Retiming with Wire Delay and Post-retiming Register Placement

Tong Ka Yau Dennis

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

June, 2004

考慮導線延遲的時序重置及其後觸發器的佈局

唐家佑
香港中文大學
計算機科學與工程學課程
哲學碩士論文
二零零四年六月

摘要

隨著互補金屬氧化物半導體(CMOS)技術進入納米尺度，集成電路(Integrated Circuits)的時頻和模子大小也平穩地增加。今天，在許多超微米的電路設計中，很多長導線(global wire)都要求多個時鐘週期去傳遞電子信號。因此，近來有不少研究嘗試用觸發器(registers)去管線輸送(pipeline)長導線，以減少長導線延遲的影響。

其中一個可行的方法是利用時序重置(retiming)的技術。在不改變電路設計的功能下，時序重置能夠通過重置觸發器的位置以優化電路的表現。可是，在傳統的時序重置中並沒有考慮導線延遲的影響，很多時序重置的演算方法都假設導線並沒有延遲電子信號的傳遞。明顯地，這樣的假設並不能反映當前導線延遲已超過邏輯門延遲(gate delay)的現象。

在這篇論文中，我們設計了二個考慮邏輯門和導線延遲的新時序重置方法。第一個方法是由這份論文的作者提出，他把問題變換成一個混雜整數線性程式(mixed-integer linear program)的一種特殊情況，而這特殊情況是可解決的。因此，第一個方法可以得出考慮邏輯門和導線延遲的時序重置的最佳答案。而第二個方法則是第一個方法的改良版本，主要是在演算速度上的改善。它可以在較短的時間內得出非常接近最佳答案的結果。這份論文的作者主要負責第二個方法的程式編寫。使用 ISCAS89 基準電路，實驗結果顯示第二個方法得出來的結果平均只比最佳答案差 0.13%。在二零零三年的一個學術會議中，當我們把有關研究成果發佈時，另外一份解決同樣問題的研究報告也同時發表。爲使這份論文更加全面，上述三種方法都包括在這論文的第四課中。

雖然考慮邏輯門和導線延遲的時序重置問題已解決，但是在時序重置後觸發器的佈局亦十分重要。在這篇論文中，我們研究如何佈置觸發器在長導線上，從而達到時序重置後得出來的目標時頻。相比以前一些作簡單計算以確定觸發器位置的研究，我們提出的算法可以保存指定的時頻並使用最少量的觸發器。此外，我們證明的了這算法能夠爲少於五個終端的電路網找出最佳答案，而這類型的電路網結構平均佔了 90%的電路網。使用 ISCAS89 基準電路，實驗結果顯示我們的算法可以在一分鐘爲大部份電路網找出最佳分享觸發器的佈局，即是使用最少量的觸發器及保存目標時頻。

Retiming with Wire Delay and Post-retiming Register Placement

Submitted by

Tong Ka Yau Dennis

for the degree of Master of Philosophy in

Computer Science and Engineering

at the Chinese University of Hong Kong

in June, 2004

Abstract

As the CMOS technology is scaled into the dimension of nanometer, the clock frequencies and die sizes of ICs increase steadily. Today, global wires that require multiple clock cycles to propagate electrical signal are prevalent in many deep sub-micron designs. Efforts have been made to pipeline the long wires by introducing registers along these global paths, trying to reduce the impact of wire delay dominance.

The technique of retiming to relocate registers in a circuit without affecting the circuit functionality can be applied in this problem. However, in the traditional formulation of retiming, wire delay was safely ignored. Most retiming algorithms have assumed ideal conditions for the non-logical portions of the data path, which are not accurate to reflect the current situations where wire delay has no less importance than gate delay.

In this thesis, we study and implement two new retiming approaches which consider both gate and wire delay such that solutions can be found in polynomial time. The first approach was proposed by the author of this thesis in which the problem was transformed into a special case of a mixed-integer linear program such that optimal solutions can be obtained. The second approach was an improvement of the first approach in terms of execution time and near-optimal solutions can be found. The author was involved in the implementation of the second approach. Using the ISCAS89 benchmark circuits, experimental results showed that the second approach can give solutions that are only 0.13% larger than the optimal on average but in a much shorter runtime as compared to the first approach. When these approaches were made public in an academic conference in 2003, another publication solving the same problem appeared at the same time. To make this thesis more comprehensive, all of the above three approaches will be included and discussed in chapter 4.

Though the problem of retiming with gate and wire delay has been solved, the placement of registers after retiming is non-trivial. In this thesis, we will also study the problem of realizing a retiming solution on a global netlist by inserting registers in the placement to achieve the target clock period. In contrast to those previous works that performed simple calculations to determine the positions of the registers, our proposed algorithm can preserve the given clock period and utilize as few registers as possible in the realization. Our proposed algorithm is shown to be optimal for nets with 4 or fewer pins and this type of nets constitutes over 90% of the nets in a sequential circuit on average. Using the ISCAS89 benchmark circuits, experimental results showed that our algorithm can find the best sharing of registers for a net in most of the cases, i.e., using the minimum number of registers while preserving the target clock period, within one minute running on an Intel Pentium IV 1.5GHz PC.

Acknowledgments

Special thanks will be given to F. Y. Young.

Contents

1 Introduction **1**

1.1 Motivations 1

1.2 Progress on the Problem 2

1.3 Our Contributions 3

1.4 Thesis Organization 4

2 Background on Retiming **5**

2.1 Introduction 5

2.2 Preliminaries 7

2.3 Retiming Problem 9

3 Literature Review on Retiming **10**

3.1 Introduction 10

3.2 The First Retiming Paper 11

3.2.1 "Retiming Synchronous Circuitry" 11

3.3 Important Extensions of the Basic Retiming Algorithm 14

3.3.1 "A Fresh Look at Retiming via Clock Skew Optimization" 14

3.3.2 "An Improved Algorithm for Minimum-Area Retiming" . 16

3.3.3 "Efficient Implementation of Retiming" 17

3.4 Retiming in Physical Design Stages 19

3.4.1 "Physical Planning with Retiming" 19

3.4.2 "Simultaneous Circuit Partitioning/Clustering with Retiming for Performance Optimization 20
3.4.3 "Performance Driven Multi-level and Multiway Partitioning with Retiming 22
3.5 Retiming with More Sophisticated Timing Models 23
3.5.1 "Retiming with Non-zero Clock Skew, Variable Register, and Interconnect Delay" 23
3.5.2 "Placement Driven Retiming with a Coupled Edge Timing Model" 24
3.6 Post-Retiming Register Placement 26
3.6.1 "Layout Driven Retiming Using the Coupled Edge Timing Model" 26
3.6.2 "Integrating Logic Retiming and Register Placement" . . 27

4 Retiming with Gate and Wire Delay [2] 29

4.1 Introduction 29
4.2 Problem Formulation 30
4.3 Optimal Approach [2] 31
4.3.1 Original Mathematical Framework for Retiming 31
4.3.2 A Modified Optimal Approach 33
4.4 Near-Optimal Fast Approach [2] 37
4.4.1 Considering Wire Delay Only 38
4.4.2 Considering Both Gate and Wire Delay 42
4.4.3 Computational Complexity 43
4.4.4 Experimental Results 44
4.5 Lin's Optimal Approach [23] 47
4.5.1 Theoretical Results 47
4.5.2 Algorithm Description 51
4.5.3 Computational Complexity 52

4.5.4 Experimental Results 52
4.6 Summary 54

5 Register Insertion in Placement [36] 55
5.1 Introduction 55
5.2 Problem Formulation 57
5.3 Placement of Registers After Retiming 60
5.3.1 Topology Finding 60
5.3.2 Register Placement 69
5.4 Experimental Results 71
5.5 Summary 74

6 Conclusion 75

Bibliography 77

List of Figures

2.1 Correlator of 24 units of propagation delay. 5
2.2 Correlator of 17 units of propagation delay. 6
2.3 Correlator of 13 units of propagation delay. 7
2.4 Retiming graph of the correlator in fig. 2.2. 8

3.1 The advantage of nonzero clock skew. 14
3.2 Minimum clock period configuration with 4 registers. 16
3.3 Minimum clock period configuration with 5 registers. 17
3.4 Relationship between partitioning and retiming. 20
3.5 Single sink net timing model. 24
3.6 An example of single register placement. 26

4.1 Register arrangement in the optimal algorithm. 35
4.2 DFS transformation of a sequential circuit retiming graph model. 38
4.3 Physical meaning of the timing variable t_v. 39
4.4 Vertex splitting to handle gate delay. 42
4.5 Removing register on a wire representing a gate. 43

5.1 An illustration of the definition of $a(v)$ for a gate v. 56
5.2 Graph model of a 4-pin net in which each edge has a single register after retiming. 58

5.3 Case (a) - a single register is shared among the edges (maximum sharing). The topology tree of this configuration is shown on the right. 58
5.4 Case (b) - each edge has a separate register (no sharing). The topology tree of this configuration is shown on the right. 59
5.5 Case (c) - two of the edges share a register while the other has a separate one. The topology tree of this configuration is shown on the right. 59
5.6 A situation in which the registers cannot be shared in order to preserve the clock period $clk = 1.5$ units. 60
5.7 The retiming graph model (left) and the corresponding best possible topology $\Upsilon_{N_{opt}}$ (right) of a 4-pin net example. 61
5.8 The topology tree of a 3-pin net Υ_N where r_1 and r_2 are two shared registers. 70
5.9 An illustration of how the final position of register r_2 is determined. 71

List of Tables

4.1 Benchmark statistics for retiming. 45
4.2 Runtime of the algorithms and clock periods obtained. 46
4.3 Runtime of Lin's approach and the optimal clock periods obtained. 53

5.1 Benchmark statistics after retiming. 72
5.2 Results of register placement with clock preservation. 73

Chapter 1

Introduction

1.1 Motivations

As the CMOS technology continues to scale down, the clock frequencies and die sizes of integrated circuits increase steadily. Data showed that the frequencies of high-performance ICs have doubled every process generation while the die sizes increased by about 25% [11]. With such short cycles and long interconnects, it is common for a global signal to take multiple clock periods to travel across a chip. To alleviate this problem, we need to insert registers to pipeline long global wire [3, 16]. Nevertheless, arbitrary insertion of registers on a wire is forbidden because the original functionality of the circuit will be changed. As a result, retiming, a sequential circuit optimization technique that relocates registers without affecting circuit functionality, can be applied [20, 22].

Retiming is such a powerful circuit optimization technique that it can be used to minimize the clock period, the usage of registers, or a combination of both objectives of a sequential circuit. Since retiming was first formulated a decade ago, much effort has been made to improve various aspects of the algorithm. However, in the traditional settings of retiming, only gate delay was considered and wire delay was ignored. As process technology gets down to deep sub-micron, wire delay becomes a major factor of path delay which cannot be overlooked in today's circuit designs. Therefore, there is a strong

need to formulate and solve a new retiming problem such that both gate and wire delay are considered.

Besides the problem of retiming with gate and wire delay, the placement of registers after retiming is another new challenge. Since a net is represented as a set of separated edges in a retiming graph model which does not bear any information about the topology of the routes or the positions of the registers, it is unknown whether the clock period obtained from retiming can be realized in the design.

Being able to insert registers into a placement solution in order to realize a retiming solution and preserve the corresponding clock period is important, or it will make the retiming optimization meaningless. Meanwhile, minimizing the number of registers used is also essential as the size of a register is usually several times larger than that of a simple gate, regardless of the process technology being used. Even though there are a few previous works that touch briefly on the problem of post-retiming register placement, most of them suffer from the problem of over-simplification when wire delay dominates. A sophisticated method to tackle this problem is of utmost need.

1.2 Progress on the Problem

There are several previous works addressing the interconnect issues in retiming. In the papers [33, 18], the authors tried to incorporate wire delay into the retiming process but assumed that the wire delay between adjacent registers on the same wire was negligible. Another approach to integrate retiming into detailed placement was proposed in [35] where heuristics were used to estimate wire delay after initial placement and routing, retiming was then applied to optimize the circuit performance. In the paper [34], Tabbara et al. applied retiming in the DSM domain and wire delay was considered. However, all these works are constrained by overly simplified assumptions for practical uses.

When it comes to the placement of registers after retiming, there are a few works that address the problem directly. For those works that have touched on the topic of post-retiming register placement, they usually suffer from the problem of over-simplification when wire delay dominates. For example, in the paper [35], the authors assume that the position of a register is located at the geometric center of the connected gates. This assumption is natural, but the clock period resulted from retiming will be easily violated. A similar problem occurs in [26] in which the authors determine the position of a register such that the sum of the lengths of the nets connected to that register is minimized.

1.3 Our Contributions

To address the problem of retiming with gate and wire delay, we have studied two approaches to tackle the problem [2]. The first approach solves the problem optimally while the second one gives solutions very close to the optimal (0.13% more than the optimal on average) but in a much shorter runtime. To the best of our knowledge, this is a pioneer work in solving this problem.

Inspired by the MILP approach discussed in [22], we have proposed an approach that solves the problem optimally in polynomial time. In the MILP framework of [22], the authors do not take wire delay into account and model the problem of retiming as solving a set of four general constraints with both integer and real variables at the same time. Since the set of constraints is a special case of the MILP problem, they can be solved in polynomial time [21]. However, the original MILP framework is not adequate to capture the timing requirements with wire delay. To handle both gate and wire delay simultaneously, we have introduced some new variables and modified one of the four general constraints such that wire delay is incorporated into the formulation. Most importantly, our proposed modifications remain to be a special case of the MILP problem, thus it is still solvable in polynomial time.

The near-optimal approach is an improvement over the optimal approach in terms of execution runtime as it involves fewer number of variables and constraints. In fact, this approach can give optimal solutions if we consider wire delay alone. Experimental results showed that this approach gives very good solutions using only a fraction of the time required in the optimal approach. My main contributions to this near-optimal approach are in the implementation of the algorithm and in conducting the relevant experiments.

Apart from the problem of retiming with gate and wire delay, the problem of post-retiming register insertion in placement worth no less attention. In this thesis, we have proposed an algorithm to realize a retiming solution in a placement such that the target clock period is preserved with a controlled error, using as few registers as possible [36]. In contrast to those previous works, we do not have the problem of clock violation under our linear wire delay model. In addition, our algorithm can give optimal solutions for nets with 4 or fewer pins and this type of nets makes up about 90% of the nets in a sequential circuit on average. Experimental results showed that our algorithm can find the best solution for each net in most of the cases, making the retiming optimization process more applicable in today's deep sub-micron design for achieving a higher performance.

1.4 Thesis Organization

This thesis is organized as follows. We will give an introduction in chapter 1. There will be an overview on retiming in chapter 2. Literature review on retiming and physical design will be discussed in chapter 3. In chapter 4, the problem of retiming with gate and wire delay will be discussed, our two proposed approaches and Lin's approach [23] will be described in detail. In chapter 5, our algorithm for post-retiming register placement will be presented. Finally, a conclusion will follow in chapter 6.

Chapter 2

Background on Retiming

2.1 Introduction

Retiming is a technique for optimizing sequential circuits by repositioning the registers in the circuit while maintaining its original functionality. This technique was first formulated by Leiserson and Saxe in [20], and a completed version of their work [22] appeared in 1991 in which they proposed several algorithms in solving the min-period, min-area and constrained min-area problems under the framework of retiming.

An example of a correlator in [22] is described in fig. 2.1. The correlator consists of two functional elements, adders (+) and comparators (#). The black strips on the wires between two functional elements are registers.

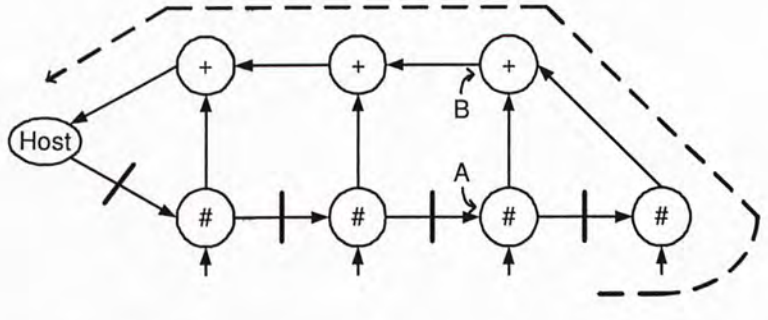

Figure 2.1: Correlator of 24 units of propagation delay.

Assuming that all registers are clocked by a single clock signal. Suppose the propagation delay of the adder and the comparator are of 7 and 3 units respectively, the longest combinational path in the correlator in fig. 2.1 has 24 units of delay as shown by the dotted line.

A design that gives better performance of the correlator is shown in fig. 2.2. This design is obtained by moving a register of the original circuit in fig. 2.1 from point A to B. To show that both correlators are functionally equivalent, let us consider the subcircuit inside the dashed box. This subcircuit communicates with the rest of the circuit through connection A and B only. Since the register at A is removed, all input signals to this box of circuit arrive one clock tick earlier, thus this subcircuit performs the same sequence of computations as the subcircuit in the previous correlator with one clock tick earlier.

On the other hand, the register is moved to position B, which is the output of the dashed box, and thus the output is delayed by one clock tick. From the perspective of the remainder of the circuit, this subcircuit behaves the same as in the previous correlator. The longest combinational delay of this *retimed* correlator is 17 units as shown by the dashed line.

In fact, the correlator can be further optimized to an optimal clock period of 13 units of delay as shown in fig. 2.3. As we can see, by simply moving the registers in a circuit, the circuit performance could be greatly improved.

