q_L) + d(Q_L^1) + d(Q_{L-1}^3) \leq R(Q_{L-1}). \quad (6.11)$$

Subtracting Equation 6.11 from Equation 6.10 yields

$$d(Q_L^2) - d(Q_{L-1}^3) \geq R_{i_n} - R_{Q_{L-1}}. \quad (6.12)$$

Summing Equation 6.8 for $k = 2, \dots, L - 1$ and Equations 6.9 and 6.12 we obtain

$$A_{i_1} + \sum_{k=1}^L d(Q_k^2) \geq R_{i_n}. \quad (6.13)$$

Note that the second term on the left hand side is nothing but $d(s)$. But we know that $s \in S$ implying

$$a_{i_1} + d(s) < r_{i_n}. \quad (6.14)$$

Subtracting 6.14 from 6.13 yields

$$A_{i_1} - a_{i_1} > R_{i_n} - r_{i_n}. \quad (6.15)$$

Contradiction!!

It is easy to obtain an external edge that satisfies the criterion once such an e is found. If e is an external edge then we are done. Suppose not, then e is an internal edge and the source of e is an input pin of a gate. Consequently the source of e has only one fanin and any path containing the fanin edge must also contain e and vice versa. Hence the fanin edge does not belong to any $q \in P$. An edge e that satisfies the criterion in Lemma 6.3.3 is called a candidate edge. ■

Corollary 6.3.4 *In any circuit, if for every pair $i_1 \in PI$ and $i_n \in PO$, such that \exists a path $p : i_1 \rightsquigarrow i_n$ it is known that $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}$, then the padding problem can be solved.*

Proof The proof is constructive and is the last piece to the sufficiency condition of the Theorem. Start with any $s \in S$. By lemma 6.3.3 $\exists e \in s$ such that $e \notin q, \forall q \in P$. Consequently adding some delay on e does not affect the paths in P . However this might lead to some paths containing e becoming critical long paths. As soon as this happens, we add these new critical long paths to the set P and repeat the procedure to find a new candidate edge. This is done until s meets the lower bound constraint. The procedure is repeated for each path in S after updating the early arrival times. ■

Let us briefly summarize the proof for sufficiency. We first show that for any short path $s \in S$, if the sufficiency condition is met, then \exists an edge which can be padded with some delay. Note that the sufficiency condition is independent of the delays of the gates and the padding. Corollary 6.3.4 gives a procedure for delay insertion, using Lemma 6.3.3 repeatedly and updating the set P . ■

An intuitive explanation for the condition $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}$ is as follows. Interpret the term $A_{i_1} - a_{i_1}$ as the uncertainty interval of arrival (of the signal) at primary input i_1 . $R_{i_n} - r_{i_n}$ is the required uncertainty at output i_n . Since the circuit is causal, it cannot make the uncertainty interval at the output any narrower than the uncertainty interval at its input².

Corollary 6.3.5 *If $A_{i_1} = a_{i_1}$, then the padding problem always has a feasible solution.*

Proof By observing the fact that $r_{i_n} \leq R_{i_n}$ for paths $p : i_1 \rightsquigarrow i_n$. Thus meeting the sufficiency condition of Theorem 6.3.2. ■

6.3.1 A naive algorithm

Let us examine the feasibility problem. Since we need to check $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}$, $\forall p : i_1 \rightsquigarrow i_n$, such that $i_1 \in PI$, $i_n \in PO$, it is equivalent to a reachability analysis of the graph. The complexity of this is $O(|E^E \cup E^I|)$. The complexity of computing the arrival times is $O(|E^E \cup E^I| \log |V|)$.

Given below is the pseudo-code for an algorithm to pad delays, described in Corollary 6.3.4.

Procedure 6.3.1

Let

$C =$ combinational circuit

$P =$ set of long paths in C

$S =$ set of short paths in C

while $S \neq \emptyset$ {

- Pick $s \in S$

– while (s does not meet lower bound) {

- * Find a candidate edge $e \in s$ | e satisfies Lemma 6.3.3

Insert delay on e until one of the following is true

1. some path p containing e becomes a critical long path

$$P = P \cup p$$

continue

²This is akin to the fact that a passive network cannot produce an output voltage that is larger than the source that drives it


```

    2. s meets lower bound
       break (success in padding)
  }
  Delete s from S
}

```

There are several issues that need to be addressed.

- Enumeration of paths must be avoided since it can be exponential. In particular we need to implicitly maintain P and S .
- Finding an $e \in s$ that meets the conditions of Lemma 6.3.3 has also to be done implicitly.
- Detecting the amount of delay that can be inserted on a candidate edge before some other path containing it becomes a critical long path is another issue.

We resort to the notion of slack. For each vertex i , define the required times as follows —

1. if i is a primary output then r_i and R_i are specified by the user or an algorithm at the higher level, or
2. if i is an input/output pin of a gate

$$r_i = \max_{j \in FO(i)} (r_j - w_{ij}), \quad (6.16)$$

$$R_i = \min_{j \in FO(i)} (R_j - w_{ij}). \quad (6.17)$$

The slacks at each vertex i are defined as

$$\delta_i = r_i - a_i, \quad (6.18)$$

$$\Delta_i = R_i - A_i. \quad (6.19)$$

A primary output i has a short path containing it, if $\delta_i > 0$. This solves the problem of maintaining a set of short paths. We simply keep track of the primary outputs which have strictly positive δ 's at any instant of the algorithm. Note that for all primary outputs $\Delta_i \geq 0$.

Lemma 6.3.6 *Let i be a vertex such that $\delta_i > 0$, then $\exists j \in FO(i)$, such that $\delta_j \geq \delta_i > 0$*

Proof Since $\delta_i > 0$, $\Rightarrow r_i > a_i$. From the definition of the required time at i in Equation 6.16, we know $\exists j \in FO(i)$, such that $r_j - w_{ij} = r_i$. Now $a_j \leq a_i + w_{ij}$. Subtracting this from the previous equality we get $r_j - a_j \geq r_i - a_i$. ■

Corollary 6.3.7 *Let i be a vertex such that $\delta_i > 0$, then i belongs to some short path in S .*

Proof By induction using Lemma 6.3.6 repeatedly. We know, $\exists j \in FO(i)$, such that $\delta_j \geq \delta_i > 0$. Continue this process until we hit a primary output l with $\delta_l > 0$. This gives a path, say p^1 . Next find a path $p^2 : i_1 \rightsquigarrow i$, $i_1 \in PI$ such that $a_i = a_{i_1} + d(p^2)$. The path $p^2 p^1$ is a short path. ■

Lemma 6.3.8 *For all vertices i , $R_i \geq A_i$ ($\Delta_i \geq 0$).*

Proof For sake of contradiction assume not, so $\exists i$, such that $R_i < A_i$. From Equation 6.17, $\exists j \in FO(i)$, such that $R_j - w_{ij} = R_i$. Also $A_j \geq A_i + w_{ij}$. Consequently $R_j - A_j \leq R_i - A_i$. Hence $\Delta_j \leq \Delta_i < 0$. Continue this process until we hit a primary output l with $\Delta_l < 0$. Contradicts the fact that all paths satisfy the upper bound. ■

Corollary 6.3.9 *Let i be a vertex such that $\Delta_i = 0$, then $\exists j \in FO(i)$, such that $\Delta_j = 0$.*

Proof As a part of the proof to Lemma 6.3.8 we showed that $\exists j \in FO(i)$, such that $\Delta_j \leq \Delta_i = 0$. Also since $\Delta_j \geq 0$ for all j , we conclude that $\Delta_j = 0$. ■

Corollary 6.3.10 *Let i be a vertex such that $\Delta_i = 0$, then i belongs to some long path in P .*

Proof By induction using Corollary 6.3.9 repeatedly. We know, $\exists j \in FO(i)$, such that $\Delta_j = \Delta_i = 0$. Continue this process until we hit a primary output l with $\Delta_l = 0$. This gives a path, say p^1 . Next find a path $p^2 : i_1 \rightsquigarrow i$, $i_1 \in PI$ such that $A_i = A_{i_1} + d(p^2)$. The path $p^2 p^1$ is a long path. ■

To maintain the set of long paths we keep track of all primary outputs i such that $\Delta_i = 0$. To find a candidate edge that meets the conditions of Lemma 6.3.3, we maintain a list of external edges e_{ij} satisfying the following conditions:

1. $a_j = a_i + w_{ij}$, $r_j > a_j$ and
2. $A_j > A_i + w_{ij}$.

The maximum amount of delay that can be inserted on such an edge is $R_j - A_i - w_{ij}$. Any more, will result in some path p containing e_{ij} becoming a critical long path. If $r_j - a_j < R_j - A_i - w_{ij}$

then padding more than $\tau_j - a_j$ does not aid the short path. So we insert an amount of delay d_{ij} equal to

$$d_{ij} = \min(R_j - A_i - w_{ij}, \tau_j - a_j) \quad (6.20)$$

The implicit procedure may be described as follows:

Procedure 6.3.2

Let

$C = \text{combinational circuit}$

$\mathcal{L} = \text{list of possible candidate edges for inserting delay}$

$\pi_j = \text{list of primary outputs } k \text{ reachable from vertex } j \text{ such that } \delta_k > 0$

while $(\exists k \in PO \text{ such that } \delta_k > 0)$ {

- Find e_{ij} , $e_{ij} \in \mathcal{L}$ with $k \in \pi_j$

Insert delay = d_{ij}

Update a , A , τ , R , δ , Δ and lists π in the graph

Update \mathcal{L}

}

The choice of e is done in a heuristic manner, so as to minimize the amount of delay that needs to be inserted. We assume that the area penalty is directly proportional to the delay inserted. Hence we are seeking an area optimal solution. As a heuristic, with each vertex j we maintain a list π_j of primary outputs that are reachable from j and violate the short path bounds. We choose an edge e_{ij} with the least value of $\frac{d_{ij}}{|\pi_j|}$; *i.e.* we choose the edge with the best delay gain per critical output. A refinement to the above procedure is to find a cutset of external edges so that delays can be inserted independently on each edge. A min-cut found using a flow algorithm as described in [66] can be used to insert delays simultaneously on several edges without updating the values for the slacks.

6.4 A linear programming approach

In this section we show that the minimum padding problem is equivalent to a linear program obtained by relaxing Equations 6.1 and 6.2. Let the early (late) arrivals at a primary input i be denoted by λ_i (Λ_i). Consider the following optimization problem (*OPT1*):

$$OPT1 : \quad \min(\sum_{e_{ij} \in E^E} w_{ij})$$

$$\begin{aligned}
A_j &= \max_{i \in FI(j)} (A_i + w_{ij}) \\
a_j &= \min_{i \in FI(j)} (a_i + w_{ij}) \\
A_i &\leq R_i && \forall i \in PO \\
a_i &\geq r_i && \forall i \in PO \\
A_i &= \Lambda_i && \forall i \in PI \\
a_i &= \lambda_i && \forall i \in PI.
\end{aligned}$$

Consider the relaxed linear program ($P1$) obtained by replacing the min and max operators by appropriate inequalities:

$$\begin{aligned}
P1 : \quad & \min(\sum_{e_{ij} \in E^E} w_{ij}) \\
& A_j \geq A_i + w_{ij} \quad \forall i \in FI(j) \\
& a_j \leq a_i + w_{ij} \quad \forall i \in FI(j) \\
& A_i \leq R_i \quad \forall i \in PO \\
& a_i \geq r_i \quad \forall i \in PO \\
& A_i = \Lambda_i \quad \forall i \in PI \\
& a_i = \lambda_i \quad \forall i \in PI.
\end{aligned}$$

Theorem 6.4.1 *Let (w^*, A^*, a^*) be an optimum solution to $P1$, then an optimum solution to $OPT1$ can be constructed with the edge weights w^* .*

Proof The objective functions in the two optimization problems are the same. Since the feasible region of $P1$ contains the feasible region of $OPT1$ the following statements are true —

- if $P1$ is infeasible then so is $OPT1$, and
- the value of the objective function in $P1$ at optimality is a lower bound on the value of the objective function in $OPT1$.