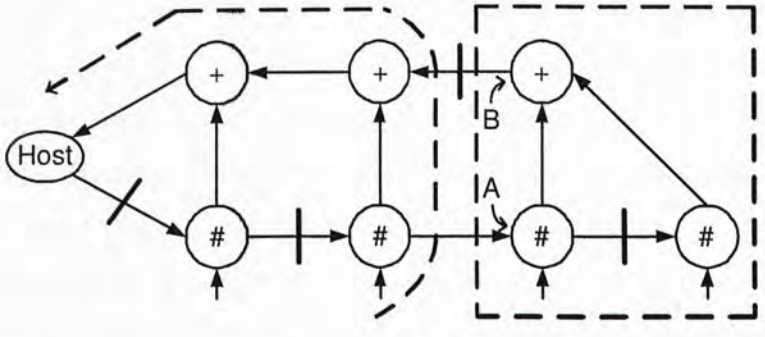

Figure 2.2: Correlator of 17 units of propagation delay.

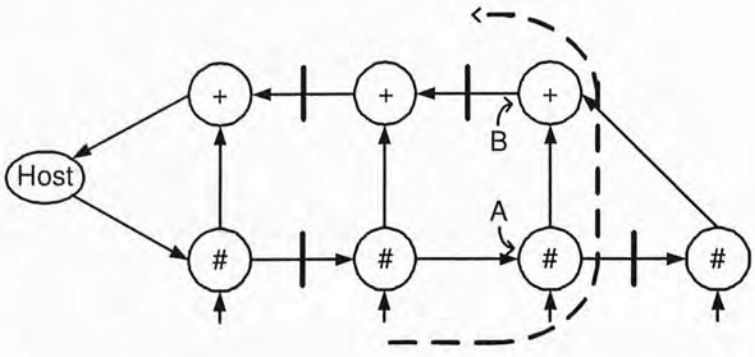

Figure 2.3: Correlator of 13 units of propagation delay.

2.2 Preliminaries

To apply the technique of retiming, we need to represent a sequential circuit as a graph-theoretical model and define some notations and definitions.

In the subsequent discussion of retiming, we assume that a sequential circuit can be viewed as a network of functional elements and globally clocked registers. A register is a storage element which has a single input and a single output; and all registers are clocked by the same periodic waveform. At each clock tick, the data at the input of a register is sampled and stored at the output. Besides, we assume that there is no race condition, i.e., the changes in the output of one register do not interfere with the input of another at the same clock tick. This is made possible by the use of a typical D-type flip-flop.

A sequential circuit is modeled as a finite, vertex-weighted, edge-weighted, directed multigraph $G =< V, E, d_v, w(u,v) >$, where V is the set of vertices in G, representing the functional elements, E is the set of edges in G, representing the interconnections, d_v is a numerical weight associated with every vertex v, representing the propagation delay of that functional element, and $w(u,v)$ is a weight associated with the edge e_{uv}, representing the number of registers on that wire. Taking the correlator in fig. 2.2 as an example, its multigraph representation G is shown in fig. 2.4.

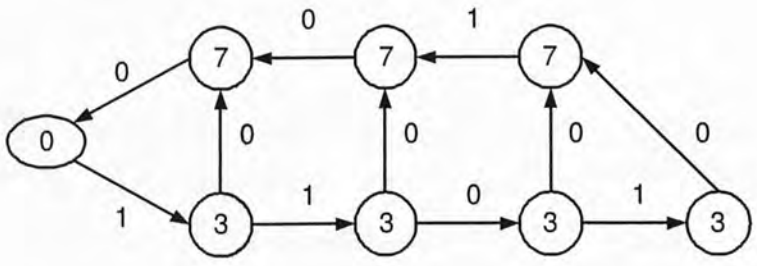

Figure 2.4: Retiming graph of the correlator in fig. 2.2.

Given a multigraph G, a path p in G can be viewed as a sequence of vertices and edges. If a path p starts at a vertex u and ends at a vertex v, we denote it as $u \xrightarrow{p} v$. A simple path contains no cycle, and therefore the number of vertices exceeds the number of edges by exactly one. For any path $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$, the path weight of p, $w(p)$, is defined as the sum of the weights of the edges on the path:

$$w(p) = \sum_{i=0}^{k-1} w(e_i).$$

Similarly, for any simple path $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$, the path delay of p is defined as the sum of the delays of the vertices on the path:

$$d(p) = \sum_{i=0}^{k} d_{v_i}.$$

In order to make the graph G have a well-defined physical meaning as a sequential circuit, nonnegativity constraints are placed on the propagation delays d_v and the register counts $w(e)$ as follows:

1. The propagation delay d_v is nonnegative for each vertex $v \in V$.

2. The register count $w(e)$ is a nonnegative integer for each edge $e \in E$.

In order to avoid race condition, the following restriction is applied such that there is no directed cycle with zero weight:

3. In any directed cycle of G, there is at least one edge with positive register count.

If a circuit satisfies the above three constraints, it is a legal synchronous circuit. For any synchronous circuit G, the clock period $\Phi(G)$ is defined as the propagation delay of the longest combinational path. Using the definitions of path delay and path weight, the clock period of a circuit G can be defined as follows:

$$\Phi(G) = max_{w(p)=0}\{d(p)\}.$$

2.3 Retiming Problem

Given the retiming graph of a sequential circuit, retiming can be viewed as a vertex-to-integer mapping, $r : V \longrightarrow \mathbf{Z}$ where $\mathbf{Z}$ is the set of integers such that a new retimed graph $G_r =< V, E, d_v, w_r(u,v) >$ is obtained by moving the registers in G as follows:

$$w_r(u,v) = w(u,v) + r(v) - r(u), \qquad \forall e_{uv} \in E \tag{2.1}$$

and all $w_r(u,v)$ found must be non-negative (it is known as a *legal* retiming). In general, retiming is performed to minimize the minimum clock period or the total register count.

Chapter 3

Literature Review on Retiming

3.1 Introduction

In order to meet the performance requirements of today's complex designs, especially in the current deep sub-micron (DSM) technology era, the possibility of putting various circuit optimization techniques into practical uses are being explored by researchers. Retiming, a well-known sequential circuit optimization technique, is of no exception. In this chapter, the first retiming paper and four major aspects of retiming are reviewed.

First of all, the original retiming paper authored by Leiserson and Saxe [22] will be presented. Secondly, various improvements to the original retiming algorithms are discussed which improve the usefulness of retiming on larger circuits. Next, the applications of retiming on different physical design stages will be described, showing the feasibility of integrating retiming early in the physical design cycle. Besides, retiming with different wire delay models are discussed. Since wire delay dominates gate delay in today's DSM design, retiming with wire delay is of utmost importance. This leads to our study of retiming with gate and wire delay which will be discussed in chapter 4. Finally, retiming with register placement will be discussed, illustrating how registers are handled in placement after retiming. Again, this leads to our work on post-retiming register placement which will be presented in chapter 5.

3.2 The First Retiming Paper

3.2.1 "Retiming Synchronous Circuitry"

The technique of retiming was first formulated by Leiserson and Saxe in [20, 22] more than a decade ago. In this paper [22], they gave a comprehensive analysis on retiming and proposed several approaches to solve the problem efficiently. Besides, various possible applications of retiming were discussed. In this section, we will focus on their algorithm OPT1 for retiming that forms the basic framework of many improvements appeared later in the literature.

Specifically, the authors formulated the clock-period-minimization problem as: *Given a circuit graph* $G =< V, E, d_v, w(u,v) >$, *find a legal retiming* r *of* G *such that the clock period* $\Phi(G_r)$ *of the retimed circuit* G_r *is as small as possible.* The design of their first algorithm OPT1 depends on the fact that the following linear-programming problem can be solved efficiently using the Bellman-Ford algorithm [8].

PROBLEM LP. *Let* S *be a set of* m *linear inequalities of the form:* $x_j - x_i \leq a_{ij}$ *on the unknowns* $x_1, x_2, \ldots, x_n$, *where the* a_{ij} *are given real constants. Determine feasible values for the unknowns* x_i, *or determine that no such values exist.*

The algorithm OPT1 is based on an alternative characterization of the clock period in terms of two quantities that are defined as: $W(u,v) = min\{w(p) : u \xrightarrow{p} v\}$ and $D(u,v) = max\{d(p) : u \xrightarrow{p} v \text{ and } w(p) = W(u,v)\}$. The quantity $W(u,v)$ is the minimum number of registers on any path from u to v. We call a path $u \xrightarrow{p} v$ such that $w(p) = W(u,v)$ a critical path from u to v. The quantities $D(u,v)$ is the maximum propagation delay on a critical path from u to v. Both quantities are undefined if there is no path from u to v.

After defining the quantities $W(u,v)$ and $D(u,v)$, the authors observed

that a feasible clock period c has a relationship with the quantities as stated below:

Lemma 4 [22] Let $G =< V, E, d_v, w(u,v) >$ be a synchronous circuit, and let c be any positive real number. The followings are equivalent:

4.1. $\Phi(G) \leq c$.

4.2. *For all vertices u and v in V, if $D(u,v) > c$, $W(u,v) \geq 1$.*

To calculate W and D, the authors pointed out that it is similar to solving an all-pair shortest-path problem on G. Common techniques such as Floyd-Warshall and Johnson's algorithm [8] can be used. The reason that W and D are so important because they behave nicely under retiming as shown by Lemma 5 below:

Lemma 5 [22] Let $G =< V, E, d_v, w(u,v) >$ be a synchronous circuit, and let W and D be defined on G. Let r be a legal retiming of G, and let W_r and D_r be defined analogously on G_r. Then

5.1. *A path p is a critical path in G_r if and only if it is a critical path in G.*

5.2. $W_r(u,v) = W(u,v) + r(v) - r(u)$ *for all connected vertices u, v in V.*

5.3. $D_r(u,v) = D(u,v)$ *for all connected vertices u, v in V.*

It is easy to see that condition 5.1 holds as retiming changes the weights of all paths from u to v by the same amount, and then 5.2 follows immediately. Condition 5.3 holds as retiming does not change the propagation delays and as a consequence of condition 5.2. This lemma leads to an important fact that if r is a legal retiming on G, the minimum clock period $\Phi(G_r)$ is equal to $D(u,v)$ for some u, $v \in V$. That is, the set of $D(u,v)$ where $u, v \in V$ includes all the possible clock periods of a legal retiming on the circuit G and binary search can then be applied on those finite number of values to find the optimal clock.

Finally, the authors used the following theorem to characterize the conditions under which a retiming process produces a circuit whose clock period is no greater than a given constant.

Theorem 7 [22] Let $G =< V, E, d_v, w(u,v) >$ be a synchronous circuit, let c be an arbitrary positive real number, and let r be a function from V to the integers, r is a legal retiming of G such that $\Phi(G_r) \leq c$ if and only if

7.1. $r(u) - r(v) \leq w(u,v)$ *for every edge* $u \xrightarrow{e} v$ *of* G.

7.2. $r(u) - r(v) \leq W(u,v) - 1$ *for all vertices* $u, v \in V$ *such that* $D(u,v) > c$.

According to Theorem 7, the authors proposed the following algorithm, OPT1, to solve the clock-period-minimization problem. Notice that the constraints 7.1 and 7.2 on unknowns $r(v)$ and $r(u)$ are linear inequalities involving only differences of unknowns, and thus the Bellman-Ford algorithm can be used to test the feasibility of a clock period c.

Algorithm OPT1. Given a synchronous circuit $G =< V, E, d_v, w(u,v) >$, this algorithm determines a retiming r such that $\Phi(G_r)$ is as small as possible.

1. Calculate W and D.
2. Sort the elements in D.
3. Binary search among D for the minimum achievable clock period. To test the feasibility of a clock c, use the Bellman-Ford algorithm on the conditions in Theorem 7.
4. For the minimum achievable clock period found in step 3, use the values of the $r(v)$ found by the Bellman-Ford algorithm as the optimal retiming solution.

Algorithm OPT1 is the first known algorithm to solve the problem of retiming optimally in polynomial time. The authors of [22] have also proposed and studied other efficient approaches to tackle the problem.

3.3 Important Extensions of the Basic Retiming Algorithm

3.3.1 "A Fresh Look at Retiming via Clock Skew Optimization"

In this paper [12], clock skew minimization techniques were used in finding a retiming solution so as to perform logic optimization.

While clock skew minimization is a continuous optimization of combinational logic, retiming is a discrete optimization with the same effect. Due to the differences in wire delays on the clock distribution network, clock signals arrive at the registers at different times. As a result, clock skews exist between different registers. However, clock skews can be viewed as a resource to improve the performance of a circuit as illustrated in fig. 3.1.

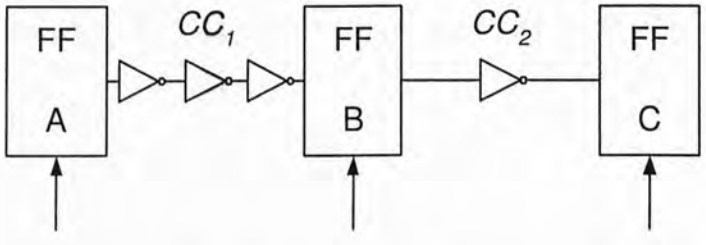

Figure 3.1: The advantage of nonzero clock skew.

Suppose each inverter has unit delay. The circuit block CC_1 and CC_2 would have delays of 3.0 and 1.0 units respectively. However, if a skew of +1.0 unit is applied to the clock line to register B (FF B), the circuit can run with a clock period of 2 units.

In the above example, if we apply the technique of retiming, we can move FF B left to cross one inverter. In this way, the circuit can also run with a clock period of 2 units after retiming. This simple example shows how clock

skew minimization and retiming have correlations with each other in terms of delay optimization.

In fact, it was proved that the effect of applying a positive clock skew on a register is equivalent to moving it from the outputs of a gate to the inputs. Similarly, the effect of a negative clock skew on a register is equivalent to moving it from the inputs of a gate to the outputs.

In this paper, an algorithm called ASTRA was proposed to solve the problem of finding a retiming solution using a clock skew minimization technique, and it guaranteed that the clock period found would be no worse than the optimal by more than one gate delay. The two-phase ASTRA algorithm is outlined below.

Phase A:
The optimal value of the skew at each register is solved, with the objective of minimizing the clock period, or to satisfy a given feasible clock period. The Bellman-Ford algorithm is performed on a constraint graph to solve this problem.

Phase B:
The skew solution is translated to a retiming solution by relocating the flip-flops across the logic gates in an attempt to set the values of all skews to zero. Any skew that cannot be set to zero exactly, is forced to zero. This would cause the clock period to increase from the optimal.

Experimental results showed that ASTRA could retime all ISCAS89 circuits in a few minutes. This paper used a novel approach to solve the problem of retiming via clock skew minimization technique which was optimally solved and well-studied.

3.3.2 "An Improved Algorithm for Minimum-Area Retiming"

In this paper [24], the authors further combined their ASTRA algorithm with the original retiming algorithm [22] to obtain an improved algorithm for minimum-area retiming, called Minret.

The original minimum-area retiming algorithm was in fact a linear programming problem subjected to two conditions: *non-negative weight for all edges (non-negative number of registers)* and *the existence of at least one register on every combinational cycle.* The authors observed that there were a lot of redundancy in the latter constraints, and thus proposed the idea of *restricted mobility of register* to reduce the number of constraints in the linear program. As a result, the experimental runtime improved significantly as compared to the ASTRA algorithm.

Using the same example shown in [24], the idea of restricted mobility of register is illustrated in fig. 3.2 and fig. 3.3. Assuming unit gate delays, the minimum clock period is 4.0 for both circuits. While the circuit in fig. 3.2 requires 4 registers, the circuit in fig. 3.3 requires 5 registers. As we can see, registers cannot be placed at just any location in the circuit in order to achieve the minimum clock period. Instead of an unrestricted movement of registers, there is only a restricted range of locations into which each register may be placed, i.e., the mobility of a register.

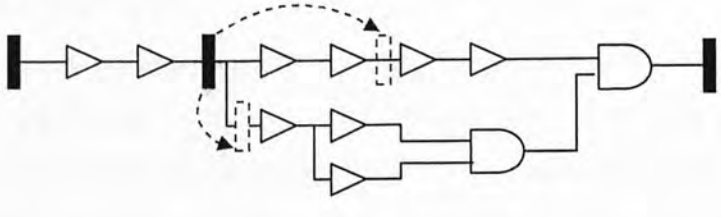

Figure 3.2: Minimum clock period configuration with 4 registers.

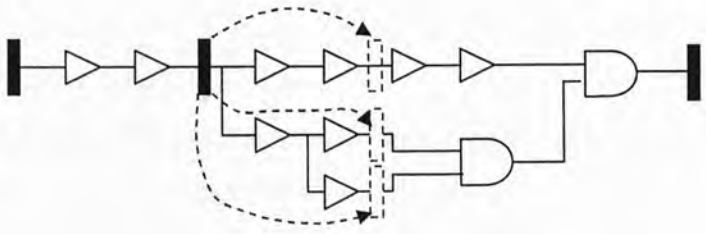

Figure 3.3: Minimum clock period configuration with 5 registers.

As shown in the example, the retiming label, $r(v)$, for each gate has a restricted value given the restricted mobility of each register. The clock skew minimization technique can be used to find the range of restricted locations for each register using the Bellman-Ford algorithm, and these information can then be used in the original Leiserson-Saxe minimum-area retiming algorithm to reduce the number of constraints in the linear programming approach.

Experimental results showed that Minret outperformed either ASTRA or the original algorithm proposed by Leiserson-Saxe in reducing the number of registers and runtime.