Given (w^*, A^*, a^*) , an optimum solution to $P1$, it is possible to construct a feasible solution to $OPT1$ with the same cost as the optimum solution to $P1$. This implies that the values of w^* are also optimum to $OPT1$. Recall that the values of A_i and a_i for all $i \in PI$, are fixed in both problems to the same values. Let us construct a solution \hat{A}, \hat{a} to $OPT1$ recursively as follows:

$$\begin{aligned}
\hat{A}_i &= \Lambda_i \quad \forall i \in PI \\
\hat{a}_i &= \lambda_i \quad \forall i \in PI \\
\hat{A}_j &= \max_{i \in FI(j)} (\hat{A}_i + w_{ij}^*) \quad \text{otherwise} \\
\hat{a}_j &= \min_{i \in FI(j)} (\hat{a}_i + w_{ij}^*) \quad \text{otherwise.}
\end{aligned}$$

These equations need to be evaluated in topological order from the inputs. Since the circuit is acyclic, this ensures that \hat{A}_j (\hat{a}_j) for each j is computed before being computing the late (early) arrival at its fanouts. Note that for all i

1. $\hat{A}_i \leq A_i^*$, and
2. $\hat{a}_i \geq a_i^*$.

Since for $i \in PO$, we have

1. $A_i^* \leq R_i$ and
2. $a_i^* \geq r_i$,

we conclude $\hat{A}_i \leq R_i$ and $\hat{a}_i \geq r_i$; implying that (w^*, \hat{A}, \hat{a}) is a feasible solution to $OPT1$. Since the value of the objective function of $OPT1$ at (w^*, \hat{A}, \hat{a}) is equal to the lower bound given by the minimum value of the objective function of $P1$, (w^*, \hat{A}, \hat{a}) is a point of optimality for $OPT1$. ■

6.5 Refinements

6.5.1 Delay model

A main drawback of the delay model used so far is that the delay from an input pin to an output pin of a gate is assigned a single value. However the delay may differ for an output rise and for an output fall. Consequently each internal edge must be assigned a minimum and a maximum delay (denoted by w_{ij}^{max} and w_{ij}^{min}). For a path $p : i_1 \rightsquigarrow i_n$, let us define

$$D(p) = \sum_{j=1}^{n-1} w_{i_j i_{j+1}}^{max},$$

$$d(p) = \sum_{j=1}^{n-1} w_{i_j i_{j+1}}^{min}.$$

If we assume that the buffers are specially designed to have equal rise and fall times, Theorem 6.3.2 can be modified to read

Theorem 6.5.1 *In any circuit, let $i_1 \in PI$, $i_n \in PO$, such that \exists a path $p : i_1 \rightsquigarrow i_n$. The padding problem has a feasible solution if and only if $A_{i_1} - a_{i_1} + D(p) - d(p) \leq R_{i_n} - r_{i_n}$, \forall such p .*

Note that the structure of the path is now relevant because of the term $D(p) - d(p)$.

6.5.2 Discrete delay insertion

An issue of concern is that delay insertion in reality can only be done in discrete steps. So far, we have assumed that an arbitrary amount of delay can be inserted on each edge. Instead let B be the minimum delay that can be inserted. A problem arises if, for a short path $s \in S$, and for every edge $e_{ij} \in s$ satisfying the requirements of Lemma 6.3.3, we have

$$R_j - A_i - w_{ij} < B. \quad (6.21)$$

We say an output has a **short path violation** if due to the discrete nature of gate delays, we are unable to meet the lower bound on the output. We will now determine a bound on the short path violation as a function of B . Once again, we will not tolerate any change in critical long paths.

Theorem 6.5.2 *Let $s \in S$, $s : i_1 \rightsquigarrow i_n$ and the following statements are true;*

1. $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}$ and
2. $\forall e \in s, e_{i_k i_{k+1}} : i_k \rightarrow i_{k+1}, k = 1, \dots, n-1$, such that $e_{i_k i_{k+1}} \notin q, \forall q \in P$, we have $R_{i_{k+1}} - A_{i_k} - w_{i_k i_{k+1}} < B$,

then $a_{i_1} + d(p) \geq r_{i_n} - mB$, where $0 < m \leq |V| - 1$.

Proof For every edge $e_{i_k i_{k+1}} : i_k \rightarrow i_{k+1}$ (henceforth denoted by e_k for brevity), $k = 1, \dots, n-1$ one of the following is true -

1. e_k belongs to one or more long paths (denoted by $\{q_k\}$) or
2. e_k is a candidate edge and $R_{i_{k+1}} - A_{i_k} - w_{i_k i_{k+1}} < B$. In this case, there must be one or more paths ($\{q_k\}$) containing e_k such that $A(q_k) + d(q_k) > R(q_k) - B$, i.e. the addition of B units of delay on e_k forces a violation of the upper bound requirement on q_k .

Use Construction C (as described in proof to Lemma 6.3.3) with the set $P^1 = \bigcup_{k=1}^{n-1} \{q_k\}$ instead of the set P . Let Q_1, Q_2, \dots, Q_L be the covering of s found by the construction process. We can see that for $k = 2, \dots, L-1$,

$$d(Q_k^2) + d(Q_k^3) - d(Q_{k-1}^3) \begin{cases} \geq R(Q_k) - R(Q_{k-1}) & \text{if } Q_k \text{ belongs to a long path} \\ > R(Q_k) - R(Q_{k-1}) - B & \text{if } Q_k \text{ contains candidate edges.} \end{cases} \quad (6.22)$$

The terminal cases are

- $k = 1$:

$$A_{i_1} + d(Q_1^2) + d(Q_1^3) \begin{cases} \geq R(Q_1) & \text{if } Q_1 \text{ belongs to a long path} \\ > R(Q_1) - B & \text{if } Q_1 \text{ contains candidate edges.} \end{cases} \quad (6.23)$$

- $k = L$:

$$d(Q_L^2) - d(Q_{L-1}^3) \begin{cases} \geq R_{i_n} & \text{if } Q_L \text{ belongs to a long path} \\ > R_{i_n} - B & \text{if } Q_L \text{ contains candidate edges.} \end{cases} \quad (6.24)$$

Summing up, we get $A_{i_1} + d(s) > R_{i_n} - mB$, where m is the number of paths covering s that have candidate edges. But $A_{i_1} - a_{i_1} \leq R_{i_n} - r_{i_n}$, namely $R_{i_n} - A_{i_1} \geq r_{i_n} - a_{i_1}$. Since $d(s) + mB > R_{i_n} - A_{i_1}$, we conclude $d(s) + mB > r_{i_n} - a_{i_1}$, implying $a_{i_1} + d(s) > r_{i_n} - mB$. So as B is decreased, we will be able to control the amount of short path violation. A naive upper bound on m is $|V| - 1$. ■

To solve the padding problem with discrete delay, we first solve the problem with the continuous delay relaxation. We use the optimum solution to the continuous problem as a *heuristic* to solving the discrete padding problem. Let $B = \{b_1, \dots, b_k\}$ be a set of buffers available in the library. Let $d(b_i)$ be the delay and $a(b_i)$ be the area of buffer b_i . Two problems arise in discrete padding.

- Each edge e requires a delay of W_e to be inserted on it. We need to find a set of buffers from B that best approximates W_e . We want the delay inserted to be as close as possible to W_e without exceeding it. Mathematically, we find a set B_e consisting of elements from B , such that $(W_e - \sum_{i \in B_e} d(b_i))$ is minimized, subject to $\sum_{i \in B_e} d(b_i) \leq W_e$. Any ties are broken by comparing the respective area penalties $(\sum_{i \in B_e} a(b_i))$. With $d(b_i) \in \mathbb{Z}^+$ for all i , and W_e a positive number, the problem is equivalent to the subset-sum problem [19], which is NP-complete.
- The problem above concentrates only on the best solution for an edge. A difficulty arises when (see Figure 6.3) multiple fanouts from a gate require delay insertions. Dramatic area gains are to be made if instead of buffering each fanout separately, a buffer tree is built. In Figure 6.3, quite clearly the second buffer tree is superior in terms of area to the first buffer tree. If we assume that elements from B can be always combined to give a delay of W_e exactly, the buffer tree construction is needed for area recovery. Let s be a gate with fanouts $\{f_1, f_2, \dots, f_n\}$. Let the i^{th} fanout edge from s to f_i , require a delay of w_i to be inserted.

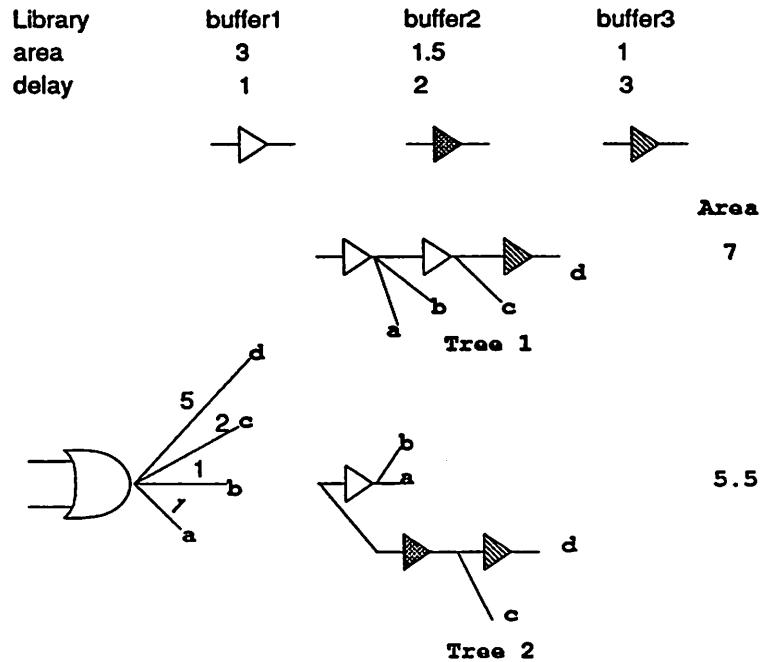


Figure 6.3: Area optimization during delay insertions

The problem is to construct a tree rooted at the output of s , whose vertices are elements of \mathcal{B} . The leaves of the tree are inputs of the fanout gates of s . The constraint is that for each root-to- j^{th} leaf path, say $p_j : s \rightsquigarrow f_j$, $\sum_{b_i \in p_j} d(b_i) = w_j$ and the objective is to minimize $\sum a(b_i)$. Quite clearly, if we can solve this problem, we can also solve the previous problem by setting $n = 1$.