3.3.3 "Efficient Implementation of Retiming"

In this paper [32], an efficient implementation of retiming was proposed such that the runtime was 100X faster than a straight-forward implementation of the original algorithm by Leiserson-Saxe for some of the largest circuits, e.g., s38584 in ISCAS89 suite.

In solving the problem of minimum clock period retiming, Leiserson-Saxe gave three algorithms to tackle the problem in [22]. In the most efficient algorithm, a relaxation algorithm (the Bellman-Ford algorithm) was employed. To determine if a clock c is a feasible clock to yield a valid retiming solution, the Bellman-Ford algorithm will iterate $|V| - 1$ times to prove the feasibility

of c. If there is a negative cycle in the graph after all the $|V|-1$ iterations, the shortest path to any vertex on that cycle from the source will be undefined, and it means that no solution exists.

The authors showed that if there was any hope of speeding up the relaxation algorithm, focus must be put on detecting if a clock period was infeasible before completing the requisite $|V|-1$ iterations. Therefore, they proposed a heuristic to abort the iteration at any point if a negative cycle was detected.

The predecessor heuristic was to maintain a predecessor vertex pointer, denoted by *pred*(), for each vertex. All *pred*() pointers are set to empty before the Bellman-Ford algorithm starts. Every time the distance to a vertex u is decreased (relaxed), the fanin node that caused the change is saved as *pred*(). As a result, each vertex v is associated with a predecessor graph at every instant of the iteration that is defined by traversing the *pred*() pointers. Starting at v and ending when either the predecessor pointer is empty or a cycle is found, we check whether a cycle is found. If a cycle is found, it means that the clock is infeasible.

The proposed heuristic is elegant and easy to understand. Also, it was the first paper to report the retiming results for large circuits of over 10,000 logic gates.

3.4 Retiming in Physical Design Stages

3.4.1 "Physical Planning with Retiming"

In this paper [6], a unified approach to partitioning, floorplanning, and retiming for performance optimization was proposed. This approach allows the partitioner to exploit the underlying geometric delay model given by the floorplanner. Besides, simultaneous consideration of partitioning and retiming leads to an effective reduction in global interconnect delay.

Given a sequential circuit, let C denote the cells consisting of gates and registers, and N denotes the netlist. The authors formulated a new partitioning problem for their unified approach, named Geometric Embedding based Performance Driven K-way Partitioning (GEPDP). To solve this problem means to assign the cells in K blocks while the locations of the blocks are determined under prescribed area constraints (L_i, U_i), where L_i and U_i denote the lower and upper bound of the size of a block B_i respectively. The primary objective is to minimize delay, and the secondary objective is to minimize cutsize and wirelength. Retiming is used as a timing analysis engine to guide the partitioning process. In their geometric delay model, the delay of a path includes both gate and wire delay. Wire delay is defined to be linearly proportional to the Manhattan distance between two connected cells.

Using the above problem formulation, the authors proposed an algorithm called GEO to solve it. There are two main phases in GEO: construction phase and FF placement phase. In the construction phase, multi-level partitioning with floorplanning is performed, and in the FF placement phase, retiming algorithm is used to perform a coarse FF placement followed by a fine-grained placement on wire.

Experimental results showed that their GEO algorithm performed better than the traditional physical design flow in general, showing a benefit of unifying different physical designing steps in today's complex systems.

3.4.2 "Simultaneous Circuit Partitioning/Clustering with Retiming for Performance Optimization

In this paper [5], the authors proposed a method to incorporate retiming into the problem of partitioning and clustering so as to improve circuit performance.

The clustering problem is to decompose a given circuit into a number of clusters such that their sizes are bounded by a given number. In order to optimize the performance, the authors redefined the problem of clustering as: *Given a sequential circuit, construct a clustered circuit with the minimum clock period under retiming with possible node replication. The area of each cluster is bounded by a given number A.* This formulation allows retiming to be integrated into circuit clustering. To see how retiming relates to partitioning, consider fig. 3.4 below.

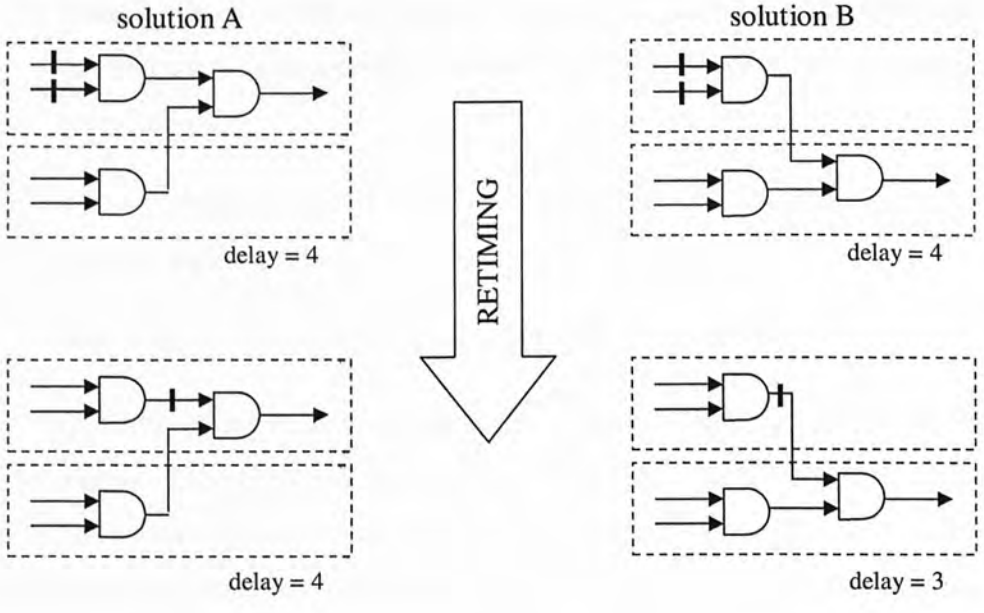

Figure 3.4: Relationship between partitioning and retiming.

In the two partitioning solutions shown in fig. 3.4 (both cutsizes are 1),

retiming applied to solution A (left) does not provide any optimization of clock period. On the contrary, retiming on solution B (right) reduces the delay from 4 to 3, assuming unit gate delay and that inter-block delay is 2 units.

Given a sequential circuit C and a target clock period clk, a label, $l_C(v)$, is computed for each node v in C that denotes the maximum path delay from primary inputs to v in C. It is shown that if the labels of the primary outputs in C are larger than the target clock period clk, a clustered circuit of C that has clock period clk or less is impossible. Otherwise, C can be clustered and *then* retimed to achieve a clock period of less than $clk + D$, where D is the interconnect delay between a pair of clusters.

Based on the node labels computed, a clustered circuit can be formed with clock period $clk + D$ guaranteed. To solve the problem, there are three main steps in their algorithm as shown below:

1. Binary search to obtain the minimum clock period clk_{min} with label computation. Note that the labels of all primary outputs must be smaller than clk_{min},

2. Form a clustered circuit based on the node labels corresponding to the minimum clock period,

3. Retiming to achieve the best clock period.

Experimental results showed that the proposed algorithm improved the clock period of the clustered circuit compared to most traditional partitioners in which cutsize minimization was the only primary goal. Besides, various optimization techniques on the label computation step were discussed, making their algorithm's runtime much shorter than most of the previous algorithms.

3.4.3 "Performance Driven Multi-level and Multiway Partitioning with Retiming

The authors of [5] extended their work further in this paper [7]. In this work, the authors considered the significant difference between local and global wire delay induced by partitioning. They proposed an algorithm called HPM that considers cutsize, delay minimization and retiming simultaneously.

As wire delay is becoming more and more important, there are two major kinds of performance-driven partitioning methods in common use: bottom-up clustering and top-down partitioning. In the bottom-up clustering, logic gates are grouped into clusters such that the area of each cluster is bounded and the delay of the circuit is minimized. However, the bottom-up clustering approach suffers from two main limitations: it has worse cutsize compared to traditional partitioners and it is difficult to control area balance between blocks.

In top-down partitioning, the circuit is divided into a pre-determined number of partitions. The size of each partition is maintained within a user specified range. Again, the primary goal is to minimize the delay of the circuit in which retiming and logic replication techniques are considered. However, these algorithms have long runtimes and there is no guarantee of optimality.

The proposed HPM algorithm consists of two phases: clustering phase and refinement phase. In the clustering phase, an initial cluster structure with consideration of subsequent retiming is built, and then a cutsize-driven clustering algorithm is used to add more levels to the hierarchy. In the refinement phase, a partitioning algorithm that considers both cutsize and delay simultaneously is used to decompose the clusters at each level.

Experimental results showed that not only did HPM minimize the clock period of a circuit greatly compared to traditional partitioners, it also gave comparable cutsize results which were unachievable in most bottom-up clustering algorithms.

3.5 Retiming with More Sophisticated Timing Models

3.5.1 "Retiming with Non-zero Clock Skew, Variable Register, and Interconnect Delay"

In this paper [33], a retiming algorithm is proposed that considers variable register, wire delay, and non-zero clock skew. These delay components are integrated into retiming by assigning a set of values, called *Register Electrical Characteristics (REC)*, to each edge in the retiming graph model of the sequential circuit. Their proposed timing model is more realistic than the one proposed in the original retiming algorithm that considers gate delay alone [22].

Since electrical information is captured by the REC of each edge, a path between logic elements is defined as a traversal from a weighted edge to another weighted edge rather than a traversal from a weighted vertex to another weighted vertex. An edge, being interpreted as a connection between logic elements, contains zero, one, or more registers. The REC of an edge in the graph has the form: T_{CD} : T_{Setup} / T_{C-Q} | T_{Int1} / T_{Int2}, where T_{CD} is the clock delay from the global clock source to each register, T_{Setup} is the required time for the data at the input of a register to latch, T_{C-Q} is the required time for the data to appear at the output of the register upon arrival of the clock signal, T_{Int1} and T_{Int2} are the wire delays along that edge if the edge has one or more registers.

After the calculation of the RECs for each edge, the authors defined a matrix, called *Sequential Adjacency Matrix (SAM)*, to capture the delay between each pair of edge-to-edge path in the retiming graph model. This matrix is similar to the one in [22] that captures the path delay between vertex-to-vertex connections. Next, a set of timing constraints is derived from SAM that considers clock skew, wire delay and valid retiming labels. If the set of constraints

is satisfiable, a valid retiming solution can be obtained.

An iterative, branch-and-bound method using ranges of vertex labels rather than constant vertex labels are proposed to solve for the edge weights (i.e., the number of registers on each edge). Experimental results showed that the proposed algorithm could perform a more accurate and generalized retiming than existing algorithms that did not employ a more sophisticated delay model.

3.5.2 "Placement Driven Retiming with a Coupled Edge Timing Model"

In this paper [25], a retiming algorithm using a highly accurate timing model is proposed. The timing model considers the effect of retiming on the capacitive loads of a single wire as well as the fanout tree of a node. Since registers retimed into a fanout net may change the topology of the net significantly, the arrival times at the neighboring branches would also be affected. Here, we focus on their proposed timing model of a net such that all the above effects are taken into accounts.

To handle the changes in the capacitive loads of a net, the author classified the nets into two categories: *single sink net* and *multiple-sink net*, and proposed a timing model for each type of nets. For a single sink net, edge $e = (u, v)$, three delay values are assigned as shown in fig. 3.5.

Figure 3.5: Single sink net timing model.

t_w denotes the delay for a signal traveling from u to v when there is no register (i.e., $w_r(u, v) = 0$), t_i denotes the delay for a signal traveling from u to the input of a register on e when there is at least one register on e (i.e.,

$w_r(u, v) > 0$). Similarly, t_o denotes the delay for a signal traveling from the output of a register on e to v.

For a multiple-sink net, the author suggested two models: *complex model* and *simple model.* The former is more accurate but suffers from the problem of great complexity when the fanout net is large, the latter is less accurate but useful for modeling larger nets. In the complex model, it distinguishes for each branch b in a fanout tree B whether $w_r(b) = 0$ or $w_r(b) > 0$. Therefore, there are 2^n different cases for a fanout tree with n branches. Since the presence of a register on a branch may affect the loads of the other branches, the delay values (t_w, t_i and t_o) of each branch will behave differently in each of the 2^n configurations. To accurately record the effects of each case, a table with 2^n entries is associated with each branch during retiming. Obviously, it has the problem of exponential table growth.

When it comes to the simple model, the author assumed that the delay values of a particular branch $b_i \in B$ are the same for all configurations containing the same number of branches b_j with registers, i.e., given the same number of registers on other branches, the capacitive loads of b_i does not depend on which particular branches have registers. By doing this simplification, the size of a table will only grow linearly at the expense of less accurate delay model. However, this inadequacy diminishes as the number of branches in a net increases.

To achieve the best performance, the author suggested the use of the complex model for nets with four or fewer sinks and the simple model for the rest of the nets. Since nets with four or fewer sinks constitute over 90% of the nets in a sequential circuit on average, the complex models are often used which offer precise delay estimations for most of the nets. Experimental results showed that their retiming algorithm, using their proposed timing models, gave higher accuracy and tighter integration of placement and retiming as compared to previous works.

3.6 Post-Retiming Register Placement

3.6.1 "Layout Driven Retiming Using the Coupled Edge Timing Model"

In this paper [26], the authors proposed a new approach for making retiming more practical. There are two main parts in their approach: a new coupled edge timing model and a new retiming algorithm. Here, we focus on their suggested post-retiming register placement method.

First of all, maximum sharing of registers among the fanout edges of a node is assumed, i.e., the minimum number of registers is utilized for every net. The author argued that since a register was several times larger than a simple gate, it was preferable to use fewer registers in the actual layout. Next, for each new register, a position is determined such that the sum of the lengths of the nets connected to this register is minimized. An example is shown in fig. 3.6.

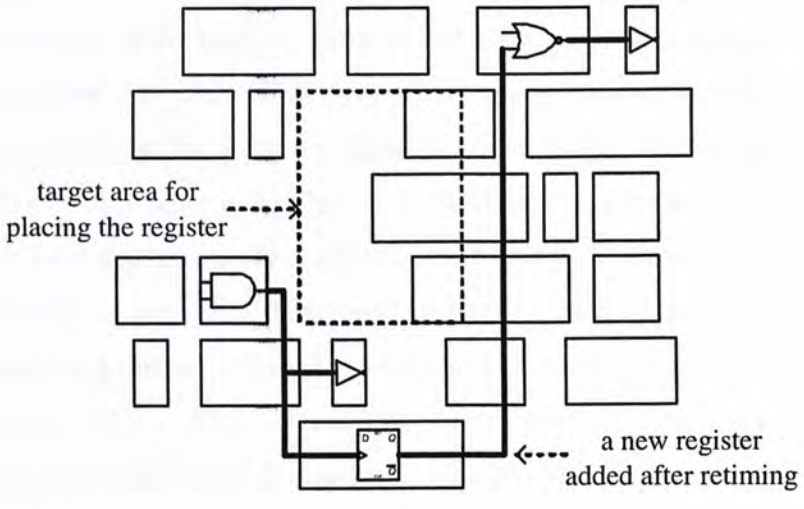

Figure 3.6: An example of single register placement.

In most of the cases, the feasible location for a register is a rectangular area, but not a particular point. After yielding a rectangular region for placing a

register, the algorithm looks for the most suitable cell gap inside this area and places the register there.

If the gap does not have enough free space, neighboring cells are pushed aside until enough room is made available to fit in the new register; however, no cell overlapping will occur. For those areas that are empty after the removal of a register, no work will be done to reclaim the area as the simulated annealing placer will reuse the spaces at the subsequent iterations.

3.6.2 "Integrating Logic Retiming and Register Placement"

In this paper [35], the author proposed a post-layout retiming technique that consists of heuristics for wire delay estimation, retiming incorporating wire delay, and post-retiming placement. Again, we focus on its post-retiming placement scheme here.

Since it is hard to alter the positions of registers and gates simultaneously, the authors assumed that all gates would remain at their original locations during register placement. Having made this assumption, they naturally predicted the location for an added register as the closest slot to the geometric center of its connecting gates. However, this simple assignment lacks the flexibility of finding other feasible locations of a register. They studied the layouts of many circuits and found that most registers were not necessarily placed at the estimated locations. Only the register that started or ended a path with delay close to the cycle time should be placed near to the estimated location.

To identify the range of area that a register can be placed, the authors proposed a controlled variable, called *slack*, for each fan-in and fan-out node of a register. It is an upper bound of the interconnect delay between the node and the register, and is defined as below.

$$(\textit{cycle time} - \textit{path delay}) / 2 + (\textit{estimated interconnect delay}) \quad (3.1)$$

where (*estimated interconnect delay*) denotes the estimated interconnect delay from a register to its fan-in and fan-out nodes, (*cycle time* − *path delay*) is the total slack of the path, *cycle time* is the clock period of the circuit and *path delay* is the total delay of the path. This value is divided by two because there are two parts in the path, the starting and the ending parts. Under this definition, the slack of a node on the critical path connected to the starting or the ending register is exactly equal to the estimated interconnect delay. This makes the starting or the ending register very likely to be placed at the estimated location.

According to the slacks, the placement range for each register can be calculated. Since all gate locations are fixed and registers are removed, there are many slots left in the layout for the registers generated after retiming. If the number of registers increases significantly after retiming, more slots will be added to both ends of a row to accommodate all the registers.

Chapter 4

Retiming with Gate and Wire Delay [2]

4.1 Introduction

Retiming is a well-known optimization technique that can be used to minimize the clock period, the usage of registers, or a combination of both objectives of a sequential circuit. By relocating the registers under several restrictions, retiming can improve the circuit performance without changing the original functionalities.