Thus finding the best approximation for delay insertion for a single fanout and the buffer tree construction are difficult problems. It is known that the problem of inserting buffers to meet upper bounds on arrival times with an area constraint and taking fanout loads into account, is NP-complete [5, 73]. To solve the discrete delay problem, we first find the best approximation for each fanout edge independently by a branch and bound procedure that enumerates the different sets B_e . Area recovery is then done by extracting buffers common to fanouts, in a greedy manner. Note that buffer tree construction is further complicated by the fact that the delays of buffers change as the fanouts change. During the greedy tree construction, attempts are made to take the load into account. We recognize that this approach can be far from optimal, but as a first cut the heuristic works fine. A reason for permitting a rough approximation at this stage of the design is that fine delay tuning can be often achieved successfully by transistor sizing in MOS (resistor sizing in ECL)

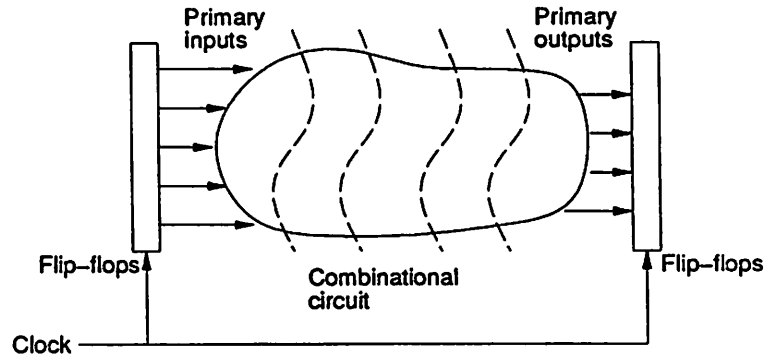


Figure 6.4: Wave pipelining

circuits.

6.6 Relation to wave pipelining

Wave pipelining is a design technique that allows multiple streams of data to flow in a combinational region at a given instant of time. The clock period in a circuit (using flip-flops) is determined by the longest combinational path in the circuit. In the case of wave pipelining, the system is clocked at a faster rate. Consider the structure used for wave-pipelining as shown in Figure 6.4. The flip-flops at the inputs and outputs are clocked at a rate faster than the standard rate determined by the longest path. The primary inputs have equal early and late arrival times. Let T denote the clock period. The clock period depends[81] on —

- the maximum difference between the longest and shortest delay (to any gate) from the inputs (T_{ls}),
- the maximum clock skew (T_{skew}), and
- set-up and hold times for the memory elements (T_{sh}).

A first order approximation for T is $T_{ls} + 2 * T_{skew} + T_{sh}$. Wong *et al.* [81] focus on minimizing the first term. We shall focus on inserting minimum delays to meet a target T_{ls} . In the terminology of Wong *et al.* [81] this is called rough tuning. In order to prevent data from one wave colliding with (corrupting) data in the previous wave we require

$$a_i + T_{ls} \geq A_i \quad (6.25)$$

at every vertex i . So given T_{ls} , the minimum padding problem may be phrased as:

$$\begin{aligned}
 OPT2 : \quad & \min(\sum_{e_{ij} \in E^E} w_{ij}) \\
 A_j &= \max_{i \in FI(j)} (A_i + w_{ij}) \\
 a_j &= \min_{i \in FI(j)} (a_i + w_{ij}) \\
 A_i &\leq R_i && \forall i \in PO \\
 a_i &\geq A_i - T_{ls} && \forall i \in V \\
 A_i &= 0 && \forall i \in PI \\
 a_i &= 0 && \forall i \in PI.
 \end{aligned}$$

We still enforce $A_i \leq R_i$ at each primary input. This is necessary because the number of “waves” that simultaneously exist in the combinational circuit depend on T_{ls} and the length of the longest path. Removing this constraint helps the optimization problem but can result in a large number of data “waves” in the circuit. Consider the relaxed LP:

$$\begin{aligned}
 P2 : \quad & \min(\sum_{e_{ij} \in E^E} w_{ij}) \\
 A_j &\geq A_i + w_{ij} && \forall i \in FI(j) \\
 a_j &\leq a_i + w_{ij} && \forall i \in FI(j) \\
 A_i &\leq R_i && \forall i \in PO \\
 a_i &\geq A_i - T_{ls} && \forall i \in V \\
 A_i &= 0 && \forall i \in PI \\
 a_i &= 0 && \forall i \in PI.
 \end{aligned}$$

It is easy to show (along similar lines to the proof of Theorem 6.4.1) that if (w^*, A^*, a^*) is an optimum solution to $P2$, then we can construct an optimum solution to satisfy $OPT2$. Since the feasible region of $P2$ contains the feasible region of $OPT2$, if $P2$ is infeasible then $OPT2$ is also infeasible. However, we are no longer guaranteed to have a feasible solution even if the discrete padding requirement is waived. If we set $T_{ls} = 0$ in $OPT2$, we obtain $a_i = A_i$ for all i . If we force all primary outputs i to have $A_i = DMAX$ (where $DMAX$ is a predefined value), it is equivalent to the *Balancing Problem* defined by Wong *et al.* in [81].

6.7 Results

We present the results on a set of combinational multi-level examples. The first set of experiments deals with the greedy algorithm (Procedure 6.3.2) and the linear programming approach

(Section 6.4). We use the linear program solver *lp_solve* - a sparse matrix implementation of the simplex algorithm due to M. Berkelaar [4]. We hasten to point out that it is not known if these combinational circuits were designed to operate in a sequential environment. Consequently the delay inserted and the area penalty is quite large. The purpose of these experiments is to examine the violation of the early arrival constraints due to the discrete nature of gates.

- Experiment 1: The circuits are unmapped and we use a unit delay fanout model. The circuits are optimized using *script.rugged* provided with the SIS distribution. The early and late arrivals at each primary input are set to 0. The late required time at each output is the longest path from an input to the output in question. The early required time at each output is set to be a fraction (0.3) of the longest path to it. The results are shown in Table 6.1. The area of the circuit is given in column 2. The area is measured in terms of the number of 2 input and/or gates and inverters. The area of an and/or gate is assigned to be unity and the area of an inverter is assigned to be 0.5 units. Columns 3 and 4 give the amount of delay that needs to be inserted to meet the delay constraints. A circuit with 200 gates typically gives rise to a linear program with 500 variables and a thousand constraints. Columns 5 and 6 show the time taken by the algorithms and columns 7 and 8 give the area penalty (area of the inserted buffers). The last two columns give a measure of violation³ of the short path bounds due to the discrete nature of delay insertion. We compute the percentage of violation at each output to the early required time at the output. The worst violation amongst all outputs is reported.
- Experiment 2: The second set of experiments are carried out on the same examples described above, but using an industrial standard cell library (see Table 6.2). The library has a set of 4 buffers with different areas and delays. The second column lists the initial area of the circuit. Columns 3 and 4 give the delay to be inserted for the relaxed problem. The next two columns give the time taken and columns 7 and 8 give area penalty after packing. The last two columns give a measure of violation.
- Experiment 3: This experiment deals with wave pipelining of circuits. Table 6.3 gives the data on unmapped circuits, while table 6.4 presents the data for mapped circuits. Column 2 gives the value of T_{l_s} without any delay insertion. We decrease the target T_{l_s} by a factor of 0.3 in column 3. The time taken by the procedure is the content of column 4. The last column gives the area penalty.

³See section 6.5.2

<i>name</i>	<i>Initial Area</i>	<i>delay padded</i>		<i>time(sec.)</i>		<i>Area Penalty</i>		<i>% Violation</i>	
		<i>G</i>	<i>LP</i>	<i>G</i>	<i>LP</i>	<i>G</i>	<i>LP</i>	<i>G</i>	<i>LP</i>
9symml	221.00	0.00	0.00	0.3	9.0	0.00	0.00	0.00	0.00
C432	287.50	971.58	187.92	10.2	19.5	224.50	65.00	6.71	2.82
C499	398.00	323.84	323.84	5.7	34.4	64.00	64.00	3.07	3.07
x4	305.00	882.02	430.78	18.9	31.1	197.00	145.50	12.53	12.53
i3	310.00	6.00	3.00	0.9	12.4	2.00	1.00	11.11	11.11
count	128.00	232.58	92.02	1.6	4.7	46.50	28.00	13.67	13.67
frg1	196.00	30.68	8.24	0.5	5.7	7.50	2.50	16.67	22.62
b9	132.50	44.28	26.40	0.8	4.1	9.00	7.00	35.19	16.67
x1	319.00	101.92	74.88	4.5	24.9	25.00	20.50	12.09	12.09
apex7	200.00	272.84	150.16	5.3	11.6	51.00	45.00	6.43	6.43
adder	145.00	907.80	271.72	2.8	7.0	237.00	94.00	4.42	6.81

G = Greedy procedure

LP = Linear Programming approach

Table 6.1: Delay insertion using unit delay fanout model

<i>name</i>	<i>Initial Area</i>	<i>delay padded</i>		<i>time(sec.)</i>		<i>Area Penalty</i>		<i>% Violation</i>	
		<i>G</i>	<i>LP</i>	<i>G</i>	<i>LP</i>	<i>G</i>	<i>LP</i>	<i>G</i>	<i>LP</i>
9symml	278.00	102.28	25.48	1.0	2.8	109.00	34.00	1.48	1.48
C432	290.00	749.34	175.08	4.8	5.0	1039.00	276.00	1.20	2.59
C499	689.00	310.54	309.64	4.4	16.9	425.00	395.00	0.17	0.17
x4	518.00	1169.11	530.27	19.0	25.1	1573.00	879.00	0.78	0.78
i3	178.00	0.00	0.00	0.2	0.9	0.00	0.00	0.00	0.00
count	225.00	804.62	315.26	3.4	3.5	1129.00	452.00	0.68	0.68
frg1	209.00	16.00	12.00	0.3	1.4	19.00	10.00	4.96	4.96
b9	182.00	46.04	29.04	0.8	2.2	29.00	22.00	2.78	2.78
x1	447.00	203.66	116.45	4.4	12.8	213.00	157.00	0.43	3.32
apex7	328.00	584.26	240.20	6.7	8.5	631.00	337.00	0.79	0.79
adder	285.00	765.49	254.53	4.4	7.3	1191.00	371.00	1.11	1.11

G = Greedy procedure

LP = Linear Programming approach

Table 6.2: Delay insertion using library delay model

<i>name</i>	<i>initial</i> <i>T_{ls}</i>	<i>final</i> <i>T_{ls}</i>	<i>time</i> <i>(sec.)</i>	<i>Area</i> <i>penalty</i>
9symml	18.40	12.88	14.1	13.00
C432	64.40	45.08	21.8	73.00
C499	24.80	17.36	45.0	96.00
x4	25.00	17.50	19.7	7.00
i3	1.60	1.12	28.0	0.00
count	24.20	16.94	4.9	18.00
frg1	15.20	10.64	10.5	6.00
b9	11.40	7.98	4.4	2.00
x1	13.20	9.24	27.9	5.00
apex7	19.80	13.86	10.1	8.50
adder	50.60	35.42	5.7	53.50

Table 6.3: Wave pipelining using unit delay fanout model

<i>name</i>	<i>initial</i> <i>T_{ls}</i>	<i>final</i> <i>T_{ls}</i>	<i>time</i> <i>(sec.)</i>	<i>Area</i> <i>penalty</i>
9symml	35.50	24.85	3.5	69.00
C432	67.60	47.32	5.3	304.00
C499	36.50	25.55	21.9	527.00
x4	33.90	23.73	14.2	52.00
i3	0.00	0.00	1.4	0.00
count	53.80	37.66	2.8	289.00
frg1	19.70	13.79	1.7	15.00
b9	13.40	9.38	2.1	12.00
x1	17.30	12.11	11.5	31.00
apex7	29.70	20.79	6.0	26.00
adder	52.20	36.54	5.7	157.00

Table 6.4: Wave pipelining using library delay model

It was observed that constructing a buffer tree (albeit a greedy tree) leads to savings of area by a factor of 2 to 3 in all the cases.

The experiments (1 and 2) show that the percentage violation depends to a large extent on the nature of the circuit. It is important to come up with circuit restructuring techniques that prepare the circuit for delay insertion. The large area penalties are due to outputs that have a small delay from the inputs. In order to meet an early required time constraint, set to a fraction of the longest delay, these outputs need to be buffered by a significant amount.

6.8 Discussion

The main motivation for this Chapter lies in the synthesis of sequential circuits (with level-sensitive memory elements) to meet a target clock period [63]. We have introduced the minimum padding problem in circuit synthesis. Under the pip-to-pin delay model, we give a theorem that prescribes necessary and sufficient conditions to solve the padding problem. We present two algorithms to solve the relaxed problem. For practical application of these algorithms, the approach is extended to handle gates with discrete delays. We show how a solution to the continuous problem can be used to approximate a solution to the discrete problem. We derive bounds on the amount of violation possible.

Chapter 7

Conclusions

This thesis has sought to explore problems in VLSI design that arise due to the sequential nature of synchronous circuits. As VLSI designs increase in size and complexity, it becomes impossible for the human mind to trace signals through wires and gates to ensure that memory elements “latch” correct values. Synthesis for improved performance is complicated by the fact that changing regions of a circuit can affect behavior in rest of the circuit.

We start out by examining the constraints for correct clocking of synchronous circuits. Chapter 3 presents a polynomial time algorithm for the verification of clock schedules. As a consequence to clock schedule verification, the optimization of clock schedules is considered in Chapter 4. An algorithm with complexity polynomial in the size of the circuit (exponential in the number of phases) is described. Proposed algorithms for clock schedule verification and optimization assume a pessimistic approach for data propagation; namely every path in the circuit is deemed capable of propagating an event. We feel that the existence of *false paths* (due to unreachable states) is more likely in sequential circuits than in purely combinational circuits. Using paths that actually propagate events for purposes of clock schedule verification and optimization is the next logical step. Efforts[56] are currently underway to approximate the optimum clock schedule which includes only sensitizable paths. The problem of clock schedule analysis with multiple frequency clock signals is another issue that merits investigation. Lastly, permitting arbitrary signals to control memory elements remains largely unsolved. We have made some progress in this direction using a restricted form of qualified clocks[28].

Chapter 5 investigates the resynthesis of multi-phase pipeline circuits to meet target clock periods. A novel approach using the notion of “minimal perturbation” is presented to solve the problem. The problem of extending resynthesis to general cyclic circuits remains complicated. A

reason is the lack of efficient path based delay optimization techniques.

Ensuring that short paths do not cause erroneous latching in sequential circuits leads to the problem of satisfying minimum delay requirements on combinational paths. In Chapter 6 we offer a technique to solve this problem using active delay insertion.

The effectiveness of the proposed algorithms has been demonstrated by experiments on some benchmarks. The implementations mirror the theoretical efficiency indicated by the outlined analyses.

All electronic designs, due to their sequential nature, require attention to some or all of the problems that are described in this thesis. It is sincerely hoped that designers are able to take advantage of the proposed automated synthesis and analysis techniques to speed the design process.

Appendix A

Optimality of the SmM Delay Model

In this Appendix, we seek to justify the use of the *simplified min-max* (SmM for short) delay model for a gate in the optimal clock schedule computation. We re-formulate the optimal clock schedule computation problem using the *consistent min-max* (CmM for short) delay model. Conditions are described when the two models are equivalent and when they differ. We describe a property which is a desired feature of clocking schemes in most designs, called the *robustness* property. Under this requirement, we show the simplified model to be equivalent to the consistent model.

A.1 Introduction

This section briefly reviews the clocking constraints in Chapter 2. Let us focus on the constraints for correct clocking using the SmM delay model and the CmM delay model. The min-max delay model may be used in two styles—

1. **simplified min-max delay model (SmM):** A circuit operates correctly if the sum of the delays along various paths satisfy certain linear inequalities. The delay of a gate in each inequality is determined by a *worst case* assignment. A gate delay is assigned the upper (lower) bound if it appears on the left (right) hand side of an \leq inequality with positive coefficients. So the same gate may be assigned different delays in different inequalities and sometimes in the same inequality. This has been the model of choice for all the previous work on optimal clocking [55, 69, 62].

2. consistent min-max delay model (CmM): In this case, the delay is determined as a part of the optimization step for the clock period. The delay of a gate is permitted to take on any value between the upper and lower bounds. The important thing is that each gate is forced to have the same delay in all equations. It is akin to assuming a uniform probability distribution for each gate and computing the minimum clock period at which the circuit will operate with unit probability.

For ease of presentation, only level-sensitive memory elements (latches) are assumed to be present in a circuit. The circuit is assumed to consist of two networks; a data network used for computations and a clock distribution network (skew network) that is used for the distribution of clock signals to the control inputs of memory elements. The data network is modeled by a directed graph $G(V, E)$, where each vertex i represents a latch i in the circuit. Every vertex i has a phase $\phi(i)$ associated with it. Let P_i be the path associated with the skew to latch i . A combinational path in the circuit consists of a sequence of gates, such that the output of each gate is an input to the gate following it. A combinational cycle is a path whose first and last gates are the same. There are no combinational cycles in a synchronous sequential circuit. Let p_{ij} be a combinational path from the output of latch i to the input of latch j . Each edge $i \rightarrow j$ in G denotes the existence of a path p_{ij} . A path in the graph is an alternating sequence of vertices and edges, beginning and terminating at a vertex; each edge being directed from the vertex preceding it to the vertex succeeding it in the sequence. A cycle in the graph is a path whose first and last vertices coincide. Each edge $i \rightarrow j$ in the graph has a value for K_{ij} as defined in Equation 4.1. If p is a path in the circuit, then $|p|$ denotes the sum of the K 's along the path (Equation 4.3). Let \mathcal{P}_{ij}^k denote the set of paths from vertex i to vertex j with exactly $|p| = k$ and containing no cycles. \mathcal{P}_{ij}^0 denotes the set of combinational paths from i to j . \mathcal{P}_{ij} is used for the set $\bigcup_k \mathcal{P}_{ij}^k$.

Each gate g_i has two real numbers associated with it, D_i and d_i , the upper and lower bounds on the delay of the gate. The actual delay of a gate g_i is a variable δ_i , which lies in the interval $[d_i, D_i]$. For sake of simplicity, we assume that the propagation delay through a latch is zero¹. The phase signal to each latch i undergoes a skew along path P_i . P_i is part of the clock distribution network of buffers/inverters and is disjoint from the data network. A clocking scheme is said to be robust if the circuit will operate without any latching errors under the given clocking scheme when every gate delay is within its specified bounds. Since gate delays cannot be accurately

¹Any delay associated with the memory elements can be abstracted out and modeled by buffers at the data input/output and clock input ports.

predicted *a priori* to fabrication, most designs require a *robust* clocking scheme.

A.1.1 CmM delay model

There are three types of constraints that arise in optimal clocking:

1. *internal* constraints— arising from the topological structure of the circuit and the distribution of delays on gates,
2. *model* constraints— arising from the clock model, forcing the solution to obey certain assumptions on the clock,
3. *external* constraints— arising due to environment constraints. The designer may constrain the duty cycle or enforce the clock events to be separated by a specific amount of time.

For the time being, we shall ignore the external timing constraints. The internal clocking constraints may be divided into two categories:

1. *long path* constraints— these force a minimum separation between clock events and arise due to the set-up constraint.

$$(a) \forall i, j \text{ and } \forall p \in \mathcal{P}_{ij}$$

$$e_{\phi(j)} + \left(\sum_{k \in \mathcal{P}_j} \delta_k \right) \geq s_{\phi(i)} + \left(\sum_{k \in \mathcal{P}_i} \delta_k \right) + \left(\sum_{k \in p} \delta_k \right) - |p|c + S. \quad (A.1)$$

$$(b) \forall \text{ cycles } C : j \rightarrow j, \text{ for correct operation we require (see Lemma 4.2.1)}$$

$$c \geq \frac{\sum_{k \in C} \delta_k}{|C|}. \quad (A.2)$$

2. *short path* constraints— these force a maximum separation between clock events and arise due to the hold constraint.

$$\forall p \in \mathcal{P}_{ij}^0$$

$$e_{\phi(j)} + \left(\sum_{k \in \mathcal{P}_j} \delta_k \right) + H \leq s_{\phi(i)} + \left(\sum_{k \in \mathcal{P}_i} \delta_k \right) + \left(\sum_{k \in p} \delta_k \right) + (1 - |p|)c \quad (A.3)$$

Note that for all $p \in \mathcal{P}_{ij}^0$, $|p| \in \{0, 1\}$.

We briefly mention the model constraints. We require

$$e_i \geq s_i \geq 0 \quad (A.4)$$

and

$$e_{i+1} \geq e_i \quad i = 1, \dots, l-1 \text{ and } e_l = c \quad (\text{A.5})$$

To find the optimum clock period for a given assignment of gate delays, we solve a linear program with the objective function $\min(c)$ and the constraints described above. Let us represent the linear constraints as $A\mathbf{x} \geq \mathbf{b}(\delta)$, where δ is a vector of gate delay assignments and \mathbf{x} is a vector consisting of clock events (variables) and \mathbf{b} is a vector consisting of linear functions of δ . A clocking scheme thus computed, is not necessarily robust.

A.1.2 Smm delay model

This formulation was suggested by Sakallah [55] and has been used by Szymanski [69] and Shenoy [62]. This is a conservative approach. The internal constraints translate as:

1. long path constraints-

$$(a) \quad \forall i, j \text{ and } \forall p \in \mathcal{P}_{ij}$$

$$e_{\phi(j)} + \left(\sum_{k \in P_j} d_k \right) \geq s_{\phi(i)} + \left(\sum_{k \in P_i} D_k \right) + \left(\sum_{k \in p} D_k \right) - |p|c + S. \quad (\text{A.6})$$

$$(b) \quad \forall \text{ cycles } C : j \rightarrow j \text{ in the graph}$$

$$c \geq \frac{\sum_{k \in C} D_k}{|C|}. \quad (\text{A.7})$$

2. short path constraints- We need $\forall p \in \mathcal{P}_{ij}^0$

$$e_{\phi(j)} + \left(\sum_{k \in P_j} D_k \right) + H \leq s_{\phi(i)} + \left(\sum_{k \in P_i} d_k \right) + \left(\sum_{k \in p} d_k \right) + (1 - |p|)c \quad (\text{A.8})$$

To find the optimum clock period using the Smm model, we solve a linear program. The objective function of this linear program is to minimize the clock period c , subject to the clocking constraints described above. Let us represent these constraints as $A\mathbf{x} \geq \hat{\mathbf{b}}$. Note that the right hand side of the inequalities is a vector of constants (as opposed to a vector of linear functions of δ in the CmM case). Paths P_i and P_j may have a common sub-path. Gates on the common sub-path will be assigned different delays in the same equation, *e.g.* in constraint A.6, a gate k common to P_i and P_j is assigned a delay of d_k on the left hand side and a delay of D_k on the right hand side of the inequality. In the CmM model, the gate k is assigned the same delay on both sides and the constraint is unaffected by the choice of the delay δ_k . Due to the conservative estimate, a clocking scheme found using the Smm model is robust. We seek to examine whether assigning delays inconsistently to the gates can cause an artificial increase of the clock period.