Since retiming was first formulated a decade ago, much effort has been made to improve the efficiency of the algorithm [32] or to apply the technique in various areas like physical planning [6], circuit partitioning [28], power reduction [31], and testability [13] and so on. However, retiming has still shown little practical uses in industry. This is especially true in the physical design stages in which a lot of long global wires exist between blocks. Ignoring wire delay in retiming will yield solutions that are unreliable to be used.

In this chapter, we study the problem of retiming with gate and wire delay. Two approaches are proposed to solve the problem optimally and near-optimally, assuming that the delay of a wire grows directly proportional to

its length. In section 4.3, the optimal approach will be presented. This approach solves the problem optimally by transforming it into a special case of a mixed-integer linear programming problem that can be solved in polynomial time. In section 4.4, the near-optimal approach will be described. The second approach is an improvement of the first one in terms of execution time and gives near-optimal solutions as shown by the experimental results.

When our works was published in ICCAD 2003, another optimal approach solving the same problem was proposed by Lin et al at the same conference. In section 4.5, their approach will also be discussed.

4.2 Problem Formulation

A sequential circuit can be modeled by a directed graph $G(V, E)$, where each vertex $v \in V$ represents a combinational gate, and each directed edge $e_{uv} \in E$ denotes a connection from the output of gate u to the input of gate v. Each vertex v has a gate delay of d_v and each edge e_{uv} has a wire delay of d_{uv} if no register lies along the edge. Notice that wire delay is assumed to be proportional to the length of the wire. This assumption is valid because buffers can be inserted optimally to make the delay of a wire exhibit a linear behavior [27]. Besides, $w(u, v)$ denotes the number of registers on edge e_{uv} before retiming is applied.

A retiming solution can be viewed as an vertex-to-integer labeling, $r : V \longrightarrow Z$ where Z is the set of integers, such that the equation: $w_r(u, v) = w(u, v) + r(v) - r(u) \geq 0$ holds for every edge in G, where $w_r(u, v)$ denotes the number of registers on edge e_{uv} after retiming is performed. The retiming label $r(v)$ for a vertex v represents the number of registers moved from its outputs to its inputs.

Unlike the traditional formulation where wire delay is ignored, a retiming solution should also specify the exact positions of the registers on each edge.

As wire delay dominates in DSM designs, the exact position of each register will affect the clock period. Therefore, the problem of retiming with gate and wire delay can be formulated as below:

PROBLEM STATEMENT. *Given a sequential circuit $G(V, E)$, we want to find a retiming solution, i.e., a retiming label $r(v)$ at each vertex v, and the exact positions of the $w_r(u, v)$ registers on each edge e_{uv} such that the clock period of the circuit is minimized.*

4.3 Optimal Approach [2]

This approach is an extension of the MILP approach in the paper [22] and can solve the problem optimally, i.e., relocating the registers to give the minimum possible clock period. We observed that even if both gate and wire delay are considered, the problem can still be formulated as a MILP problem and a polynomial time algorithm that solves the problem exists [21].

This section is organized as follows: First, the mathematical programming framework for retiming discussed in [22] is presented. Next, we will show how we extend the method to handle both gate and wire delay, thus giving a modified approach to solve the problem optimally. At the end, the implementation outline will be discussed.

4.3.1 Original Mathematical Framework for Retiming

In section 6 of [22], *A Mathematical-Programming Framework for Retiming*, the authors presented an algorithm for clock-period minimization based on a special case of mixed-integer linear programming (MILP). As usual, wire delay was ignored.

The authors perform the clock period feasibility check based on the following special case of MILP problem that can be solved in polynomial time [21].

PROBLEM SPECIAL-MILP. *Let S be a set of m linear inequalities of the form $x_i - x_j \leq a_{ij}$ on the unknowns $x_1, x_2, \ldots, x_n$, where the a_{ij} are given real constants. Let k be given. Determine feasible values for the unknowns x_i subject to the constraint that x_i is integer for $i = 1, 2, \ldots, k$ and real for $i = k+1, k+2, \ldots, n$, or determine that no such values exist.*

Together with the condition of a legal retiming that the number of registers on every edge after retiming must remain non-negative, the authors characterized the feasible clock period checking by an MILP according to Lemma 9 in [22].

LEMMA 9. *Let $G =< V, E, d, w >$ be a synchronous circuit, and let T be a positive real number. Then there exists a retiming r of G such that $\Phi(G_r) \leq T$ if and only if there exists an assignment of a real value $s(v)$ and an integer value $r(v)$ to each vertex $v \in V$ such that the following conditions are satisfied.*

9.1 $-s(v) \leq -d(v)$ for every vertex $v \in V$,

9.2 $s(v) \leq T$ for every vertex $v \in V$,

9.3 $r(u) - r(v) \leq w(u, v)$ for every edge $e \in E$,

9.4 $s(u) - s(v) \leq -d(v)$ for every edge $e \in E$ such that $r(u) - r(v) = w(u, v)$.

The function $s(v)$ for every vertex in G can be viewed as the longest combinational delay path from a register to the outgoing pin of gate v. This physical reasoning of $s(v)$ explains why condition 9.1 and 9.2 are necessary. Condition 9.1 means that the minimum combinational delay at those outgoing pins of gate v must at least equal the gate delay of v itself, it is because the closest possible position that you can place a register toward a gate is at the incoming pins of it but it never got retimed "inside" the gate. Condition 9.2 means that the maximum combinational delay at those outgoing pins of a gate v must be

less than or equal to T; otherwise, T is infeasible. Condition 9.3 guarantees a legal retiming as $w_r(u,v) = w(u,v) + r(v) - r(u) \geq 0$ always holds.

Finally, condition 9.4 considers the case when there is no register got retimed on an edge e_{uv}. If there is no register on an edge e_{uv}, obviously, the maximum combinational delay at vertex v, $s(v)$, is no less than the maximum combinational delay at vertex u, $s(u)$, plus the gate delay of v itself. Otherwise, if there are registers on the edge, this constraint is not needed.

By using the substitution $s(v) = T(R(v) - r(v))$, the authors was able to eliminate the defining clause in condition 9.4 and gave the following system of difference inequalities in Theorem 10 which were in the form of the special case of MILP that can be solved in polynomial time.

10.1 $r(v) - R(v) \leq -d(v)/T$ for every vertex $v \in V$,

10.2 $R(v) - r(v) \leq 1$ for every vertex $v \in V$,

10.3 $r(u) - r(v) \leq w(u,v)$ for every edge $e \in E$,

10.4 $R(u) - R(v) \leq w(u,v) - d(v)/T$ for every edge $e \in E$.

Any solution to the conditions above yields a solution to the conditions in Lemma 9; equivalently, a feasible retiming solution such that the clock period T is satisfied.

4.3.2 A Modified Optimal Approach

In the previous section, a mathematical-programming framework for retiming was briefly discussed. In this section, we will show how the framework can be extended to handle both wire and gate delay to obtain the optimal clock by retiming.

In the modified optimal approach, we still have condition 9.1, 9.2 and 9.3 since they must also be satisfied even when wire delay is considered. However, we have to modify condition 9.4 in order to handle wire delay. Obviously,

condition 9.4, $s(u)-s(v) \leq -d(v)$ for every edge $e \in E$ such that $r(u)-r(v) = w(u,v)$, does not take wire delay d_{uv} into account. To tackle this problem, we proposed the following condition in place of condition 9.4:

(*) $\quad s(u)+d_{uv}+d_v-T(w(u,v)+r(v)-r(u)) \leq s(v)$ for every edge $e \in E$,

Divide (*) by T and use the same substitution $s(v) = T(R(v) - r(v))$, it can be rewritten as below:

(**) $\quad R(u) - R(v) \leq w(u,v) - d_{uv}/T - d_v/T$ for every edge $e \in E$,

Let us analyze the physical meaning of (*) to see why this condition is proposed. First, consider if there is no register on the edge e_{uv}, i.e., $w(u,v)+r(v)-r(u) = 0$, (*) becomes $s(u)+d_{uv}+d_v \leq s(v)$. It means that the maximum delay at the output of gate v, $s(v)$, is no less than the maximum delay at the output of gate u, $s(u)$, plus the wire delay of e_{uv} and the gate delay of v itself. Obviously, this is true because there is no register on the wire.

Next, if there are registers on the edge e_{uv}, i.e., $w(u,v)+r(v)-r(u) \geq 1$, we subtract an amount of T for each register from $s(u)+d_{uv}+d_v$ on the left hand side of (*). Since $s(u)$ is no greater than T and there is at least one register on the edge, we can always place the first register at a distance of $T-s(u)$ from the outgoing pin of gate u which can be viewed as a subtraction of T from $s(u)+d_{uv}+d_v$. For the rest of the registers, we can place them at a distance of T from each other.

If $s(u)+d_{uv}-T(w(u,v)+r(v)-r(u))$ becomes negative, it means that there are many registers on e_{uv} and the last one can be placed right in front of the input of gate v, and thus $s(v)$ can be as small as d_v. On the other hand, if $s(u)+d_{uv}+d_v-T(w(u,v)+r(v)-r(u))$ is non-negative, it means that the last register on the edge is at a distance of $s(u)+d_{uv}-T(w(u,v)+r(v)-r(u))$ away from the input of gate v. If there is no register on e_{uv}, $s(v)$ will be equal

to $s(u)+d_{uv}+d_v$. Otherwise, $s(v)$ will be equal to $s(u)+d_{uv}+d_v-T(w(u,v)+r(v)-r(u))$.

The modified condition is very much the same as condition 9.4 which still preserves the form of $x_i - x_j \leq a_{ij}$ so that it remains as a special case of MILP and can be solved efficiently.

Given a feasible solution that satisfies condition 9.1, 9.2, 9.3 and the proposed condition (**), we are able to compute the locations for every register such that the clock period under test, T, is satisfied. First, consider an edge e_{uv}, since $r(v)$ and $s(v)$ are all known where $d_v \leq s(v) \leq T$ for every vertex v in G, we can place the first register, if any, at a distance of $T - s(u)$ from the outgoing pin of gate u and others at a distance of T from the previous one until reaching the incoming pin of gate v. Once we reach the incoming pins of gate v, we can place the remaining registers there. Such an arrangement of registers must be feasible since this is derived from a feasible solution satisfying the necessary conditions.

For example, consider an edge e_{uv}, assuming that $T = 5$, $s(u) = 3$, $d_{uv} = 17$, and $w_r(u,v) = 5$. The positions of the registers on e_{uv} would look like:

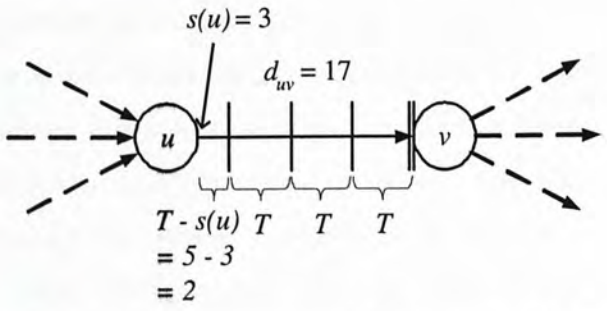

Figure 4.1: Register arrangement in the optimal algorithm.

Together with the technique of binary search, we can find the optimal clock period of retiming with wire and gate delay in polynomial time. In addition, we know exactly where to place the registers on each edge.

Implementation Outline

The implementation of the optimal algorithm is based on [21] which has given three different efficient algorithms to solve the special case of MILP, i.e., to determine a feasible solution for the MILP in the form of $x_i - x_j \leq a_{ij}$, or to determine there is no solution.

The first two algorithms R and M are mainly based on the idea of the Bellman-Ford algorithm [8] and can be implemented straightforwardly using adjacency-list or adjacency-matrix, giving a runtime of $O(|V||V_I||E|)$, where $|V|$ denotes the total number of variables (integer + real), $|V_I|$ denotes the number of integer variables and $|E|$ denotes the number of edges in the constraint graph, i.e., the number of constraints. The last algorithm D is the most efficient implementation using many techniques such as reweighting in Dijkstra's algorithm [17] and Fibonacci heap [14] to achieve a complexity of $O(|V||E| + |V||V_I|lg|V|)$.

Since the near-optimal fast algorithm (to be discussed in the next section) was implemented using some simple data structures, we chose to implement algorithm M for the optimal approach such that we could have a fair comparison of the two approaches by using similar data structures.

Every vertex is associated with a retiming label $r(v)$ and a real function $s(v)$. We initialized a graph, G_M, with $2 \times |V|$ number of vertices, half of them representing $r(v)$ and the other half representing $s(v)$. Constraints are extracted by reading the vertices and edges in G, and the corresponding constraints will be added to G_M, i.e., by inserting edges into G_M.

For example, given two vertices u and v, and an edge e_{uv}, we first consider condition 10.1 which states that $r(v) - R(v) \leq -d_v/T$ for every vertex $v \in V$. This condition requires us to add an edge of weight $-d_v/T$ from vertex $R(v)$ to vertex $r(v)$ in G_M. Similarly, condition 10.2 means inserting an edge of weight 1 from vertex $r(v)$ to $R(v)$ in G_M. Both condition 10.1 and 10.2 are associated

with vertices in G. Next, condition 10.3 which states that $r(u)-r(v) \leq w(u,v)$ for every edge $e_{uv} \in G$ can be enforced by inserting an edge of weight $w(u,v)$ from $r(v)$ to $r(u)$ in G_M. Similarly, condition 10.4 can be enforced by inserting an edge of weight $w(u,v) - d_v/T$ from $R(v)$ to $R(u)$ in G_M. Finally, algorithm M will be applied on G_M to determine a feasible solution and registers will be placed accordingly.

4.4 Near-Optimal Fast Approach [2]

In the previous section, an algorithm that solves the problem of retiming with gate and wire delay optimally is presented. However, the optimal approach runs too slow to be practical, though its runtime is polynomial in theory. To increase the applicability of retiming, a near-optimal fast approach is proposed and implemented. My main contributions in this approach were implementing the algorithm, testing and conducting the experiments.

This near-optimal fast approach gives optimal solution if gate delay is neglected and only wire delay is considered. When both types of delay are considered, it gives solutions that are very close to the optimal as shown by the experimental results.

In this approach, we first replace each gate by a wire of the same delay and then solve the problem optimally and efficiently with wire delay only. Those registers got retimed "inside" a gate are moved either to the input or the output wires of the gate. Once the number of registers on each wire is fixed, we can determine the exact positions of the registers by a linear program such that the clock period is minimized. In the following, we first present how the retiming problem with wire delay only can be solved optimally. Then, we discuss in detail how gate delay can be handled simultaneously.

4.4.1 Considering Wire Delay Only

In this subsection, we assume that $d_v = 0$ for all $v \in V$, i.e., gate delay is ignored. On the other hand, wire delay is assumed to be proportional to the length of the wire as buffers can be inserted optimally. We will show that the problem of retiming with wire delay only can be transformed into a problem of solving a set of linear constraints.

Transformation to a Directed Acyclic Graph

In the first step, the retiming graph model G is transformed into a directed acyclic graph (DAG), G', by breaking the back edges obtained when traversing the graph in a depth-first manner. This is illustrated in fig. 4.2 (start DFS from A).

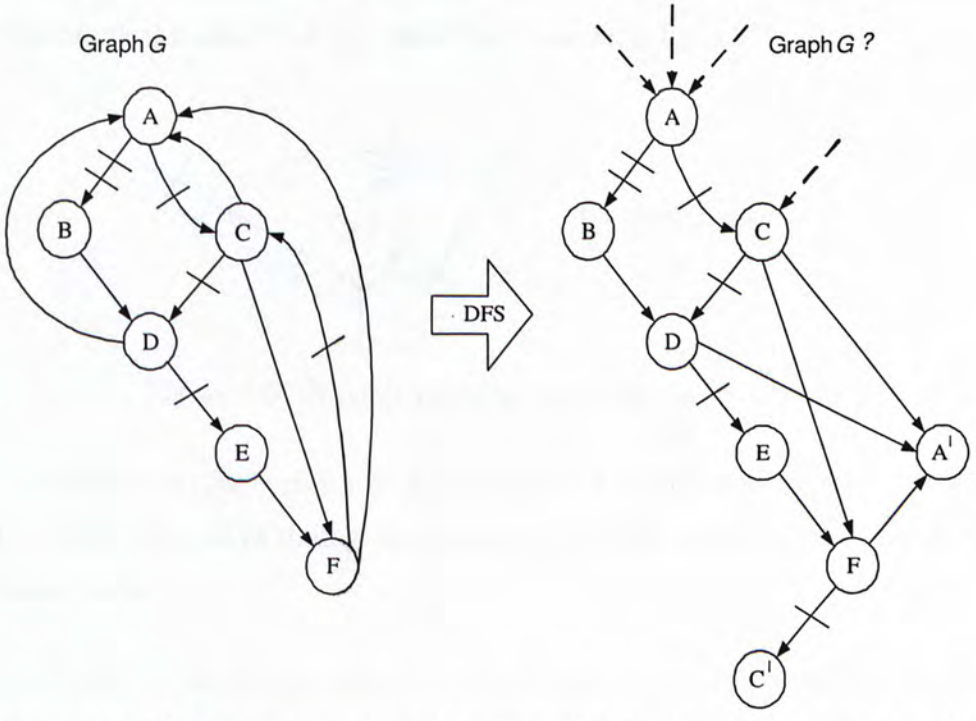

Figure 4.2: DFS transformation of a sequential circuit retiming graph model.