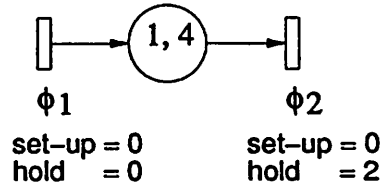


Figure A.1: Robust clocking: an example

A.2 Relating the two models

A.2.1 Preliminaries

Consider now, the optimal clocking problem using the CmM delay model. A lower bound on the clock period is found for the worst possible delay assignment to the gates. This leads to a *min-max* formulation described below. Intuitively, we consider the optimal clock computation to be a game played by two players, A and B respectively. A chooses a delay assignment to the gates consistent with the bounds, namely for each gate i he picks a $\delta_i \in [d_i, D_i]$. His sole objective is to obtain as large a clock period as possible. A is called the *maximizing* player (or the adversary). The assignments to the gate delays are then made known to B, who uses a LP (using the constraints for the CmM model) to find the minimum clock period at which the circuit will operate. B is called the *minimizing* player (and represents the algorithm). If there are g gates, A picks a point in a subspace of R^g defined by $\prod_{i=1}^g [d_i, D_i] = \Omega$. Mathematically, we are interested in solving the following problem: $\min_{\delta \in \Omega} \max c$, subject to $Ax \geq b(\delta)$.

Let us briefly examine the notion of a *robust* clocking scheme in the context of the two delay models. Consider the example shown in Figure A.1. The gate has a max-delay of 4 and a min-delay of 1. The constraints are (assuming $e_1 \leq e_2$)

- Smm model:
 - a1: $e_2 \leq s_1 + (1 - 2) + c$ short path constraint
 - a2: $e_2 \geq s_1 + 4$ long path constraint
 - a3: $e_1 \leq e_2$ model constraint
 - a4: $s_1 \leq e_1$ model constraint
 - a5: $s_2 \leq e_2$ model constraint
 - a6: $e_2 = c$ model constraint
 - a7, a8: $s_1, s_2 \geq 0$ model constraint

Constraints a1 and a2 imply, $c \geq 5$. So a clocking scheme with $s_1 = 1, e_1 = 2, s_2 = 3, e_2 = 5$, will work for all possible delays of the gate within the bound $[1, 4]$.

- CmM model: Let δ be the delay of the gate.

b1:	$e_2 \leq s_1 + (\delta - 2) + c$	short path constraint
b2:	$e_2 \geq s_1 + \delta$	long path constraint
b3:	$e_1 \leq e_2$	model constraint
b4:	$s_1 \leq e_1$	model constraint
b5:	$s_2 \leq e_2$	model constraint
b6:	$e_2 = c$	model constraint
b7, b8:	$s_1, s_2 \geq 0$.	model constraint

From b1 and b6 we conclude $s_1 \geq 2 - \delta$. Together with b7, we obtain $s_1 \geq \max(0, 2 - \delta)$. This leads to $c \geq \max(\delta, 2)$. Let us concentrate on the values for s_1 and e_2 . They are—

$$s_1 = \max(0, 2 - \delta)$$

$$e_2 = \max(2, \delta).$$

The values of the rise/fall times of phases depend on the actual delay of the gate. The worst case delay assignment using the CmM model gives a clock period of 4, but with the caveat that as the gate delay changes, we are permitted to shift the rise of phase 1. This is a drawback, since we would like a clocking scheme that is robust. Intuitively, it is clear that there can be a gap between the clock periods found using the two models. The central question is whether the imposition of *robustness* still maintains the gap.

A.2.2 Equivalence of the two models

For the rest of the discussion we will ignore the model constraints and external constraints since they are the same in both models and affect the feasible regions in the same manner. Gates in circuits can be divided into two groups;

1. Data: these are gates that appear on combinational paths from a latch to another latch, and
2. Skew: these are inverters/buffers that are used in the clock distribution.

The two groups are disjoint, hence no gate can ever appear along a data path, say p ($p : i \rightarrow j$) and along a skew path, namely P_i or P_j . This implies that the delay of each data gate can appear as a variable only once in each constraint. Before proceeding further, we need to modify the manner in which Smm constraints are obtained. We know that each data gate can appear only once in each constraint. The skew paths P_i and P_j may have a some gate/wires in common. Since the delays will appear on opposite sides of the inequality, the common delay should be ignored in both

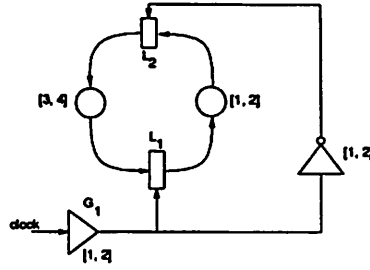


Figure A.2: Skew effect: an example

models. However the Smm model assigns each such gate, different delays in the same constraint. We make a modification which ignores such gates. Henceforth each gate can appear only once in each inequality in both models.

To observe the effect of assigning inconsistent delays to skew gates, consider the circuit shown in Figure A.2. Let P_1, P_2 be the paths from the clock to latches L_1, L_2 . Let the clock rise at s and fall at c . The control signal to L_1 rises in the interval $[s + 1, s + 2]$ and falls in the interval $([c + 1, c + 2] \bmod c)$. Similarly the control signal to L_2 rises in $([c + 2, c + 4] \bmod c)$ and falls in $[s + 2, s + 4]$. If the longest and shortest paths from clock to latch are constructed in a simple manner as described in the Smm model, then the constraints (only long path) are

- 1: $c \geq 6$ long path cycle constraint
- 2: $c + s + 2 \geq s + 2 + 2$ long path $L_1 \rightarrow L_2$
- 3: $c + 1 \geq c + 4 + 4 - c$ long path $L_2 \rightarrow L_1$.

The constraints imply $c \geq 7$. If we make use of the fact that P_1 and P_2 have a common gate G_1 , the constraints become

- 1: $c \geq 6$ long path cycle constraint
- 2: $c + s + 1 \geq s + 2$ long path $L_1 \rightarrow L_2$
- 3: $c \geq c + 2 + 4 - c$ long path $L_1 \rightarrow L_2$,

implying $c \geq 6$. Thus the latter set gives a smaller clock period. Henceforth the Smm model constraints will mean the constraints with no gate (data gate or skew gate) appearing more than once in each constraint. This restriction forces the gate delay to be consistent in each constraint but not necessarily consistent in all the constraints.

Let $\mathcal{X}(\delta)$ denote the set of feasible clock schedules, under the Cmm delay model, for a delay assignment $\delta \in \Omega$. Hence $\mathcal{X}(\delta) = \{x | Ax \geq b(\delta)\}$.

Lemma A.2.1 *The set $\mathcal{X}(\delta)$ is closed.*

Proof Let the $\{x_n\}$ denote a convergent sequence in $\mathcal{X}(\delta)$. The measure of distance (metric) we shall use is the max metric. For any two points x_i and x_j , define the distance between them as

$$\rho(x_i, x_j) = \max_{k=1, \dots, 2l} |[x_i]_k - [x_j]_k|, \quad (\text{A.9})$$

where $[x_i]_k$ is the k^{th} component of x_i . Let $\{x_n\}$ converge to x^* . We need to show $x^* \in \mathcal{X}(\delta)$. Assume $x^* \notin \mathcal{X}(\delta)$. Then there must be some constraint in $Ax \geq b(\delta)$ which is violated at x^* by an amount η . Let the constraint be denoted by $ax \geq b(\delta)$, where a is the vector of coefficients and $b(\delta)$ is the corresponding component of $b(\delta)$. We know $ax^* = b(\delta) - \eta$, with $\eta > 0$. Let α be the coefficient with largest magnitude in a . Now $\{x_n\} \rightarrow x^*$, implying $\forall \epsilon > 0, \exists N$, such that $\rho(x_n, x^*) < \epsilon, \forall n > N$. Set $\epsilon = \frac{\eta}{4\alpha l}$. Thus there must be some N for which $\rho(x_n, x^*) < \frac{\eta}{4\alpha l}$, for all $n > N$. But $ax_n = a(x^* + x_n - x^*) = ax^* + a(x_n - x^*)$. Implying $ax_n = b(\delta) - \eta + a(x_n - x^*)$. The largest positive value $a(x_n - x^*)$ can take is less than $\frac{\eta}{2}$. Thus $ax_n < b(\delta) - \frac{\eta}{2} \Rightarrow ax_n < b(\delta)$ and x_n is not a feasible clock schedule. Contradiction to the fact $x_n \in \mathcal{X}(\delta)$. ■

Proposition A.2.2 *If a clock schedule \hat{x} is robust, then $\hat{x} \in \bigcap_{\delta \in \Omega} \mathcal{X}(\delta)$.*

The above proposition follows from the definition of robustness given in Section A.1. Note that since each set $\mathcal{X}(\delta)$ is closed, the intersection of a family of closed sets is closed and well defined.

Lemma A.2.3 *Let $\mathcal{X}(\delta) = \{x | Ax \geq b(\delta)\}$, then $\bigcap_{\delta \in \Omega} \mathcal{X}(\delta) = \{x | Ax \geq \max_{\delta \in \Omega} b(\delta)\}$, where the max on a vector function $b(\delta)$ is the vector of the component-wise maxima.*

Proof The proof is divided into two parts.

- $\bigcap_{\delta \in \Omega} \mathcal{X}(\delta) \subset \{x | Ax \geq \max_{\delta \in \Omega} b(\delta)\}$: Let $y \in \bigcap_{\delta \in \Omega} \mathcal{X}(\delta)$. Consider any constraint, say $ax \geq b(\delta)$. Now since $y \in \bigcap_{\delta \in \Omega} \mathcal{X}(\delta)$, we conclude $ay \geq b(\delta)$ for all $\delta \in \Omega$, implying $ay \geq \max_{\delta \in \Omega} b(\delta)$. As this is true for each constraint, the result holds for all the constraints.
- $\{x | Ax \geq \max_{\delta \in \Omega} b(\delta)\} \subset \bigcap_{\delta \in \Omega} \mathcal{X}(\delta)$: If y is a solution to $Ax \geq \max_{\delta \in \Omega} b(\delta)$, then $Ay \geq b(\delta)$ for all $\delta \in \Omega$. Thus y is feasible to the constraints of the CmM delay model. Hence it must be in the intersection of all the $\mathcal{X}(\delta)$'s. ■

Let \mathcal{Y} denote the set of feasible clock schedules under the Smm delay model. By definition $\mathcal{Y} = \{x | Ax \geq \hat{b}\}$.

Theorem A.2.4 $\mathcal{Y} = \bigcap_{\delta \in \Omega} \mathcal{X}(\delta)$.

Proof By virtue of previous lemma (Lemma A.2.3), we know $\bigcap_{\delta \in \Omega} \mathcal{X}(\delta) = \{\mathbf{x} | \mathbf{A}\mathbf{x} \geq \max_{\delta \in \Omega} \mathbf{b}(\delta)\}$. Since $\mathcal{Y} = \{\mathbf{x} | \mathbf{A}\mathbf{x} \geq \hat{\mathbf{b}}\}$. It remains to show that $\hat{\mathbf{b}} = \max_{\delta \in \Omega} \mathbf{b}(\delta)$. Consider constraint A.1:

$$\begin{aligned} b(\delta) &= \sum_{k \in P_i} \delta_k + \sum_{k \in P} \delta_k - \sum_{k \in P_j} \delta_k + S \\ \Rightarrow \max_{\delta \in \Omega} b(\delta) &= \sum_{k \in P_i} D_k + \sum_{k \in P} D_k - \sum_{k \in P_j} d_k + S \\ \Rightarrow \max_{\delta \in \Omega} b(\delta) &= \hat{b} \end{aligned}$$

The same is true for constraints A.2 and A.3. ■

Corollary A.2.5 *The optimum robust clock schedules under the CmM delay model are identical to the optimum clock schedules under the SmM delay model.*

Proof Theorem A.2.4 proves that the feasible regions for the two problems are identical. So the infimum of x_{2l} in the feasible region for the two problems will be identical. Moreover, since the feasible regions are closed, the infimum will be attained, so we can replace the infimum by the more common minimum. Hence the set of clock schedules \mathbf{x} with minimum value of $x_{2l} = c$ (the clock period) must also be the same. ■

As a consequence, we conclude that if the clock period of a circuit is less than the value given by the SmM model, then there must exist a consistent assignment of delays to gates (within their respective bounds), for which at least one of the clocking constraints is violated. Said otherwise, the adversary A can always select a consistent assignment to make the clock period be no less than the value predicted by the SmM model.

Appendix B

Quadratic Optimization

A quadratic optimization problem (henceforth referred to as *QOPT*) is of the form

$$\begin{aligned} QOPT : \min[(x - x_0)^T B(x - x_0) = f(x)] \\ Ax \leq b. \end{aligned}$$

The matrix B is positive definite. A naive approach is to use a constrained quadratic optimization algorithm. This requires projecting the gradient onto tangent planes defined by the inequalities with zero slack. Solving the primal problem in this manner would also require an initial feasible point. Instead we use a variant of the gradient projection algorithm on the dual problem to *QOPT*. The advantages of this approach are twofold; computational tractability, and choice of an arbitrary starting point.

We shall first consider detecting infeasibility of the constraints. Note that the feasible region is a cone in the positive orthant with its vertex at \hat{d} (or at the point $\hat{H} = (H, 2H, \dots, (n-1)H, H, \dots, (n-2)H, \dots, H)$ obtained by substituting $d_{i;i+1} = H$ in Equation 5.4 in the general case).

Lemma B.1.6 *If the constraint region P is not empty then $\hat{d} \in P$.*

Proof True by inspection of the constraints. ■

If the region is infeasible then we need to extend the constraints with the d 's as variables. The lemma now becomes:

Lemma B.1.7 *If the extended constraint region is not empty then $\hat{H} \in P$.*

Lemmas similar to Lemma B.1.6 and Lemma B.1.7 can be proven easily for the case when assumption 1 in Section 5.3 is relaxed (Section 5.6). Hence we can detect infeasibility by setting the variables to the appropriate values and checking if the constraints are satisfied. We shall henceforth assume that $P \neq \emptyset$.

Lemma B.1.8 *Every local minimum of $f(x)$ is a global minimum.*

Proof A consequence of the fact that $f(x)$ is a convex function of x and the feasible region $Ax \leq b$ is a convex set. ■

Let the Lagrangian $\mathcal{L}(\lambda)$ be defined as the unconstrained problem

$$\mathcal{L}(\lambda) = -\lambda^T b + \min_x \left(\frac{1}{2} (x - x_0)^T B (x - x_0) + \lambda^T A x \right) \quad (\text{B.1})$$

for all $x \in R^n$ and $\lambda \geq 0$. The variables denoted by λ are known as the **Lagrange multipliers**. Quite clearly

$$\begin{aligned} \mathcal{L}(\lambda) &\leq \min_{Ax \leq b} \frac{1}{2} (x - x_0)^T B (x - x_0) \\ \Rightarrow \max_{\lambda \geq 0} \mathcal{L}(\lambda) &\leq \min_{Ax \leq b} \frac{1}{2} (x - x_0)^T B (x - x_0) \end{aligned}$$

This is a restatement of the fact that the primal problem *QOPT*, has an optimum value greater than or equal to the optimum of the dual $\{\max_{\lambda \geq 0} \mathcal{L}(\lambda)\}$. Given any λ define

$$x_\lambda = x_0 - B^{-1} A^T \lambda. \quad (\text{B.2})$$

It should be noted that given a λ , the problem $\{\min_x \frac{1}{2} (x - x_0)^T B (x - x_0) + \lambda^T A x\}$ has a unique solution given by x_λ . So we can write $\mathcal{L}(\lambda)$ as

$$\begin{aligned} \mathcal{L}(\lambda) &= -\lambda^T b + \min_x \left(\frac{1}{2} (x - x_0)^T B (x - x_0) + \lambda^T A x \right) \\ \Rightarrow \mathcal{L}(\lambda) &= -\lambda^T b + \frac{1}{2} (x_\lambda - x_0)^T B (x_\lambda - x_0) + \lambda^T A x_\lambda \\ \Rightarrow \mathcal{L}(\lambda) &= -\lambda^T b + \frac{1}{2} (-B^{-1} A^T \lambda)^T B (-B^{-1} A^T \lambda) + \lambda^T A (x_0 - B^{-1} A^T \lambda). \end{aligned}$$

Simplifying results in

$$\mathcal{L}(\lambda) = -\lambda^T b - \frac{1}{2} \lambda^T A B^{-1} A^T \lambda + \lambda^T A x_0. \quad (\text{B.3})$$

Lemma B.1.9 $\mathcal{L}(\lambda)$ is concave and continuous.

Proof The Lagrangian is concave because its Hessian $(-AB^{-1}A^T)$ is negative definite. It is clearly continuous. ■

The Kuhn-Tucker conditions for optimality of the primal problem are

$$B(x^* - x_0) + \lambda^* A = 0 \quad \text{KT1}$$

$$\lambda^* \geq 0 \quad \text{KT2}$$

$$\lambda^*(Ax^* - b) = 0 \quad \text{KT3}$$

$$Ax^* \leq b. \quad \text{KT4}$$

We now show that it is possible to construct an ascent algorithm to compute the maximum of $\mathcal{L}(\lambda)$. Let $\hat{\lambda}$ be the optimum to the dual then, it is easy to show $(x_{\hat{\lambda}}, \hat{\lambda})$ satisfy the Kuhn-Tucker conditions for optimality of the primal problem.

Lemma B.1.10 *At any point λ_k the dual has an ascent direction given by $Ax_{\lambda_k} - b$.*

Proof

$$\begin{aligned} \mathcal{L}(\lambda) &= -\lambda^T b - \frac{1}{2} \lambda^T AB^{-1}A^T \lambda + \lambda^T Ax_0 \\ \Rightarrow \nabla \mathcal{L}(\lambda) &= -b - AB^{-1}A^T \lambda + Ax_0 \\ \Rightarrow \nabla \mathcal{L}(\lambda) &= -b + A(x_0 - B^{-1}A^T \lambda) \\ \Rightarrow \nabla \mathcal{L}(\lambda) &= -b + Ax_{\lambda}. \end{aligned}$$

We now present the algorithm in Procedure B.1.1, which is based on gradient projection. The advantage of this procedure is that the gradient projection for the dual problem is a trivial operation because of the non-negativity constraints. In the primal problem, the constraint matrix can be fairly complex requiring a complicated projection operation.

Procedure B.1.1

$\lambda_0 = \text{initial guess (typically 0)}$

- (i) $x_k = x_0 - B^{-1}A^T \lambda_k$
- (ii) $[h_k]_i = \begin{cases} [\nabla \mathcal{L}(\lambda_k)]_i & \text{if } [\lambda_k]_i > 0 \\ \max\{0, [\nabla \mathcal{L}(\lambda_k)]_i\} & \text{if } [\lambda_k]_i = 0 \end{cases}$
- (iii) if $h_k \neq 0$ {

- $\alpha_1 = \max\{\alpha : \lambda_k + \alpha h_k \geq 0\}$
 $\lambda_{k+1} = \arg \max_{0 \leq \alpha \leq \alpha_1} (\mathcal{L}(\lambda_k + \alpha h_k))$
increment k , go to (i)

} else {

- $\hat{x} = x_k, \hat{\lambda} = \lambda_k$ *return*

}

The vector h is called the gradient of the Lagrangian \mathcal{L} for a reason that will soon be clear. To find α in each iteration we resort to a line search algorithm in the direction given by the gradient of \mathcal{L} .

We need to show that $\hat{\lambda}, \hat{x}$ are optimum to the primal problem. Note the following

- $\alpha_1 > 0$: Assume $\lambda_k \geq 0$. To ensure $\lambda_{k+1} \geq 0$ we need $\alpha = \max_{i: [h_k]_i < 0} \frac{[\lambda_k]_i}{[h_k]_i}$. If $h_k \geq 0$, $\alpha_1 = +\infty$. If $[h_k]_i < 0$ then $[\lambda_k]_i > 0$, by definition. Hence $\alpha_1 > 0$. This requires that the initial guess $\lambda_0 \geq 0$.
- If $h_k \neq 0, \exists \alpha : \alpha_1 \geq \alpha > 0 : \mathcal{L}(\lambda_k + \alpha h_k) > \mathcal{L}(\lambda_k)$. First note that $\nabla^2 \mathcal{L}(\lambda_k) = -AB^{-1}A^T$, a negative definite matrix. Since the Lagrangian is continuous, we can expand using the Taylor's series to obtain

$$\mathcal{L}(\lambda_k + \alpha h_k) = \mathcal{L}(\lambda_k) + \alpha h_k \nabla \mathcal{L}(\lambda_k) + \alpha^2 h_k (-AB^{-1}A^T) h_k$$

Note that if $h_k \neq 0$, then $h_k \nabla \mathcal{L}(\lambda_k) > 0$ from the definition of h_k . Since the last term arises from the Hessian (the matrix of second derivatives of the objective function) — a negative definite matrix, we conclude that for a sufficiently small α , h_k is a direction of increase (and hence h_k is called the gradient).

- The objective function $\mathcal{L}(\lambda)$ is monotone increasing
- $h_k = 0$ implies the following must hold for each row of A , say i :

- $[\nabla \mathcal{L}(\hat{\lambda})]_i = 0$ i.e. $[A\hat{x} - b]_i = 0$ and $[\hat{\lambda}]_i \geq 0$ or
- $[\nabla \mathcal{L}(\hat{\lambda})]_i < 0$ i.e. $[A\hat{x} - b]_i < 0$ and $[\hat{\lambda}]_i = 0$.

This implies $A\hat{x} \leq b$, so \hat{x} is feasible. In addition we find $\sum_i [\hat{\lambda}]_i [\nabla \mathcal{L}(\hat{\lambda})]_i = 0$, hence we conclude $\hat{\lambda}(A\hat{x} - b) = 0$. The first Kuhn-Tucker condition (KT1) is satisfied at each iteration in step (i) of Procedure B.1.1. The second condition (KT2) is implicitly satisfied by the

definition of α_1 . The third (KT3) and fourth (KT4) conditions are met by $(\hat{x}, \hat{\lambda})$, and thus \hat{x} is optimum to the primal problem.