As we can see in the figure above, there are some extra vertices introduced

(A' and C' in this example). Let E_b be the set of back edges and V_b be the set of all vertices with an incoming back edge. For each $v \in V_b$, we introduce an extra vertex v'. Also, we remove the back edge e_{uv} from G and introduce an extra edge $e_{uv'}$ in G'. Therefore, $G'(V', E')$ is a new graph where $V' = V \cup \{v' | v \in V_b\}$ and $E' = E - E_b \cup \{e_{uv'} | e_{uv} \in E_b\}$. Obviously, G' is a DAG.

Timing Constraints

In the second step, a set of timing constraints is constructed so that the Bellman-Ford algorithm can be applied to solve the problem.

For each vertex $v \in V'$, we introduce a timing variable t_v that is the maximum wire delay from the closest register connecting to an input of gate v. Since back edges are removed from G', for all $v \in V_b$, another variable $t_{v'}$ is introduced to capture the timing requirement on v due to the back edges. The physical meaning of t_v is graphically shown in fig. 4.3.

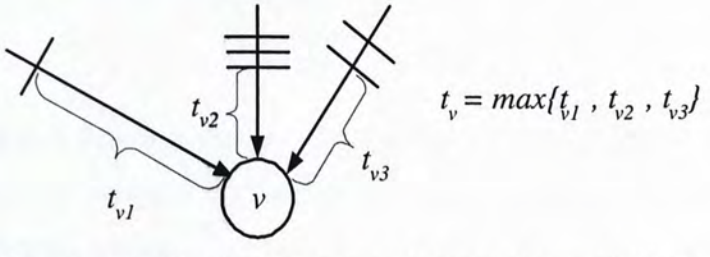

Figure 4.3: Physical meaning of the timing variable t_v.

Define $in(v)$ for each $v \in V'$ to be the set of vertices with an edge pointing to v in G'. The set of timing constraints on t_v can be expressed as equation 4.1 shown below.

$$\forall v \in V, \quad t_v \geq max_{u \in in(v)}\{t_u + d_{uv} - (w(u,v) + r(v) - r(u))T\} \qquad (4.1)$$

Consider an edge e_{uv}, we can place the registers at a distance of at most T from each other, and the longest possible length covered by the registers on an edge e_{uv} after retiming would be $(w_{uv}+r(v)-r(u))T$. Since the total amount of wire delay is t_u+d_{uv} where t_u represents the delay "inherited" from vertex u and d_{uv} represents the wire delay of e_{uv}, we can subtract the amount of delay covered by the registers to compute the maximum wire delay from the closest register on e_{uv} to the input of v. Therefore, t_v is given by constraint 4.1 when every incoming edge of v is considered.

Since t_v denotes the maximum wire delay from a register connecting directly to an input of gate v, the following constraint must hold as well, assuming that T denotes the target clock period.

$$\forall v \in V', \quad t_v < T \tag{4.2}$$

In addition, the following constraint is required to guarantee a valid retiming solution in the original graph G.

$$\forall v \in V_b, \quad t_{v'} \leq t_v \tag{4.3}$$

Constraint 4.3 is to make sure that there are enough registers on any path from vertex u to vertex v' for all pair of u and v where $v \in V_b$. Besides, since v' is a duplicated vertex of v, the following constraint must be satisfied.

$$\forall v \in V_b, \quad r(v') = r(v) \tag{4.4}$$

If there exists a feasible solution of t_v and $r(v)$ for all $v \in V'$ satisfying 4.1-4.4, we can perform a legal retiming on G with clock period T.

Algorithm Outline

Having constructed the constraints 4.1-4.4 for every edge in G', we are able to translate the set of constraints into a system of difference inequalities in the form of: $x_i - x_j \leq a_{ij}$ by simple variable substitution.

The main idea of the translation is to express t_v for all $v \in V'$ in terms of t_u and $r(u)$ where $u \in V_b$. As shown in constraint 4.1, t_v is related to t_u for every edge e_{uv} in G' where G' is a DAG. We can recursively express the constraint using simple substitution until reaching the ancestor vertices in V_b. The whole approach is described in procedure ALG_NEAROPT below.

Procedure ALG_NEAROPT;
Input : A sequential circuit C with wire delay only;
Output : An optimally retimed circuit of C;
begin
  build graph $G(V, E)$ from C;
  build graph $G'(V', E')$ from G using DFS;
  C_{up} = a feasible clock, C_{low} = an infeasible clock;
  do
    $T = (C_{up} + C_{low})/2$;
    setup constraints 4.1-4.4 from G';
    translate the constraints into a set of difference inequalities,
    denoted by I;
    if(BellmanFord(I) returns a solution) then
      success = 1; $C_{up} = \mathrm{T}$;
    else
      success = 0; $C_{low} = \mathrm{T}$;
  while($(C_{up} - C_{low})$ / $C_{up}) > \epsilon$;
  $T = C_{up}$;
  compute $R(v)$ and $r(v)$ for each vertex $v \in V$;
  compute the exact position of each register on a wire;
end.

First, the algorithm performs a DFS on G to discover all the back edges and constructs the corresponding G'. Next, it enters a binary search process

until an optimal T is found. For each iteration, the constraints 4.1-4.4 are constructed and the Bellman-Ford algorithm is used to determine the feasibility. Once we have reached a desired bound of error, ϵ, the binary search process terminates.

4.4.2 Considering Both Gate and Wire Delay

While the discussion in the previous subsection assumes the existence of wire delay alone, we have also proposed a method to deal with gate delay simultaneously. The basic idea is to split every vertex v into two vertices v_1 and v_2, and an edge $e_{v_1v_2}$ with $d_{v_1v_2}$ equals the gate delay of v as shown in fig. 4.4.

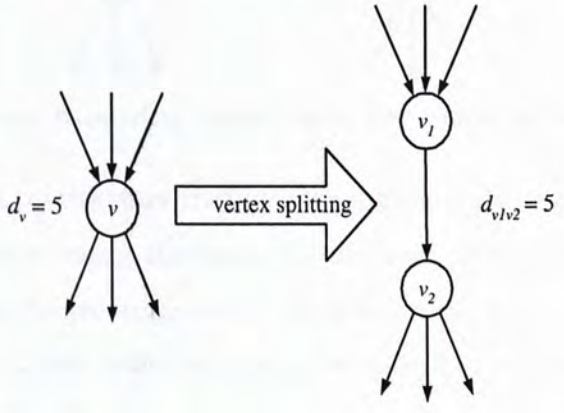

Figure 4.4: Vertex splitting to handle gate delay.

After splitting the vertex, we can use the algorithm described in the previous subsection to perform retiming (i.e., the algorithm that assumes wire delay only). However, the retiming solution obtained may not be a feasible solution for the original graph since some of the registers may be retimed into a wire that represents a gate originally. Therefore, a post-processing method is required to fix the problem.

The post-processing proposed consists of two main steps. First, we will remove the registers from those wires that represent gates. They are removed either backward to the input wires or forward to the output wires of the gate,

depending on their positions in the retiming solution. If the register is closer to the input wires, it will be moved backward; otherwise, it will be moved forward. Fig. 4.5 shows an example of moving a register backward. Once we have fixed the number of registers on each edge, we can use linear programming to minimize the clock period of G.

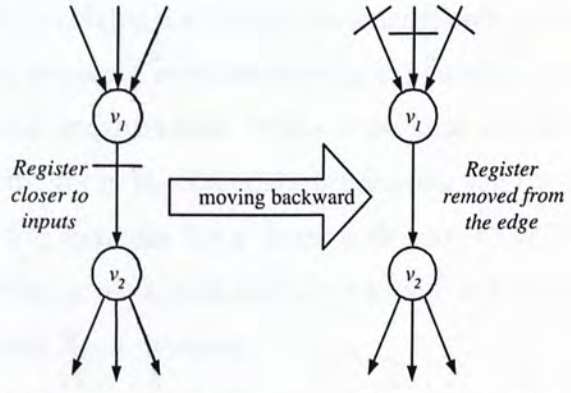

Figure 4.5: Removing register on a wire representing a gate.

Experimental results show that by splitting the vertices into two to handle gate delay and then using the technique of linear programming to minimize the clock period, the resulting clock is still very close to the theoretical lower bound [29]. The largest deviation is reported to be 3.01% from the lower bound among all the ISCAS89 benchmarks, while most other gives the optimal lower bound retiming solutions.

4.4.3 Computational Complexity

The runtime complexity is $O(|V_b| \times max\{|E|, |V_b|^2\} \times lg(K/\epsilon T_{opt}))$ where $|V_b|$ denotes the number of vertices having incoming back edges in G, $|E|$ denotes the number of edges in G, K denotes the difference of C_{up} and C_{low} at the beginning of the binary search, ϵ denotes the degree of accuracy we want and T_{opt} is the optimal clock for G. Notice that comparing to the runtime of the optimal approach, $O(|V||V_I||E|)$, the near-optimal approach is of greater practical interest.

In fact, a closer investigation of the near-optimal approach reveals that its significant speedup over the optimal approach is the result of solving the longest path problem on a much smaller graph given the same sequential circuit. Notice that the cores of both approaches involve the application of the Bellman-Ford algorithm; however, the near-optimal approach manages to *reduce* the problem to solving a single-source longest path problem on a reduced graph. Therefore, apparent runtime speedup is possible.

In section 4.4.4, experimental results show that the proposed algorithm can retime the circuits in the ISCAS89 benchmark suite within a reasonable amount of time. For example, for a circuit with more than 22K gates and 32K wires, the algorithm gives a near-optimal solution in 83.56 seconds on a PC with a 1.8GHz Intel Xeon processor.

4.4.4 Experimental Results

We implemented the two approaches, *optimal* and *near-optimal fast*, on a 1.8GHz Intel Xeon PC with 512KB cache and 512MB RAM. In our experiments, we used the circuits from the ISCAS89 benchmark suite and implemented them using 0.25 μm process technology. Besides, we layout the circuits using Silicon Ensemble and wire delays are extracted from the layout accordingly. Notice that wire delays are assumed to be proportional to the wire lengths and wire lengths are measured using the shortest Manhattan distance between the two connecting points.

In our implementation, the lower and upper bounds of the binary search are set to 0 and 100ns respectively. For the optimal approach, we set the error bound ϵ to 0.01% and call its resulting clock period T_{opt}. For the near-optimal approach, the error bound is set to 1%. After assigning the registers retimed into a gate to the appropriate wires, a linear program is set up to relocate the registers on the wires to get the smallest possible clock period T^*. On average,

Circuit	Number of nodes in V	Number of edges in E	Number of nodes in V_b	Number of edges in E_H
s1488	655	1405	27	627
s1494	649	1411	30	749
s3271	1574	2707	112	3360
s3330	1791	2890	56	1200
s3384	1687	2782	98	2041
s4863	2344	4093	154	20413
s5378	2781	4261	66	2554
s6669	3082	5399	67	1876
s9234	5599	8005	325	26570
s13207	7953	11302	550	44825
s15850	9774	13794	603	100738
s35932	16067	28590	884	163945
s38417	22181	32135	1657	308790

Table 4.1: Benchmark statistics for retiming.

the binary search iterates 16.5 times for the optimal approach and 9.6 times for the near-optimal approach.

The ISCAS89 benchmark statistics are shown in table 4.1. The first column shows the names of the circuits under test. The second and third columns give the number of vertices and the number of edges in the graph G respectively. Notice that all the circuits are not strongly connected. The number of vertices and edges listed are those after the addition of a source vertex, a target vertex and the associated edges. The fourth and fifth columns show the number of vertices and the number of edges in the constraint graphs to be solved respectively, as discussed in section 4.4.1. These two values are dependent on the vertex being chosen to be the root in the depth-first traversal. In our implementation, we pick the additional source vertex s as the root. We notice that using other vertices as root does not improve the results significantly.

The experimental results are shown in table 4.2. The second column shows the runtime of the near-optimal approach while the third column shows the runtime of the optimal approach. We can see that the near-optimal approach is much more efficient than the optimal approach, and this is especially true in

Circuit	CPU Time		Clock Period		
	Near-Optimal (sec)	Optimal (sec)	T^* (ns)	T_{opt} (ns)	$\frac{T^*-T_{opt}}{T_{opt}}$ (%)
s1488	0.28	5.62	18.85	18.82	0.16
s1494	0.25	4.37	20.78	20.78	0.00
s3271	1.09	33.70	10.24	10.24	0.00
s3330	0.50	43.14	27.05	27.05	0.00
s3384	0.74	25.19	24.21	24.16	0.21
s4863	3.12	87.75	23.58	23.58	0.00
s5378	1.16	138.68	27.27	27.25	0.07
s6669	1.91	177.59	23.07	22.96	1.00
s9234	4.08	512.86	42.73	42.73	0.00
s13207	8.11	1161.07	72.34	72.34	0.00
s15850	24.02	1545.59	67.82	67.82	0.00
s35932	61.25	8644.27	29.59	29.54	0.17
s38417	83.56	7680.79	36.53	36.52	0.03

Table 4.2: Runtime of the algorithms and clock periods obtained.

those larger circuits. The reason why the optimal approach runs so slowly can be explained by its $O(|V||V_I||E|)$ time complexity (though it is polynomial).

For example, consider the circuit s4863 which has 2344 vertices and 4093 edges. Since every vertex is associated with an integer function $r(v)$ and a real function $s(v)$, the number of vertices in the constraint graph would be 2×2344, i.e., $|V| = 4688$ and $|V_I| = 2344$. Next, consider the constraints 4.1-4.4, every vertices would have constraints 4.1 and 4.2, and every edge would involve constraints 4.3 and 4.4. Therefore, the total number of constraints would be $2 \times 2344 + 2 \times 4093$, i.e., $|E| = 4688 + 8186 = 12974$. Obviously, $|V||V_I||E|$ would result in billions of unit step for even such a circuit with moderate size, not to mention the number of steps required in those larger circuits.

4.5 Lin's Optimal Approach [23]

To have a comprehensive study of this problem, Lin's approach is described in the following sections. They focused on applying retiming under the context of system-on-chip (SoC) designs, i.e., instead of dealing a netlist of simple gates, they applied retiming on a netlist of macro-blocks.

Their problem formulation is basically the same as the one discussed in section 4.2 except that they assumed that a global routing solution was given and they classified the set of edges E in $G(V, E)$ into two types: E_1 and E_2. E_1 denotes the set of edges that does not allow registers to be placed on them while E_2 denotes the rest. Since we are not able to insert registers into the macro-blocks of a SoC design, E_1 is introduced to represent those forbidden areas. Besides, given a global routing solution, the Steiner points of the nets are also treated as vertices in V, apart from the gates and the macro-blocks of a circuit.

4.5.1 Theoretical Results

Notations and Constraints

Since a retiming solution must include the positions of the registers on each wire if wire delay is considered, the authors overcome this problem by specifying the arrival time of every vertex with respect to a clock period. For each vertex $v \in V$, they also used a timing variable, t_v, as discussed in section 4.4 (denoted by $t(v)$ in [23]), to represent its arrival time with respect to the nearest register on its incoming paths.

Given t_v, the positions of the registers directly fanning into v can be found, and the positions of the other registers can also be computed. Using this notation, the requirements for a retiming solution can be expressed as follows.

First, to ensure that no register is inserted on forbidden edges, we need

$$r(x) = r(y), \quad \forall e_{xy} \in E_1. \tag{4.5}$$

Then, to make sure the availability of registers, we must have

$$w(u,v) + r(v) - r(u) \geq 0, \quad \forall e_{uv} \in E_2. \tag{4.6}$$

Besides, we need the following inequalities to guarantee that the arrival times are all achievable.

$$t_y \geq t_x + d(x,y), \quad \forall e_{xy} \in E_1. \tag{4.7}$$

$$t_v \geq t_u + d(u,v) - (w(u,v) + r(v) - r(u))T, \quad \forall e_{uv} \in E_2. \tag{4.8}$$

Finally, we need the ensure that t_v is nonnegative and smaller than the target clock period T.

$$0 \leq t_v \leq T, \quad \forall v \in V. \tag{4.9}$$

As we can see, the fixed period wire retiming problem is expressed as a mixed integer linear program (MILP) given by 4.5-4.9. In addition, we also need to set $t_v = 0$ for every primary input v, and $r(v) = 0$ for every primary input or output v. In general, solving a MILP problem is NP-hard. However, this problem can be solved in polynomial time by simplifying the set of constraints into a special case of MILP that will be discussed in the subsequent subsections.

Lower and Upper Bounds of Clock Period

From constraints 4.5-4.9, the authors derived some lower bounds and upper bounds of the feasible clock period. First, from 4.7-4.9, it is easy to observe that any feasible clock period T must satisfy

$$T \geq T_1 \stackrel{def}{=} max_{e_{xy} \in E_1} d(x, y).$$

They defined the path delay $d(p)$ of a path p to be the sum of the delays of the edges on p. If a path forms a cycle c, $d(c)$ denotes the cycle delay. Similarly, they defined $w(p)$ and $w(c)$ as the number of registers on path p and cycle c respectively. By applying 4.5, 4.7, and 4.8 on any cycle and PI-PO path, a feasible clock period T must also satisfy

$$T \geq T_2 \stackrel{def}{=} max_{c \in cycle} \frac{d(c)}{w(c)}, \tag{4.10}$$

$$T \geq max_{p \in PI \rightarrow PO} \frac{d(p)}{w(p) + 1}. \tag{4.11}$$

To simplify the notation, the authors introduced a virtual vertex M as well as directed edges from each PO to M with zero delay and weight, and directed edges from M to each PI with zero delay and unit weight. In this way, the set of cycles covers every PI-PO path. As a result, 4.11 can be incorporated into 4.10.

To derive an upper bound of the feasible clock, the authors showed that if each connected component in the subgraph $G_1 = (V, E_1)$ is a complete bipartite graph, the optimal clock period can be upper bounded by $T_1 + T_2$. Otherwise, a procedure is proposed to compute a tighter bound as follows. First, an optimal retiming solution without considering the forbidden edges is found. This can be done based on the computation of T_2. Next, a local adjustment to move registers out of the forbidden edges is done to obtain a feasible solution. In this step, the increase in clock period is kept as small as possible.

In general, there are two kinds of forbidden edges: those that form a complete bipartite graph and the others that do not. If a register is moved out of an edge that forms a complete bipartite graph, there is at most a T_1 increase in

clock period. Otherwise, if a register is moved out of an edge not in a complete bipartite graph, the increase will be larger. However, the local adjustment step will keep the number of registers being moved out of a non-forbidden edge to minimum.

Fixed Period Wire Retiming

In this subsection, we will see how the MILP problem can be solved such that a retiming solution that satisfies a clock period T can be found, if T is achievable. The authors used an approach similar to that of Leiserson and Saxe [22] to solve the problem in polynomial time.

First of all, they define the sequential delay of a path p, $sd(p)$, as follows:

$$sd(p) = d(p) - w(p)T$$

where T is the clock period. Next, they define the sequential delay $sd(u, v)$ of any two vertices u and v to be

$$sd(u, v) = max_{p \in u \to v} sd(p).$$

Then, consider the following set of inequalities

$$r(v) - r(u) \geq \lceil sd(u, v)/T \rceil - 1 \quad \forall u \neq v \in V \tag{4.12}$$

Using the definition of sequential delay, (4.12) means that for any edge $e_{uv} \in E_2$,

$$w(u, v) + r(v) - r(u) \geq \lceil d(p)/T \rceil - 1 \geq 0.$$

Obviously, (4.12) implies (4.6). In addition, the authors showed that equations (4.5) and (4.12) give a solution to the fixed period wire retiming problem under the lower bound condition of (4.12). This is stated in the following theorem.

Theorem 1 The fixed period wire retiming problem is feasible if and only if (4.5), (4.10) and (4.12) have a solution.

4.5.2 Algorithm Description

In this section, their proposed algorithm in solving the fixed period wire retiming problem will be discussed. The algorithm can be divided into three major parts as follows.

The first step is the computation of the lower and upper bound for the binary search. Condition 4.10 actually denotes a cycle property in graph also known as the *maximum cycle ratio problem* (MCRP). To solve MCRP, a lot of algorithms have been designed and presented such as Burns' [1], Lawler's [19], Howard's [10], etc. Since Howard's algorithm has the lowest complexity, they adopted an improved version of this algorithm in their implementations. To calculate the upper bound, they followed the steps described in the previous subsections.

In the second step, binary search is used to find the optimal clock period. Given a target clock period T, the authors apply the Johnson's all-pair shortest path algorithm [8] to compute $sd(u,v) = max_{p \in u \to v} sd(p)$, i.e., the sequential delay of all pairs of u to v. Next, a new graph rG is created with all the vertices in V and all edges e_{uv} where $u \neq v \in V$ of weight $\lceil sd(u,v)/T \rceil - 1$ if there exists a path from u to v in G. An extra node is added to rG as well as directed edges from each PO to it and from it to each PI with zero weight. The Bellman-Ford algorithm is then applied in rG to check if there exists a positive cycle. If no positive cycle exists, T is a feasible clock period. Otherwise, T cannot be achieved. The bounds of the binary search are then adjusted accordingly.

In the final step, the optimal clock period and the corresponding retiming label $r(v)$ are used to calculate the maximum arrival time t_v for all $v \in V$.

By Theorem 1, a feasible solution for all t_v is guaranteed. To further improve the performance of their proposed algorithm, pruning is proposed to eliminate redundancy in the constraints. Experimental results showed that optimal solutions were resulted but the runtime was still too slow for large benchmark circuits.

4.5.3 Computational Complexity

To calculate the lower and the upper bounds for the binary search, Howard's algorithm is invoked. In [9], the runtime of Howard's algorithm is bounded by $O(|V||E|\alpha)$ and $O(|V|^2|E|(w_{max} - w_{min})/\epsilon)$, where α is the number of simple cycles in G. Next, Johnson's all-pair shortest path algorithm is used to calculate the sequential delay $sd(u, v)$ between any pair of u and v if v is reachable from u in G. The algorithm makes use of the Bellman-Ford algorithm and Dijkstra's algorithm. Using Fibonacci heap to implement the priority queue in the Dijkstra's algorithm, the Johnson's all-pair shortest path algorithm is bounded by $O(|V|^2 lg|V| + |V||E|)$.

However, the dominant part of runtime is spent on applying the Bellman-Ford algorithm to rG. In the worst case, rG is a complete graph and consists of $|V|^2$ number of edges. Solving this will take $O(|V|^3)$. Therefore, the time complexity of the whole algorithm is $O(|V|^3 lg\frac{T_u - T_l}{\epsilon})$, where ϵ measures the degree of accuracy, T_u and T_l are the upper and the lower bounds of the binary search respectively.

4.5.4 Experimental Results

Lin et al implemented their approach on a PC with two 2.4GHz Intel Xeon processors and 1GB RAM. They performed retiming on the ISCAS89 benchmark suite. In the absence of delay information for the circuits, they randomly assigned delay values between 1.0 and 2.0 units to gates (they treated them as

Circuit	$\|V\|$	$\|E\|$	Runtime (sec.)	T_{opt}
myex	21	24	0.00	18.7
s386	519	700	1.97	51.1
s400	511	665	1.64	32.2
s444	557	725	2.23	35.2
s838	1299	1206	8.79	76.0
s953	1183	1515	9.76	60.6
s1238	1581	2100	7.88	100.3
s1488	2054	2780	35.17	70.6
s1494	2054	2792	34.13	76.9
s5378	7205	8603	684.60	111.2

Table 4.3: Runtime of Lin's approach and the optimal clock periods obtained.

macro-blocks) and 0.2 to 5.0 for wires. Since they focused their discussion on chip level, the delay range was intentionally chosen in order for wire delay to be commensurate or even many times larger than block delay.

Experimental results are shown in table 4.3. The first column shows the name of the circuits. The number of vertices and edges are shown in the second and third column respectively. Notice that the number of vertices and edges are more than the number of gates and wires in the original benchmark circuits. It is because they took the global routing solutions into account, i.e., the Steiner points and their associated edges were treated as extra vertices and edges respectively. The fourth column shows the runtime of the algorithm while the fifth column shows the optimal clock period found. On average, the binary search iterated 5 times for each circuit.

To compare Lin's approach and our two proposed approaches in terms of runtime, we can measure the runtime of a single iteration in the binary search for circuits with similar sizes. For example, consider the circuit s3384 in table 4.2 and the circuit s1238 in table 4.3. The former circuit has 1687 vertices and 2782 edges while the latter one has 1581 vertices and 2100 edges. Both circuits have similar sizes.

The average runtime of a single iteration on s3384 using our optimal and near-optimal approaches are 1.526 seconds and 0.077 seconds respectively (assuming that the number of iterations taken are 16.5 and 9.6 respectively). However, the average runtime of a single iteration on s1238 using Lin's approach is 1.576 seconds, although they had used a faster computing platform to perform the experiments.

4.6 Summary

Regardless of how powerful the technique of retiming is, it draws little attention in industrial practice because it does not take wire delay into consideration. While most traditional retiming algorithms have ignored wire delay safely, we have proposed two elegant approaches to solve the problem of retiming with gate and wire delay efficiently and simultaneously.

Our first approach is extended from the MILP approach in paper [22] and can solve the problem optimally. Our second approach is an improvement over the first one in terms of practical applicability. The main idea is to solve the problem of retiming in a reduced constraint graph, which involves vertices with incoming back edges only. We have implemented both algorithms and compared their performance on the ISCAS89 benchmark suite. Experimental results show that the near-optimal fast approach gives solutions that are only 0.13% larger than the optimal on average but in a much shorter runtime.

Compared with the proposed method in [23], our approaches can give solutions for large benchmark circuits within a reasonable amount of time whereas their method could not. In fact, no experimental results for large circuits were reported in their paper.

Chapter 5

Register Insertion in Placement [36]

5.1 Introduction

While the technique of retiming can be applied to improve the performance of a sequential circuit, the placement of registers after the optimization is another challenge. Since a multiple-pin net is represented as a branch of edges in a retiming graph model, there is no information about the topology of the routes or the position of each register in the net. Thus, it is unknown whether the clock period obtained from retiming can be realized in the actual layout. If we arbitrarily share the registers among the interconnections of a net, it may violate the retiming solution and render the optimization process useless.

In this chapter, we study the problem of realizing a retiming solution on a global netlist by inserting registers in placement to achieve the target clock period. This problem involves two main sub-problems, namely, *topology finding* and *register placement.*

As we have mentioned before, a net is modeled as a branch of edges in the retiming graph, the problem of *topology finding* refers to the determination of an optimal sharing of the registers among the fanout edges of a net given the geometric positions of the connected gates such that the optimal clock period

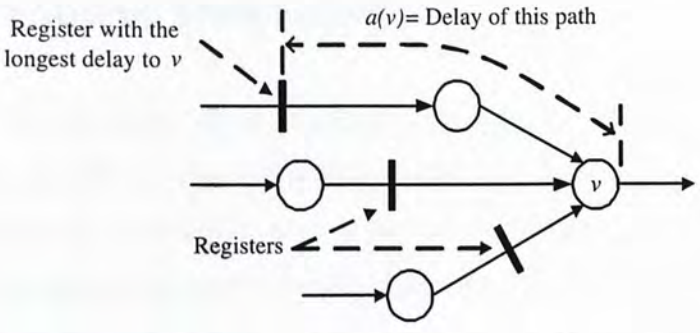

Figure 5.1: An illustration of the definition of $a(v)$ for a gate v.

obtained from retiming can be preserved. After obtaining the topology tree of a net, we need to find an appropriate position for each register given the constraints in placement (some occupied areas do not allow register insertion) and this problem is known as *register placement.* Given a placement (we used standard cell design in our experiments) and a retiming solution (we used the technique in [2] to generate the retiming solutions in our experiments), our proposed algorithm can insert registers into the placement solution to preserve the clock period as much as possible.

Notice that our algorithm has no dependency on the retiming algorithm being used, as long as it considers wire delay and gives a retiming solution with a target clock period, retiming labels and the maximum arrival time at each gate output, $a(v)$. Fig. 5.1 shows an example that illustrates the definition of $a(v)$.

Our algorithm can find the optimal topology, i.e., using the minimum number of registers while preserving the clock period, for nets with 4 or fewer pins. Since nets with 4 or fewer pins constitute over 90% of nets in a circuit on average, the proposed algorithm offered an agreeable performance as shown in the experiments. Nearly all the nets had their best topology found and registers inserted successfully while achieving the target clock period.

5.2 Problem Formulation

PROBLEM STATEMENT. *Given a placed sequential circuit and a retiming solution, i.e., an optimal clock period clk, a retiming label $r(v)$ at each gate v and the maximum arrival time $a(v)$ at the output of gate v, we want to insert registers into the circuit layout to realize the retiming solution, preserving the clock clk as much as possible.*

We can represent the circuit as a graph $G(V, E)$, where each vertex $v \in V$ corresponds to a combinational gate, and each directed edge $e_{uv} \in E$ represents a connection from the output of gate u to the input of gate v. Let $w(u, v)$ be the number of registers along the edge e_{uv}, d_{uv} be the wire delay of edge e_{uv} if no register lies along the edge. Note that the wire delay d_{uv} is assumed to be proportional to the shortest Manhattan distance between u and v.

Now, consider a net $N(s, D, L)$, where s denotes the driving gate, D denotes the set of all driven gates, and L denotes the set of interconnections between s and each of the gates $d_i \in D$. Obviously, $\{s\} \cup D \subseteq V$ and $L \subseteq E$. For each edge $e_{sd_i} \in L$, we have a value $w_r(s, d_i)$ representing the number of registers along the edge e_{sd_i} after retiming. The problem is to insert the minimum number of registers for this net into the circuit according to the retiming solution such that the clock period is preserved as much as possible. This problem comprises two main sub-problems known as *topology finding* and *register placement*. *Topology finding* is the problem of finding a topology, Υ_N, of net N given the exact geometric positions of the gates such that the minimum number of registers is used and the target clock period is preserved. *Register placement* is the problem of finding the corresponding position for each register given the topology Υ_N of net N.

A topology $\Upsilon_N = (P, K)$ is a tree (an acyclic graph with no designated root yet) that describes the structure of net N on the plane. Each node

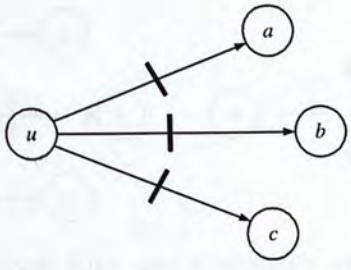

Figure 5.2: Graph model of a 4-pin net in which each edge has a single register after retiming.

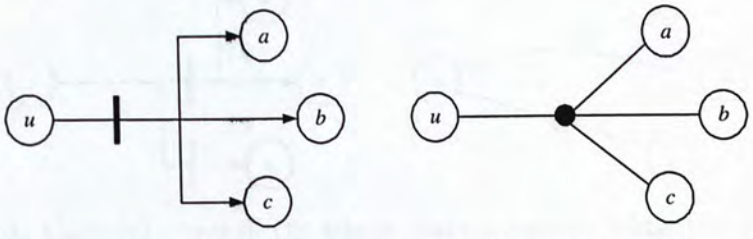

Figure 5.3: Case (a) - a single register is shared among the edges (maximum sharing). The topology tree of this configuration is shown on the right.

$p \in P$ corresponds to either a combinational gate or a register, and each edge $k_{uv} \in K$ represents a physical connection between gate u and gate v. Each node $p \in P$ that has only one adjacent node in Υ_N, i.e., $deg(p) = 1$, represents a combinational gate while an internal node $p \in P$ that has more than one adjacent node, i.e., $deg(p) > 1$, represents a register. In fig. 5.2, an example of a 4-pin net in which each *source-to-sink* edge has a register after retiming is shown. There are five possible register sharing topologies in this example: (a) all the edges share a single register (maximum sharing) as shown in fig. 5.3; (b) each edge has its own register (no sharing) as shown in fig. 5.4; (c) for the rest three equivalent cases, two of the edges share a single register while the other has a separate one, as shown in fig. 5.5.

Although we can always identify the topology tree which has the maximum sharing of registers for a net, it is not always possible to place the registers on a chip using that topology while preserving the given clock period. Using case (a) in fig. 5.3 as an example, suppose the clock period resulted from

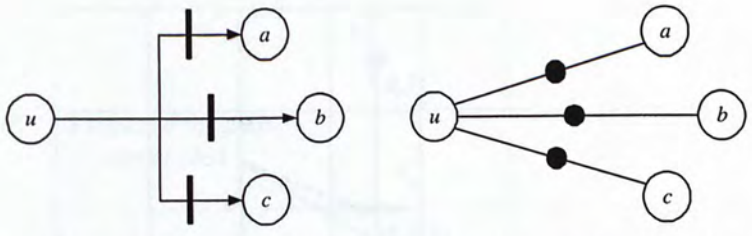

Figure 5.4: Case (b) - each edge has a separate register (no sharing). The topology tree of this configuration is shown on the right.

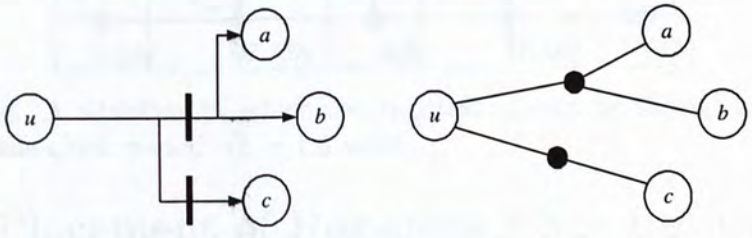

Figure 5.5: Case (c) - two of the edges share a register while the other has a separate one. The topology tree of this configuration is shown on the right.

retiming, clk, equals 1.5 units and the positions of gate u, a, b and c are $(0,0)$, $(-3,0)$, $(0,3)$ and $(3,0)$ respectively, as depicted in fig. 5.6 below.

Obviously, it is impossible to share a single register among the three edges without clock violation. Three separate registers have to be allocated and inserted exactly at $(-1.5,0)$, $(0,1.5)$ and $(1.5,0)$ for edge e_{ua}, e_{ub} and e_{uc} respectively in order to satisfy clk.

Even if we have a feasible topology tree, it can happen that the suggested position for a register has been occupied by some other gates, i.e., the target area is blocked, and we have to look for another appropriate position. Section 4 will address how a feasible topology tree can be found and how the positions of the registers can be finalized.

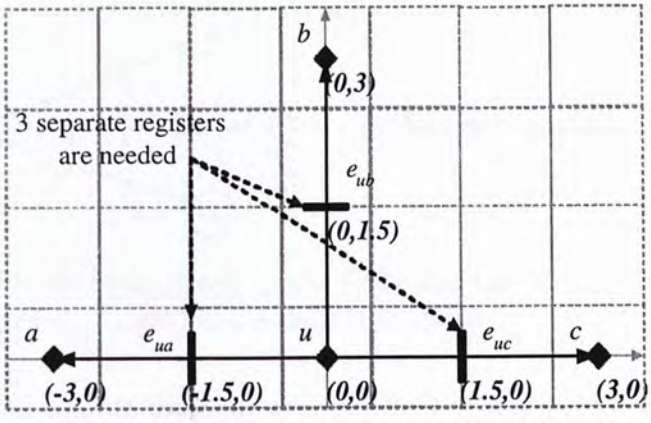

Figure 5.6: A situation in which the registers cannot be shared in order to preserve the clock period $clk = 1.5$ units.