As an example, consider the following quadratic problem taken from Section 5.7.1:

$$\begin{aligned} \min(x-3)^2 + (y-4)^2 \\ x \leq 3 \quad \text{constraint 1} \\ x + y \leq 6 \quad \text{constraint 2} \\ y \leq 4 \quad \text{constraint 3.} \end{aligned}$$

Let $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ be the Lagrange multipliers associated with the constraints. From Equation B.2, the following equations for x and y are obtained.

$$x = 3 - \lambda_1 - \lambda_2 \quad (\text{B.4})$$

$$y = 4 - \lambda_2 - \lambda_3. \quad (\text{B.5})$$

The Lagrangian function from Equation B.3 is

$$\mathcal{L}(\lambda) = \lambda_2 - \frac{\lambda_1^2}{2} - \lambda_1\lambda_2 - \lambda_2^2 - \lambda_2\lambda_3 - \frac{\lambda_3^2}{2}. \quad (\text{B.6})$$

The gradient of the Lagrangian from Lemma B.1.10 is given by

$$\nabla \mathcal{L}(\lambda) = (x-3, x+y-6, y-4). \quad (\text{B.7})$$

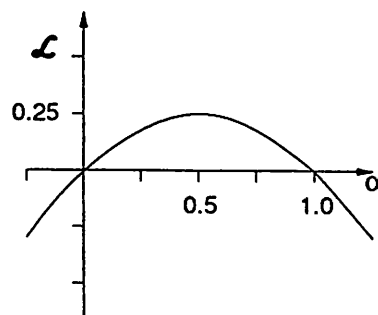


Figure B.1: Plot of \mathcal{L} versus α

The values of x , y and the Lagrange multipliers for each iteration are shown in Table B.1. The optimum solution is obtained in two iterations. In the first iteration $\lambda = (0, 0, 0)$ and the values

<i>iteration</i>	<i>x</i>	<i>y</i>	λ	<i>h</i>	α_1	α
1	3	4	(0,0,0)	(0,1,0)	∞	0.5
2	2.5	3.5	(0,0.5,0)	(0,0,0)	-	

Table B.1: Iterations for quadratic programming

of x and y are calculated using Equation B.4 and Equation B.5 respectively. The gradient of the Lagrangian (from Equation B.7) is computed to be $(0, 1, 0)$. A feasible direction of ascent for the dual (h) is $(0, 1, 0)$. It is easy to see that $\alpha_1 = \infty$. Substituting $\lambda + \alpha h = (0, \alpha, 0)$ in Equation B.6, we find $\mathcal{L}(\lambda + \alpha h) = \alpha - \alpha^2$. A plot of \mathcal{L} versus α (Figure B.1) shows that the maximum is attained at $\alpha = 0.5$. The Lagrange multipliers for the second iteration are $(0, 0.5, 0)$. After updating the values of x and y , we find that the new gradient of the Lagrangian is $(-0.5, 0, -0.5)$. Consequently there is no feasible direction of ascent for the dual at $x = 2.5, y = 3.5$ — the point of optimality to the problem.

Bibliography

- [1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.
- [2] K. A. Bartlett, G. Boriello, and S. Raju. Timing optimization of multi-phase sequential logic. *IEEE Transactions on Computer-Aided Design*, pages 51–62, 1991.
- [3] H. Behnke *et al.*, editor. *Fundamentals of Mathematics*, volume 1, The Real Number System and Algebra. MIT Press, 1987.
- [4] M.R.C.M. Berkelaar and J. A. G. Jess. Private communication. June, 1993.
- [5] C. L. Berman, J. L. Carter, and K. F. Day. The Fanout Problem: From Theory to Practice. In *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, pages 69–99, 1989.
- [6] T. M. Burks, K. Sakallah, and T. N. Mudge. Identification of Critical Paths in Circuits with level-Sensitive Latches. In *Proceedings of the International Conference on Computer-Aided Design*, pages 137–141. IEEE, 1992.
- [7] T.-A. Chu. Synthesis of Hazard-free Control Circuits from Asynchronous Finite State Machine Specifications. In *Tau 92*, 1992.
- [8] M. R. Dagenais and N. C. Rumin. Automatic Determination of Optimal Clocking Parameters in MOS VLSI Circuits. In *Advanced Research in VLSI: Proc. of the 5th MIT Conference*, pages 19–33, 1988.
- [9] J. D. Darringer, D. Brand, W. H. Joyner, and L. Trevillyan. LSS: A System for Production Logic Synthesis. Technical report, IBM Journal of Research and Development, 1984.

- [10] G. De Micheli. Synchronous logic synthesis: Algorithms for cycle-time minimization. In *IEEE Transactions on Computer-Aided Design*, pages 63–73, 1991.
- [11] S. Dey, F. Brglez, and G. Kedem. Partitioning Sequential Circuits for Logic Optimization. In *Proceedings of the International Workshop on Logic Synthesis*, 1991.
- [12] S. Dey, M. Potkonjak, and S. G. Rothweiler. Performance Optimization of Sequential Circuits by Eliminating Retiming Bottlenecks. In *Proceedings of the International Conference on Computer-Aided Design*, pages 504–509, 1992.
- [13] D. H. C. Du, S. H. C. Yen, and S. Ghanta. On the General False Path Problem in Timing Analysis. In *Proceedings of the Design Automation Conference*, pages 555–560. IEEE/ACM, 1989.
- [14] E. B. Eichelberger and T. W. Williams. A Logic Design Structure for LSI Testability. In *Proceedings of the Design Automation Conference*, pages 462–468, 1977.
- [15] J. Fishburn. Clock Skew Optimization. *AT&T Bell Laboratories, Murray Hill NJ 07974*, 1981.
- [16] J. P. Fishburn. A Depth-Decreasing Heuristic for Combinational Logic. In *Proceedings of the Design Automation Conference*, pages 361–364, 1990.
- [17] J. P. Fishburn. LATTIS: An Iterative Speedup Heuristic for Mapped Logic. In *Proceedings of the Design Automation Conference*, pages 488–491, 1992.
- [18] J. P. Fishburn and A. E. Dunlop. TILOS: A Posynomial Programming Approach to Transistor Sizing. In *Proceedings of the International Conference on Computer-Aided Design*, pages 326–328, 1985.
- [19] M. R. Garey and D. S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [20] D. Gregory, K. Bartlett, A. de Geus, and G. Hachtel. SOCRATES: A System for Automatically Synthesizing and Optimizing Combinational Logic. In *Proceedings of the Design Automation Conference*, pages 79–85. IEEE/ACM, 1986.
- [21] R. B. Hitchcock. Timing Verification and Timing Analysis Program. In *25 Years of Electronic Design Automation*. IEEE/ACM, 1988.

- [22] R. B. Hitchcock, G. L. Smith, and D. D. Cheng. Timing Analysis of Computer Hardware. Technical report, IBM, 1982.
- [23] D. A. Hodges and H. G. Jackson. *Analysis and Design of Digital Integrated Circuits*. McGraw-Hill Book Co., 2 edition, 1988.
- [24] A. Ishii and C. E. Leiserson. A Timing Analysis of Level-Clocked Circuitry. In *Advanced Research in VLSI: Proc. of the 7th MIT Conference*, 1990.
- [25] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou. Optimizing Two-Phase Level-Clocked Circuitry. In *Advanced Research in VLSI*, 1992.
- [26] N. P. Jouppi. *Timing Verification and Performance Improvement of MOS VLSI Designs*. PhD thesis, Stanford University, Stanford CA-94305, October 1984.
- [27] R. Kamikawai, M. Yamada, T. Chiba, K. Furumaya, and Y. Tsuchiya. A Critical Path Delay Check System. In *Proceedings of the Design Automation Conference*, pages 118–123. IEEE/ACM, 1981.
- [28] M. Kawarabayashi, N. Shenoy, and A. Sangiovanni-Vincentelli. A Verification Technique for Gated Clock. In *Proceedings of the Design Automation Conference*. IEEE/ACM, 1993.
- [29] K. Keutzer. DAGON: Technology Binding and Local Optimization by DAG Matching. In *Proceedings of the Design Automation Conference*, pages 341–347. ACM/IEEE, 1987.
- [30] T. I. Kirkpatrick and N. R. Clark. PERT as an Aid to Logic Design. Technical report, IBM Journal of Research and Development, 1966.
- [31] L. Lavagno, N. Shenoy, and A. Sangiovanni-Vincentelli. Linear Programming for Hazard Elimination in Asynchronous Circuits. In *Journal of VLSI Signal Processing*, 1993.
- [32] E. L. Lawler. *Combinatorial Optimization: networks and Matroids*. Holt, Rinehart and Winston, 1976.
- [33] C. E. Leiserson and J. B. Saxe. Optimizing Synchronous Systems. In *Journal of VLSI and Computer Systems*, pages 41–67, 1983.
- [34] Y-M. Li and M. A. Jabri. A Zero-Skew Clock Routing Scheme for VLSI Circuits. In *Proceedings of the International Conference on Computer-Aided Design*, pages 458–463, 1992.

- [35] B. Lockyear and C. Ebeling. Optimal Retiming of Multi-Phase Level-Clocked Circuits. In *Advanced Research in VLSI*, 1992.
- [36] F. Mailhot and G. De Micheli. Technology Mapping using Boolean Matching and Don't care Sets. In *Proceedings of the European Design Automation Conference*, 1990.
- [37] S. Malik, E. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and Resynthesis: Optimization of Sequential Networks with Combinational Techniques. In *Proceedings of the Hawaii International Conference on System Sciences*, pages 397–406, January 1990.
- [38] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli. Performance optimization of pipelined circuits. In *Proceedings of the International Conference on Computer-Aided Design*, pages 410–413. IEEE, 1990.
- [39] D. Marple. *Performance Optimization of Digital VLSI design*. PhD thesis, Stanford University, 1986.
- [40] P. C. McGeer. *On the Interaction of Functional and Timing Behavior of Combinational Circuits*. PhD thesis, University of California, Berkeley, 1989.
- [41] P.C. Mcgeer, A. Saldanha, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. Timing Analysis and Delay-Fault Test Generation using Path-Recursive Functions. In *Proceedings of the International Conference on Computer-Aided Design*, pages 180–183. IEEE, 1991.
- [42] T. M. McWilliams and L. C. Widdoes Jr. SCALD: Structured Computer-Aided Logic Design. In *Proceedings of the Design Automation Conference*, pages 271–277. IEEE/ACM, 1978.
- [43] C. A. Mead and L. A. Conway. *Introduction to VLSI Systems*. Addison-Wesley, 1980.
- [44] N. Megiddo. Towards a Genuinely Polynomial Algorithm for Linear Programming. In *Society for Industrial and Applied Mathematics*, pages 347–353, 1983.
- [45] K. G. Murty. *Linear Programming*. John Wiley and Sons, 1983.
- [46] L Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. Memorandum No. UCB/ERL M85/90, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, November 1985.

- [47] S. M. Nowick and D. L. Dill. Exact Two-level Minimization of Hazard-free Logic with Multiple-input Changes. In *Proceedings of the International Conference on Computer-Aided Design*, pages 626–630. IEEE, 1992.
- [48] J. K. Ousterhout. A Switch-Level Timing Verifier for Digital MOS VLSI. *IEEE Transactions on Computer-Aided Design*, CAD-4(3):336–349, July 1985.
- [49] P. G. Paulin and F. Poirot. Logic Decomposition Algorithms for the Timing Optimization of Multi-Level Logic. In *Proceedings of the International Conference on Computer Design*, pages 329–33, 1989.
- [50] D. J. Pilling and H. B. Sun. Computer-Aided Prediction of Delays in LSI Logic Systems. In *Proceedings of the Design Automation Conference*, pages 182–186. IEEE/ACM, 1973.
- [51] E. Polak, R. Trahan, and D. Q. Mayne. Combined phase I - phase II Methods of Feasible Directions. *Mathematical Programming*, 17(1):61–73, 1971.
- [52] J. Rubinstein, P. Penfield, and M. A. Horowitz. Signal Delay in RC Tree Networks. In *IEEE Transactions on CAD*, pages 119–127, July 1983.
- [53] R. Rudell. *Logic Synthesis for VLSI Design*. PhD thesis, University of California, Berkeley, 1989.
- [54] K. Sakallah, T. Mudge, and O. A. Olukotun. Analysis and Design of Latch-Controlled Synchronous Circuits. In *Proceedings of the Design Automation Conference*, pages 111–117. IEEE/ACM, 1990.
- [55] K. Sakallah, T. N. Mudge, and O. A. Olukotun. $CheckT_c$ and $minT_c$: Timing Verification and Optimal Clocking of Synchronous Digital Circuits. In *Proceedings of the International Conference on Computer-Aided Design*, pages 552–555. IEEE, 1990.
- [56] A. Saldanha, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Approximating Optimum Functional Clock Schedules. in preparation, July 1992.
- [57] S. Sapatnekar. *A Convex Programming Approach to Problems in VLSI Design*. PhD thesis, University of Illinois, Urbana-Champaign, 1992.