5.3 Placement of Registers After Retiming

5.3.1 Topology Finding

In this section, an algorithm is proposed to find the topology of a net given the constraints in placement such that maximum sharing of registers is achieved and the clock period is preserved. This method can find the optimal topology for a net with 4 or fewer pins, and can give near-optimal solution for a net with 5 or more pins according to the experimental results.

Algorithm Description

Given a net $N(s, D, L)$, a clock period clk, and the maximal arrival time at the output of gate v, $a(v)$, we can obtain a feasible topology tree of N, Υ_N, as described below.

First, we construct the best possible topology $\Upsilon_{N_{opt}}$ for N, i.e., a topology having the minimum number of internal nodes (an internal node represents a register). Obviously, the number of internal nodes in $\Upsilon_{N_{opt}}$ equals $Q = max_{d_i \in D}\{w_r(s, d_i)\}$, where $w_r(s, d_i)$ denotes the number of registers on the edge e_{sd_i} after retiming. We label each internal node as r_i representing the

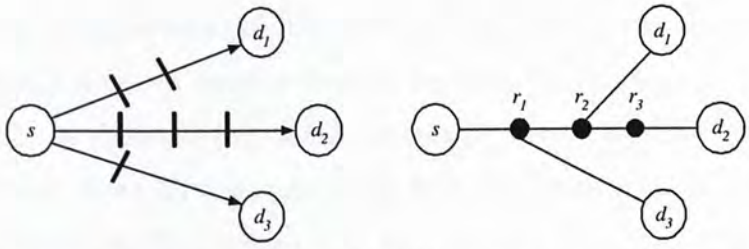

Figure 5.7: The retiming graph model (left) and the corresponding best possible topology $\Upsilon_{N_{opt}}$ (right) of a 4-pin net example.

i-th register on the net from the source s for $1 \leq i \leq Q$. An example of the retiming graph model and the corresponding best possible topology $\Upsilon_{N_{opt}}$ for a 4-pin net is shown in fig. 5.7.

We call the region of the plane where a register r can be placed the *candidate region* of r and is denoted by $C(r)$. For consistency, the candidate region $C(v)$ of a combinational gate v is the position of v itself, i.e., its coordinates (x_v, y_v), since v is fixed after placement. An *δ-extended region* of a region $\Re$, denoted by $R^{+\delta}(\Re)$, is the region of the plane at a distance δ or less from some points in $\Re$, assuming that the distance between two points is measured by their shortest Manhattan distance.

Besides, we define an *adjacent-gate region* for each node p in a topology tree, denoted by $A(p)$, as an δ-extended region from its candidate region $C(p)$, i.e., $A(p) = R^{+\delta}(C(p))$ where δ is defined differently for different types of nodes. The physical meaning of $A(p)$ refers to the region on the plane such that it encompasses *all* the possible positions for an adjacent gate of p in the net. Therefore, the value of δ for $A(p)$ of a node p is described as follows. If node p is an internal node, δ equals clk. If node p represents a driven gate, δ equals $a(p) - d_p$, where $a(p)$, given by the retiming solution, is the maximum arrival time at the output of gate p and d_p is the gate delay of p. Otherwise, node p represents the driving gate, and we set δ to $clk - a(p)$. Notice that all these regions are 45°-rotated rectangles on the rectilinear plane because of the Manhattan distance measurement.

Starting from the best possible topology $\Upsilon_{N_{opt}}$, we will modify the topology incrementally until an optimal feasible topology Υ_N is obtained for net N. First of all, we choose the node that represents the driving gate s as the root in $\Upsilon_{N_{opt}}$ and direct all the edges away from s. Then, we will process each internal node r_i in $\Upsilon_{N_{opt}}$ from $i = Q$ to $i = 1$, i.e., from the furthest register to the closest register to the driving gate s, in the following manner.

For each internal node r_i with a set of children $q_1, \ldots, q_m$, find a minimal set of all the overlapping regions between $A(q_j)$ for $1 \leq j \leq m$, denoted by $Y_{min} = (y_1, \ldots, y_k)$, such that the union of the elements in Y_{min} covers at least one point from each region $A(q_j)$. For each y_l in Y_{min}, we call the number of regions that y_l has covered at least a point as the *size* of y_l, denoted by $s(y_l)$. Sort the elements in Y_{min} in a non-ascending order of their sizes, using a greedy procedure ALGSETY as described below.

Procedure ALGSETY(r_i, Υ_N);
begin
  overlapped := a boolean flag;
  $Y_{min} \longleftarrow \phi$;
  add $A(q_1)$ to Y_{min}, i.e., $y_1 \longleftarrow A(q_1)$;
  for $j = 2$ *to* m
    overlapped $\longleftarrow$ false;
    for $l = 1$ *to* $|Y_{min}|$
      if $(y_l \cap A(q_j) \neq \phi)$
        $y_l \longleftarrow y_l \cap A(q_j)$;
        sort elements in Y_{min} in a non-ascending order of their sizes;
        overlapped $\longleftarrow$ true;
        break;
      end if;
    end for;

if (*overlapped* == false)
increment $|Y_{min}|$ by 1;
add $A(q_j)$ to Y_{min} at the end, i.e., $y_{|Y_{min}|} \longleftarrow A(q_j)$;
end if;
end for;
OUTPUT(Y_{min});
end.

Notice that the union of the elements in Y_{min} covers at least one point from each region, $A(q_j)$, for $1 \leq j \leq m$. Next, we can remove all the edges from r_i to its children $q_1, \ldots, q_m$ in $\Upsilon_{N_{opt}}$, split the node r_i into k new internal nodes, $n_1, \ldots, n_k$, where node n_l corresponds to element y_l in Y_{min} for $1 \leq l \leq k$. In addition, we will assign region y_l as the candidate region of n_l, i.e., $y_l = C(n_l)$, for all l.

Starting from the y_l whose $s(y_l)$ is the largest in Y_{min}, add an edge from n_l to each q_j that has no parent node and whose $A(q_j)$ is covered by y_l. Repeat this step until all y_l have been processed. Finally, add an edge from the parent node of r_i to every newly generated internal nodes n_l and r_i can then be removed from the topology tree. The above operations are described in the procedure ALGMODITREE below.

Procedure ALGMODITREE(r_i, Y_{min}, Υ_N);
begin
remove all the edges from r_i to its children $q_1, \ldots, q_m$ in Υ_N;
instantiate k new internal nodes, $n_1, \ldots, n_k$, where $k = |Y_{min}|$;
assign region y_l as the candidate region of n_l, i.e.,
$y_l = C(n_l)$, for all l;
for $l = 1$ *to* k
for $j = 1$ *to* m
if ($y_l \cap A(q_j) \neq \phi$ and q_j has no parent node)

add an edge from n_l to q_j;
end if;
end for;
add an edge from the parent node of r_i to n_l;
end for;
remove r_i;
OUTPUT(Υ_N);
end.

After visiting all the internal nodes r_i in $\Upsilon_{N_{opt}}$ and modifying the topology as described above, we will get a new topology tree Υ_N at the end such that the clock period *clk* is preserved. The whole algorithm for *topology finding* of a net N is described in the procedure ALGTOPOTREE.

Procedure ALGTOPOTREE(N);
begin
construct the best possible topology $\Upsilon_{N_{opt}}$ for net N;
$\Upsilon_N \longleftarrow \Upsilon_{N_{opt}}$;
for $i = Q$ *to* 1
$Y_{min} \longleftarrow$ ALGSETY(r_i, Υ_N);
$\Upsilon_N \longleftarrow$ ALGMODITREE(r_i, Y_{min}, Υ_N);
end for;
OUTPUT(Υ_N);
end.

Optimality Proof

To prove the correctness of the above algorithm, we have the following three lemmas.

Lemma 1 Given a set of n 45°-rotated rectangles $R_1, \ldots, R_n$ on a rectilinear

plane, if $R_1 \cap \ldots \cap R_n \neq \phi$, then $R^x(R_1) \cap \ldots \cap R^x(R_n) \neq \phi$, where x is a non-negative real number.

Lemma 2 Given a set of n 45°-rotated rectangles $R_1, \ldots, R_{n-1}$ and S on a rectilinear plane, if $S \cap R_i \neq \phi$ for $1 \leq i \leq n-1$ and $R_1 \cap \ldots \cap R_{n-1} \neq \phi$, then $S \cap (R_1 \cap \ldots \cap R_{n-1}) \neq \phi$.

Lemma 3 Given two 45°-rotated rectangles, A and B, on a rectilinear plane, we denote the n *times clk-extended regions* of A and B as A_n and B_n respectively, i.e., $A_n = R^{+(n \times clk)}(A)$ and $B_n = R^{+(n \times clk)}(B)$. Suppose $A \cap B = R_{AB} \neq \phi$, we denote the n *times clk-extended region* of R_{AB} by $(R_{AB})_n$, i.e., $(R_{AB})_n = R^{+(n \times clk)}(R_{AB})$. It is claimed that if there exists a point $x \in A_n \cap B_n$, $x \in R^{+clk}((R_{AB})_{n-1})$ for all $n \geq 1$.

Proof: We prove by induction on n.

Base case:

Consider the case when $n = 1$. Suppose $x \in A_1 \cap B_1$, the *clk*-extended region from the position of x is given by $R^{+clk}(x)$. Obviously, $R^{+clk}(x) \cap A_0 \neq \phi$ and $R^{+clk}(x) \cap B_0 \neq \phi$ because $x \in A_1 \cap B_1$. Since $A_0 \cap B_0 \neq \phi$ ($\because A_0 = A, B_0 = B$ and $A \cap B \neq \phi$), $R^{+clk}(x) \cap (A_0 \cap B_0) \neq \phi$ by *lemma 2*. Therefore, $x \in R^{+clk}((R_{AB})_0)$ and the claim is true for $n = 1$.

Inductive step:

Assume that the claim is true for $n = j - 1$, where j is a positive integer ≥ 2, i.e., if there exists a point $x \in A_{j-1} \cap B_{j-1}$, $x \in R^{+clk}((R_{AB})_{j-2})$. Consider the case when $n = j$. Given a point $x \in A_j \cap B_j$ and the *clk*-extended region from its position is denoted by $R^{+clk}(x)$. Obviously, $R^{+clk}(x) \cap A_{j-1} \neq \phi$ and $R^{+clk}(x) \cap B_{j-1} \neq \phi$ because $x \in A_j \cap B_j$. By *lemma 1*, since $A_0 \cap B_0 \neq \phi$, $A_{j-1} \cap B_{j-1} \neq \phi$. Therefore, $R^{+clk}(x) \cap (A_{j-1} \cap B_{j-1}) \neq \phi$ by *lemma 2*. By the induction hypothesis, if $R^{+clk}(x) \cap (A_{j-1} \cap B_{j-1}) \neq \phi$, $R^{+clk}(x) \cap R^{+clk}((R_{AB})_{j-2}) \neq \phi$. Therefore, $x \in R^{+clk}((R_{AB})_{j-1})$.

Theorem 1 The proposed algorithm finds a topology that maximizes the sharing of registers for an i-pin net, where $2 \leq i \leq 4$, and the given clock period clk is preserved.

Proof: We prove the three possible cases one-by-one.
Case 1: $i = 2$

This case is trivial because there is only one source s, one sink t_1 and one edge e_{st_1} in a 2-pin net, there is no other edges to share registers with. The algorithm will start from the furthest internal node r_Q and take the adjacent-gate region of t_1, $A(t_1) = R^{+(a(t_1)-d_{t_1})}(C(t_1))$, as the candidate region of r_Q, i.e., $C(r_Q) = A(t_1)$. Next, the algorithm will process node r_{Q-1} and take the adjacent-gate region of r_Q, $A(r_Q) = R^{+clk}(C(r_Q))$, as the candidate region of r_{Q-1}, i.e., $C(r_{Q-1}) = A(r_Q)$.

By substitution, $C(r_{Q-1})$ can be represented as an extended region from the position of the sink t_1, as $C(r_{Q-1}) = R^{+((a(t_1)-d_{t_1})+clk)}(C(t_1))$. The algorithm repeats the above steps until it reaches the first internal node r_1 where $C(r_1) = R^{+((a(t_1)-d_{t_1})+(Q-1)\times clk)}(C(t_1))$. Since the retiming solution is valid, the distance between s and t_1 will not exceed $(clk - a(s)) + ((Q-1) \times clk) + (a(t_1) - d_{t_1})$. Therefore, the algorithm will find the candidate regions for every register and return the best possible topology when it terminates.

Case 2: $i = 3$

Given a 3-pin net, let s be the source, and t_1 and t_2 be the two sinks. Let $w_r(s, t_1)$ and $w_r(s, t_2)$ be p and q respectively, where $1 \leq p \leq q$. Suppose that there exists a topology tree of maximum register sharing for the 3-pin net such that the first k registers, where $1 \leq k \leq p$, are shared (notice that if the k-th register can be shared, the h-th register can be shared where $1 \leq h \leq k$), and that the algorithm cannot find such a topology.

Since the algorithm cannot find that optimal topology, it must fail to find an overlapping region for the k-th register to be shared. At the point of failure, the algorithm should find that the regions $R^{+((a(t_1)-d_{t_1})+clk\times(p-k-1))}(t_1)$ and $R^{+((a(t_2)-d_{t_2})+clk\times(q-k-1))}(t_2)$ do not overlap. However, these two regions encompass *all* the possible positions for the placement of the k-th register from t_1 and t_2 respectively such that the clock period clk would not be violated. Therefore, should the k-th register be able to be shared as assumed, it must lie within these two regions and the algorithm must be able to find it. Contradiction occurs.

Case 3: $i = 4$

Given a 4-pin net, let s be the source, and t_1, t_2 and t_3 be the three sinks. Let $w_r(s,t_1)$, $w_r(s,t_2)$ and $w_r(s,t_3)$ be p, q and r respectively, where $1 \leq p \leq q \leq r$. Suppose the algorithm is attempting to share the k-th register where $1 \leq k \leq p$, i.e., it is trying to find a minimal subset of the overlapping regions such that it covers all the extend regions $R^{+((a(t_1)-d_{t_1})+clk\times(p-k-1))}(t_1)$, $R^{+((a(t_2)-d_{t_2})+clk\times(q-k-1))}(t_2)$ and $R^{+((a(t_3)-d_{t_3})+clk\times(r-k-1))}(t_3)$, denoted by A, B and C respectively. Notice that we only consider when $k \leq p$ and assume that the three paths from s to t_1, t_2 and t_3 are not merged yet (i.e., no sharing of registers from $k+1$ to r). It is because, otherwise, the situation will fall into case 1 or case 2 discussed above.

There are 4 distinct cases. First, if A, B and C are disjoint, it means that the k-th register cannot be shared and the algorithm will introduce three new internal nodes to represent the registers and continues with the next internal node r_{k-1}. Second, if A, B and C overlap with each other, it means that the k-th register can be shared among t_1, t_2 and t_3. The algorithm will introduce a single internal node to represent the register and continues. The correctness of the algorithm in these two cases is trivial and will not be elaborated here.

The third case is, without loss of generality, that $A \cap B \neq \phi$ and $B \cap C \neq \phi$ but $A \cap C = \phi$. Denote the region $A \cap B$ as R_{AB} and the region $B \cap C$ as R_{BC}. There are three possible options that the algorithm can choose from when evaluating the k-th register: (1) it does not share the k-th register and introduces three different registers for the sinks; (2) it shares the k-th register between t_1 and t_2 but a separate one for t_3; (3) it shares the k-th register between t_2 and t_3 but a separate one for t_1. Our algorithm will choose arbitrarily either (2) or (3) as the number of adjacent-gate regions covered by R_{AB} and R_{BC} are the same, but it will never choose (1). We assume that the algorithm chooses (2) in the following analysis.

First, we compare the choices of (1) and (2). Notice that (1) can be better than (2) only when the three separate paths can be merged together at a subsequent step of processing register h where $1 \leq h \leq k$, while the combined path of t_1 and t_2 and the path of t_3 cannot be merged at the h-th register. We are going to show that this will not happen.

If we choose (1), suppose that there exists a point x on the plane such that $x \in A_j \cap B_j \cap C_j$, where A_j, B_j and C_j represent the j times clk-extended regions of A, B and C respectively, during a subsequent step of processing register h where $1 \leq h \leq k$. By *lemma 3*, it is shown that $x \in R^{+clk}((R_{AB})_{j-1})$, where $(R_{AB})_{j-1}$ is the $(j-1)$ times clk-extended region from R_{AB}. This means that if it is possible to share the h-th register among the three edges without sharing the k-th register at the first place, by choosing (2), i.e., to share the k-th register between t_1 and t_2, the algorithm will also be able to share the h-th register among the edges.

Next, we compare the choices of (3) and (2) similarly. Suppose we choose (3) and there exists a point x on the plane such that $x \in A_j \cap (R_{BC})_j$, where A_j and $(R_{BC})_j$ represent the j times clk-extended regions of A and R_{BC} respectively, during a subsequent step of processing register h where $1 \leq h \leq k$. Obviously, there exists a point y covered by $A_j \cap B_j \cap C_j$, i.e., $y \in A_j \cap B_j \cap C_j$.

By *lemma 3*, $y \in R^{+clk}((R_{AB})_{j-1})$, i.e., $y \in R^{+clk}((R_{AB})_{j-1}) \cap C_j$, so the h-th register will also be shared among the three source-to-sink paths by choosing (2). Therefore, (2) is no worse than (3). As a result, the algorithm can find the optimal solution by choosing arbitrarily either (2) or (3) (using the greedy algorithm).

Finally, if two pairs of the regions overlap while the other is disjoint, i.e., $A \cap B \neq \phi$ but $A \cap C = \phi$ and $B \cap C = \phi$, the analysis is similar to the third case above.

5.3.2 Register Placement

In this section, we discuss how registers are actually placed using the topology tree yielded from the algorithm discussed in the previous subsection. Using an idea similar to the technique in [30, 15], the positions of the registers are determined.

Since some of the chip areas are occupied by the standard cells, we need to know where on the chip a register can be placed. To tackle this problem, we divide the chip into a mesh of $m \times n$ grids. For each grid g_{ij}, we keep track of its center coordinates, $(x_{g_{ij}}, y_{g_{ij}})$, and the size of the free space in the grid, $f(g_{ij})$. We finalize the position for a register in the following manner.

Given a topology tree Υ_N, choose arbitrarily an internal node r to be the root of Υ_N, and direct the edges of Υ_N away from r. Starting from the root r, we choose a grid whose center is contained in $C(r)$, i.e., the candidate region for placing the register r, and it has the largest free space available. We denote this grid as $g(r)$. If $f(g(r)) \geq z$, where z denotes the size of a register, we take the center of $g(r)$ as the position of the register r. Otherwise, we allow a controlled degree of inaccuracy by extending $C(r)$ one grid width further, i.e., $R^{+g_w}(C(r))$, where g_w represents the width of a grid. Repeat the same process using $R^{+g_w}(C(r))$ instead of $C(r)$ in the search of a feasible grid for

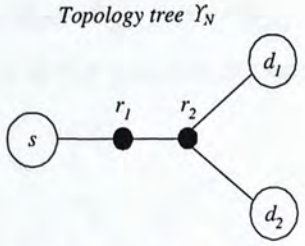

Figure 5.8: The topology tree of a 3-pin net Υ_N where r_1 and r_2 are two shared registers.

placing register r. If no such grid is found, we report that this register cannot be placed. This could happen because the register counts may increase greatly after retiming.

Let $q_1, \ldots, q_m$ be the set of internal nodes which are the children of r in a topology tree Υ_N. After fixing the position of r, register q_j, for $1 \leq j \leq m$, is placed arbitrarily in its candidate region $C(q_j)$ provided that it is at a distance of clk or less from r. After visiting all the internal nodes of Υ_N, the position of each register is located.

Suppose we have a 3-pin net $N(s, D, L)$ and its topology tree Υ_N is shown in fig. 5.8. The topology tree Υ_N shows that the two driven gates d_1 and d_2 will share two registers represented by the internal node r_1 and r_2. In this example, we assume that $clk = 3$ units. Consider a 5×5 mesh as shown in fig. 5.9, where the positions of the driving gate s and the two driven gates, d_1 and d_2, are assumed to be the centers of the grids containing them correspondingly, i.e., gate s is located at $(4, 0)$, gate d_1 is located at $(0, 4)$ and gate d_2 is located at $(2, 4)$. Suppose Υ_N is rooted at node r_1 and the algorithm has fixed its position at $(1, 0)$, let us examine how the position of r_2 is determined.

The candidate region $C(r_2)$ of r_2 covers the centers of grids g_{03}, g_{04}, g_{12}, g_{13}, g_{14}, g_{23} and g_{24}. Starting from the position of register r_1, the algorithm expands a rectangle of distance clk from it, denoted by $R^{+clk}(r_1)$ as shown. Next, the algorithm will find that $C(r_2) \cap R^{+clk}(r_1)$ is not empty and covers the center of grid g_{12} and g_{13} - the candidate positions of register r_2. If the free

space of g_{12} is greater than that of g_{13}, i.e., $f(g_{12}) \geq f(g_{13}) \geq z$, the algorithm will assign the center of g_{12} as the position of register r_2.

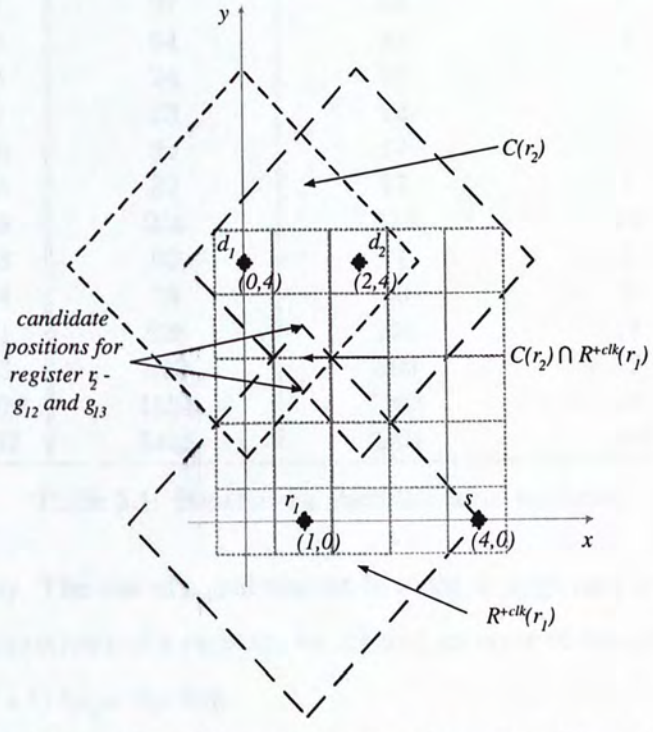

Figure 5.9: An illustration of how the final position of register r_2 is determined.

5.4 Experimental Results

We performed retiming and our proposed register placement algorithm on the ISCAS89 benchmark. The program was implemented in C language and run on a 1.5GHz Intel Pentium IV processor with 256KB cache and 512MB RAM.

In our experiments, we implemented the circuits using a 0.35μm CMOS standard cell library from Austria Micro Systems and Silicon Ensemble was used to layout the design with a setting of 50% row utilization. Gate delays were referenced from the data book while wire lengths were estimated using the Manhattan distance between the connected cells. We scaled the wire delay according to [4] in which a 1mm wire was assumed to have a delay of 150ps

Circuit	Number of logical registers after retiming	Nets with 4 or fewer edges with registers	Nets with 5 or more edges with registers
s641	87	53	0
s713	94	54	1
s820	24	23	0
s832	23	22	0
s1196	31	17	1
s1238	32	17	1
s1269	259	113	14
s1488	90	71	2
s1494	78	60	2
s3271	826	276	18
s4863	622	360	22
s15850.1	1554	1203	26
s35932	5455	2601	280

Table 5.1: Benchmark statistics after retiming.

approximately. The size of a grid was set to twice as large as a D-type flip flop. During the placement of a register, we allowed an error of one-grid width, i.e., the width of a D-type flip flop.

The ISCAS89 benchmark statistics are shown in table 5.1. The first column indicates the name of the circuits under test. The second column shows the number of logical registers existed in the retiming graph model after retiming. The number of registers had increased after retiming for most of the circuits because the retiming method that we used did not minimize the number of registers as one of its objectives. Although this increase in register counts does hinder our algorithm to place registers, it is not the main concern addressed in this paper. Next, the third column shows the statistics of the number of nets containing 4 or fewer edges with registers whereas the fourth column shows the number of nets having 5 or more edges with registers. Clearly, a larger number of nets in a sequential belongs to net with 4 or fewer edges with registers, thus our proposed algorithm can find the optimal solutions for almost all nets in each circuit.

Circuit	Min. # of regs. using max. sharing	Actual # of regs. by our method	# of regs. placed within candidate region	# of regs. placed with controlled error	CPU time (sec.)
s641	53	53	53	0	0.01
s713	55	55	55	0	0.02
s820	23	23	23	0	0.01
s832	22	22	22	0	0.01
s1196	18	18	18	0	0.00
s1238	18	18	18	0	0.00
s1269	127	127	127	0	0.02
s1488	73	73	73	0	0.02
s1494	62	62	62	0	0.01
s3271	342	438	438	0	0.18
s4863	408	417	417	0	0.25
s15850.1	1264	1264	1264	0	2.52
s35932	2899	2899	2899	0	13.41

Table 5.2: Results of register placement with clock preservation.

The experimental results are shown in table 5.2. In the second column, the minimum possible number of register required after sharing is shown, i.e., assuming that every net could be realized using the best topology. The third column shows the number of registers that have actually been inserted after using our proposed algorithm. It can be observed that the numbers in the third column are the same as those in the second column except for circuit s3271 and s4863. This observation showed that almost all the nets in our test cases could have their registers placed using the best topology, revealing that our proposed algorithm can often find near-optimal solutions for register insertion.

The fourth column shows the number of registers that are placed within their candidate regions while the fifth column shows the number of registers that are placed outside their candidate regions but with a controlled error range (one grid size). As we can see, all the registers are placed in their candidate regions successfully in all the test cases. Finally, the CPU runtime is shown in the last column.

5.5 Summary

In this chapter, we have proposed an algorithm that solves the problem of register insertion on global wire given a placed sequential circuit and a retiming solution. The proposed algorithm can preserve a given clock period with a controlled error using as few registers as possible in contrast to many previous works with post-retiming register placement that have the problem of clock period violation. In addition, the algorithm is also proved to be giving the optimal topology for nets with 4 or fewer pins. Since this type of nets makes up for about 90% of the nets in a sequential circuit on average, the algorithm performs very well and effectively under most situations.

Together with any powerful retiming method which is designed to handle global netlist with block and wire delays, our proposed algorithm can be applied to locate where a register should be inserted to pipeline long global interconnects such that the target clock is preserved. This is particularly useful in today's designs in which multiple clock cycles are required to propagate a signal across a global wire.

Improvements can be made to handle the situation when there is no room for the candidate region of a register. Instead of scanning the neighboring grids for free spaces, incremental shifting and reshuffling of cells can be performed to free continuous rooms for register insertions.

Chapter 6

Conclusion

Because of the wire delay dominance in today's DSM design, propagation of electrical signals on global wires within one single clock cycle becomes very difficult, if not impossible. As a result, multiple clock cycles are introduced on these long wires by inserting more registers, i.e., wire pipelining. To pipeline the wires of a circuit without changing its functionality or increasing its latency, a sequential circuit optimization, called retiming, can be applied. The basic idea and literature review of retiming are presented in the early chapters.

However, there are two major problems that hinder the use of retiming on global nets. The first problem is the lack of wire delay consideration in retiming while the second problem is the placement of registers after the optimization. In this thesis, we have proposed solutions to the above problems.

Wire delay is ignored in the traditional formulation of the retiming problem. Even though there are some previous works attempting to integrate wire delay into retiming, they suffer from the problem of over-simplification, e.g., they do not consider the delay between a pair of adjacent registers on a wire. As the feature size of the CMOS technology continues to scale down, this kind of inaccuracy can no longer be neglected. In chapter 4, the problem of retiming with gate and wire delay are studied and discussed.

Our first approach is an extension of the MILP approach in the original retiming paper [22]. By introducing some new variables and modifying one of

the retiming constraints, we are able to extend the MILP approach to handle both gate and wire delay optimally in polynomial time. Our second approach is an improvement over the first one regarding to the dramatical speedup in runtime. In fact, this approach can give optimal solution if only wire delay is considered and can give near-optimal results when both gate and wire delays are present. Experimental results show that the second approach gives solutions that are only 0.13% larger than the optimal on average but in a much shorter runtime.

Another problem of applying retiming in practice is that the resulting solutions do not give information of how to share the registers among a net and where to put them on the layout without clock violation, because a multi-pin net is usually modeled as a branch of 2-terminal edges in the retiming graph model. To solve this problem, we have proposed an algorithm to perform post-retiming register placement that can preserve a given clock period with a controlled degree of error using as few registers as possible.

Our proposed algorithm is proved to be optimal for nets with four or fewer pins, and this type of nets comprises over 90% of the nets in a sequential circuit on average. Besides, it can handle the blockages on a chip when inserting the registers into the placement. Experimental results show that our proposed algorithm can almost always find optimal solutions for the nets of the ISCAS89 benchmark circuits.

In conclusion, we have studied and solved two problems related to retiming. One is the problem of retiming with gate and wire delay while the other problem is post-retiming register placement. Our proposed algorithms have contributed to the applicability of retiming in today's DSM designs. Future works can be focused on reducing the number of registers increased after retiming and on improving the register placement heuristics.

Bibliography

[1] S. M. Burns. Performance analysis and optimization of asynchronous circuits. In *PhD thesis, California Institute of Technology*, 1991.

[2] Chris Chu, Evangeline F. Y. Young, Dennis K. Y. Tong, and Sampath Dechu. Retiming with interconnect and gate delay. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 221–226, November 2003.

[3] P. Cochini. Concurrent flip-flop and repeater insertion for high performance integrated circuits. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 268–273, November 2002.

[4] J. Cong, L. He, K. Y. Khoo, C. K. Koh, and Z. Pan. Interconnect design for deep submicron ics. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 478–485, November 1997.

[5] J. Cong, H. Li, and C. Wu. Simultaneous circuit partitioning/clustering with retiming for performance optimization. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 460–465, June 1999.

[6] J. Cong and S. K. Lim. Physical planning with retiming. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 2–7, November 2000.

[7] J. Cong, S. K. Lim, and C. Wu. Performance driven multi-level and multiway partitioning with retiming. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 274–279, June 2000.

[8] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to algorithm. McGraw Hill, eighth edition, 1992.

[9] Ali Dasdan, Sandy S. Irani, and Rajesh K Gupta. An experimental study of minimum mean cycle algorithms. In *Techical Report 98-32, University of California, Irvine*, July 1998.

[10] Ali Dasdan, Sandy S. Irani, and Rajesh K Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 37–42, June 1999.

[11] Vivek De and Shekhar Borkar. Low power and high performance design challenges in future technologies. In *Proceedings of Great Lakes Symposium on VLSI*, pages 1–6, March 2000.

[12] Rahul B. Deokar and Sachin S. Sapatnekar. A fresh look at retiming via clock skew optimization. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 310–315, June 1995.

[13] S. Dey and S. T. Chakradhar. Retiming sequential circuits to enhance testability. In *Proceedings of 12th IEEE VLSI Test Symposium*, pages 28–33, April 1994.

[14] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithm. In *Proceedings of 25th IEEE Symposium on Foundations of Computer Science*, pages 338–346, October 1984.

[15] Joseph L. Ganley and Jeffrey S. Salowe. Optimal and approximate bottleneck steiner trees. In *Operations Reserach Letter*, pages 217–224, 1996.

[16] S. Hassoum, Charles J. Alpert, and M. Thiagarajan. Optimal buffered routing path constructions for single and multiple clock domain systems. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 247–253, November 2002.

[17] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. In *Journal of ACM*, volume 24, pages 1–13, January 1977.

[18] Kumar N. Lalgudi and Marios C. Papaefthymiou. Delay: An efficient tool for retiming with realistic delay modeling. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 304–309, June 1995.

[19] E. L. Lawler. Combinatorial optimization: Networks and matroids. In *Holt, Rinehart and Winston*, 1976.

[20] C. Leiserson and B. Saxe. Optimizing synchronous systems. In *Journal of VLSI and Computer Systems*, volume 1, pages 41–67, 1983.

[21] C. Leiserson and B. Saxe. A mixed-integer programming problem which is efficiently solvable. In *Journal of Algorithm*, volume 9, pages 114–128, 1988.

[22] C. Leiserson and B. Saxe. Retiming synchronous circuitry. In *Algorithmica*, volume 6, pages 5–35, 1991.

[23] C. Lin and H. Zhou. Retiming for wire pipelining in system-on-chip. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 215–220, November 2003.

[24] Naresh Maheshwari and Sachin S. Sapatnekar. An improved algorithm for minimum-area retiming. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 2–7, June 1997.

[25] Ingmar Neumann and Wolfgang Knuz. Placement driven retiming with a coupled edge timing model. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 95–102, November 2001.

[26] Ingmar Neumann and Wolfgang Knuz. Layout driven retiming using the coupled edge timing model. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, volume 22, pages 825–835, July 2003.

[27] R. H. J. M. Otten and R. K. Brayton. Planning for performance. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 122–127, June 1998.

[28] P. Pan, Arvind K. Karandikar, and C. L. Liu. Optimal clock period clustering for sequential circuits with retiming. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, volume 17, pages 489–498, June 1998.

[29] Marios C. Papaefthymiou. Understanding retiming through maximum average-weight cycles. In *Proceedings of ACM Symposium on Parallel Algorithms and Architectures*, pages 338–348, July 1991.

[30] M. Sarrafzadeh and C. K. Wong. Bottleneck steiner trees in the plane. In *IEEE Trans. on Computers*, volume 41, pages 370–374, March 1992.

[31] C. V. Schimpfle, S. Simon, and Josef A. Nossek. Optimal placement of registers in data paths for low power design. In *Proceedings of 1997 IEEE International Symposium on Circuits and Systems*, volume 3, pages 2160–2163, June 1997.

[32] N. Sheony and R. Rudell. Efficient implementation of retiming. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 226–233, November 1994.

[33] T. Soyata and E. G. Friedmann. Retiming with nonzero clock skew, variable register and interconnect delay. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 234–241, November 1994.

[34] Abdallah Tabbara, Robert K. Brayton, and A. Richard Newton. Retiming for dsm with area-delay trade-offs and delay constraints. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 725–730, June 1999.

[35] T. C. Tien, H. P. Su, and Y. W. Tsay. Integrating logic retiming and register placement. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 136–139, November 1998.

[36] Dennis K. Y. Tong and Evangeline F. Y. Young. Performance-driven register insertion in placement. In *Proceedings of International Symposium on Physical Design*, pages 53–60, April 2004.