- [58] T. Sasaki, A. Yamada, T. Aoyama, K. Hasegawa, S. Kato, and S. Sato. Hierarchical Design Verification for Large Digital Systems. In *Proceedings of the Design Automation Conference*, pages 105–112. IEEE/ACM, 1981.
- [59] R. B. Segal. BDSYN: Logic Description Translator; BDSIM: Switch Level Simulator. Master's thesis, University of California, Berkeley, May 1987. ERL Memo. M87/33.
- [60] E. Sentovich *et al.* Sequential Circuit Design Using Synthesis and Optimization. In *Proceedings of the International Conference on Computer Design*, 1992.
- [61] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. A Pseudo-Polynomial Algorithm for Verification of Clocking Schemes. In *Tau 92*, 1992.
- [62] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Graph Algorithms for Efficient Clock Schedule Optimization. In *Proceedings of the International Conference on Computer-Aided Design*, 1992.
- [63] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Resynthesis of Multi-Phase Pipelines. In *Proceedings of the Design Automation Conference*, 1993.
- [64] J. Shyu, J. P. Fishburn, A. E. Dunlop, and A. Sangiovanni-Vincentelli. Optimization-based Transistor Sizing. *IEEE Journal of Solid-State Circuits*, pages 100–409, 1988.
- [65] K. J. Singh and A. Sangiovanni-Vincentelli. A Heuristic Algorithm for the Fanout Problem. In *Proceedings of the Design Automation Conference*, pages 357–360, 1990.
- [66] K. J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Timing Optimization of Combinational Logic. In *Proceedings of the International Conference on Computer-Aided Design*, pages 282–285, 1988.
- [67] P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. Combinational Test Generation using Satisfiability. Memo. No. UCB/ERL M92/12, 1992.
- [68] T. G. Szymanski. LEADOUT: A Static Timing Analyzer for MOS Circuits. In *Proceedings of the International Conference on Computer-Aided Design*, pages 130–133. IEEE, 1986.
- [69] T. G. Szymanski. Computing Optimal Clock Schedules. In *Proceedings of the Design Automation Conference*, 1992.

- [70] T. G. Szymanski. Private communication. January, 1992.
- [71] T. G. Szymanski. Private communication. February, 1993.
- [72] T. G. Szymanski and N. V. Shenoy. Verifying Clock Schedules. In *Proceedings of the International Conference on Computer-Aided Design*, 1992.
- [73] H. Touati. *Performance-Oriented Technology mapping*. PhD thesis, University of California, Berkeley, 1990.
- [74] R. S. Tsay. Exact Zero Skew. In *Proceedings of the International Conference on Computer-Aided Design*, 1991.
- [75] S. H. Unger and C. J. Tan. Clocking Schemes for High-Speed Digital Systems. *IEEE Transactions on Computers*, C-35(10):880–895, October 1986.
- [76] P. M. Vaidya. A New Algorithm for Minimizing Convex Functions over Convex Sets. *Proceedings of the IEEE Foundations of Computer Science*, 1989.
- [77] D. Wallace and C. H. Sequin. ATV: An Abstract Timing Verifier. In *Proceedings of the Design Automation Conference*, pages 154–159. IEEE/ACM, 1988.
- [78] N. Weiner and A. Sangiovanni-Vincentelli. Timing Analysis in a Logic Synthesis Environment. In *Proceedings of the Design Automation Conference*, pages 655–661. IEEE/ACM, 1989.
- [79] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesley, 1985.
- [80] M. A. Wold. Design verification and Performance Analysis. In *Proceedings of the Design Automation Conference*, pages 264–270. IEEE/ACM, 1978.
- [81] D. Wong, G. De Micheli, and M. Flynn. Inserting Active Delay Elements to Achieve Wave Pipelining. In *Proceedings of the International Conference on Computer-Aided Design*, pages 270–273. IEEE, 1989.
- [82] R. Zahir. *Controller Synthesis for Application Specific Integrated Circuits*. Hartung–Gorre Verlag, Konstanz, Germany, 1991.

Index

- active-high, 19
- active-high level-sensitive latch, 19
- acyclic graph
 - graph, 108
- advantages of
 - synchronous circuit, 4
- aggressive
 - clocking constraints, 56
- AHLSL, see *active-high level-sensitive latch*
- arrival at a memory element
 - early, 30
 - late, 30
- arrival time, 21
- arrival time at a gate
 - early, 109
 - late, 109
- asynchronous circuit, 2
- candidate
 - edge, 90, 114
- circuit model, 16
- clock
 - clock event, 2
 - clock verification problem, 35
 - optimal clock schedule computation, 55
 - pipeline resynthesis problem, 86
 - signal, 21
- clock distribution
 - network, 133
- clock event, 21
 - clock, 2
- clock period, 2
 - lower bound, 64
- clock schedule, 2, 27
- clock skew, 3
- clock verification problem
 - clock, 35
 - multiple solutions, 36, 42
 - unique solution, 43
 - unique solutions, 36
- clocking constraints, 6, 31
 - aggressive, 56
 - conservative, 56
 - dominating, 71
 - external, 55, 134
 - internal, 134
 - model, 134
- clocking scheme, 26
 - precedence, 26
 - robust, 133
- CmM, see *consistent min-max delay*
- combinational
 - path, 133
- conservative

- clocking constraints, 56
- consistent min-max delay
 - Min-max delay, 17, 133
- constraint graph, 67, 87
 - graph, 70
- Construction C:, 112
- continuous
 - padding, 109
- covering, 112
- critical long path
 - path, 109
- critical path, 6
- critical short path
 - path, 109
- cycle, 28
 - negative weight cycle, 37
 - positive weight cycle, 37
 - relevant, 88
 - simple, 28
 - zero weight cycle, 37
- cycle stealing, 24, 94
- data
 - network, 133
 - signal, 21
- delay
 - gate, 17
 - path, 109
- delay insertion problem, 110
 - linear program, 118
- depth
 - pipeline, 5
- discrete
 - padding, 109, 121
- dominating
 - clocking constraints, 71
- drawbacks
 - synchronous circuit, 4
- dual, 142
- duty cycle, 2, 55
 - maximum, 67
 - minimum, 69
- dynamic
 - power dissipation, 4
 - timing analysis, 7, 22
- early
 - arrival at a memory element, 30
 - arrival time at a gate, 109
 - equation set, 35
- edge
 - candidate, 90, 114
 - external, 108
 - internal, 108
 - weight, 28, 108
- Edge-triggered memory element, 10, 19
- equation set
 - early, 35
 - late, 35
- external
 - clocking constraints, 55, 134
 - edge, 108
- external timing constraints, 84
- falling edge-triggered D flip-flop, 19
- false path, 7, 23
- fanin, 28, 108

- set, 28, 108
- fanout, 28, 108
 - set, 28, 108
- FEDFF, see *Falling edge-triggered D flip-flop*
- Fixed Delay, 17
- fixed point, 36
- flip-flop, 10, 19
- gate
 - delay, 17
- gradient projection, 141
- graph, 88
 - acyclic graph, 108
 - constraint graph, 70
 - latch graph, 28
 - modified, 94
- hard
 - required time, 96
- hazard, 3
- hazard-free, 3
- hold, 58, 86
- hold constraint, 20, 31
- hold time, 20
- internal
 - clocking constraints, 134
 - edge, 108
- Kuhn-Tucker conditions, 143
- Lagrange multipliers, 142
- Lagrangian, 142, 143, 144
- latch, 10, 19
- latch graph
 - graph, 28
- late
 - arrival at a memory element, 30
 - arrival time at a gate, 109
 - equation set, 35
- length
 - path, 6
- level
 - pipeline, 5
- Level-sensitive memory element, 10, 19
- library delay, 17
- linear delay, 17
- linear program, 55
 - delay insertion problem, 118
 - optimal clock schedule computation, 66
- long path, 58, 134
- long path edge, 88
- lower bound
 - clock period, 64
- mapped, 5
- matrix
 - positive definite, 92, 141
- maximum
 - duty cycle, 67
- Min-max delay, 17
 - consistent min-max delay, 17, 133
 - simplified min-max delay, 17, 132
- minimum
 - duty cycle, 69
- model
 - clocking constraints, 134
- modified

- graph, 94
- multiple solutions
 - clock verification problem, 36, 42
- negative weight cycle
 - cycle, 37
- network
 - clock distribution, 133
 - data, 133
 - skew, 133
- optimal clock schedule computation
 - clock, 55
 - linear program, 66
- padding, 110
 - continuous, 109
 - discrete, 109, 121
- path, 28, 109
 - combinational, 133
 - critical long path, 109
 - critical short path, 109
 - delay, 109
 - length, 6
 - skew, 133
 - supporting path, 44
 - types, 56
- phase shift operator, 26
- phases, 1, 21
- pipeline, 4, 85
 - depth, 5
 - level, 5
- pipeline resynthesis problem
 - clock, 86
- positive definite
 - matrix, 92, 141
- positive weight cycle
 - cycle, 37
- power dissipation, 4
 - dynamic, 4
 - quiescent, 4
- precedence
 - clocking scheme, 26
- primal, 142
- primary input, 16, 108
- primary output, 17, 108
- quiescent
 - power dissipation, 4
- relevant
 - cycle, 88
- required time, 96
 - hard, 96
 - soft, 96
- resynthesis, 90
- retardation, 24
- Retiming, 7, 25
- robust, 135, 136
 - clocking scheme, 133
- set
 - fanin, 28, 108
 - fanout, 28, 108
- set-up, 58, 86
- set-up constraint, 20, 31
- set-up time, 20
- short path, 58, 134

- short path edge, 88
- short path violation, 121
- signal, 21
 - clock, 21
 - data, 21
- simple
 - cycle, 28
- simplified min-max delay
 - Min-max delay, 17, 132
- skew
 - network, 133
 - path, 133
- skew network, 33
- slack, 25, 84, 90
- slacks, 116
- SmM, see *simplified min-max delay*
- soft
 - required time, 96
- static
 - timing analysis, 6, 22
- Statistical delay, 18
- supporting path
 - path, 44
- synchronous circuit, 1
 - advantages of, 4
 - drawbacks, 4
- Technology mapping, 5
- timing analysis, 6
 - dynamic, 7, 22
 - static, 6, 22
- types
 - path, 56
 - unique solution
 - clock verification problem, 43
 - unique solutions
 - clock verification problem, 36
 - unit delay, 17
 - unit delay fanout, 17
 - unmapped, 5
 - wave pipelining, 124
 - weight
 - edge, 28, 108
 - zero weight cycle
 - cycle, 37