Clock Routing for High Performance Microprocessor Designs

TIAN, Haitong

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

The Chinese University of Hong Kong
August 2011

Thesis/Assessment Committee:

Professor XU Qiang (Chair)

Professor YOUNG Fung Yu (Thesis Supervisor)

Professor ZHANG Sheng Yu (Committee Member)

Professor HU Jiang (External Examiner)

Abstract of thesis entitled:

Clock Routing for High Performance Microprocessor Designs

Submitted by TIAN, Haitong

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in August 2011

Clock distribution in VLSI designs is of crucial importance and it is also a major source of power dissipation of a system. Most of the chips today are based on a synchronous sequential circuit design methodology. For these circuits, a global clock signal is needed to synchronize the operations of different components across the chip. The clock signal is usually generated using an external reference and delivered to the entire chip by a clock network. Since the clock signal coordinates all the elements of the chip, designing a good clock network is very important to secure high performance. As a clock network connects to a large number of chip elements e.g, latches, flip-flops and gates, and has high switching frequency, it is also responsible to a large portion of the total power consumption of the chip. Therefore, the clock network must be carefully designed to optimize the chip performance and the power consumption. For today's high performance microprocessors, clock signals are usually distributed by a global clock grid covering the whole chip, followed by post-grid routing that connects clock loads to the clock grid. Early study [50] shows that about 18.1% of the total clock capacitance dissipation was due to this post-grid clock routing, i.e., lower mesh wires plus clock twig wires. This post-grid clock routing problem is thus an important one but not many previous works have addressed it. This "grid-to-ports" routing is critical in securing high quality of the clock signal and in reducing power consumption. In this thesis, we try to solve this problem of connecting clock ports to the clock grid through reserved tracks on multiple metal layers with delay constraint. Note that a set of routing tracks are reserved for this grid-to-ports clock wires in practice because of the conventional modular design style of high-performance microprocessors. We propose a new

expansion algorithm based on the heap data structure to solve the problem effectively. Experimental results on industrial test cases show that our algorithm can improve over the latest work on this problem [65] significantly by reducing the capacitance by 24.6% and the wire length by 23.6%. To make our approach more practical and complete, we have extended our approach to use non-tree structures to further optimize the delay. We also proposed a partitioning technique that can improve the running time by 26.1% without sacrificing the solution quality. We validate our results using hspice simulation. Finally, our approach is very efficient and for larger test cases with about two thousands ports, the running time is just in seconds.

論文摘要

芯片的時鐘佈線在集成電路設計領域有著至關重要的作用，它也是系統中的主要功耗部件之一。當今的大多數芯片都採用的同步時鐘設計工藝。 對於這些芯片，我們需要一個全局的時鐘信號來同步芯片內各個部件的工作。時鐘信號一般是由外來時鐘源產生，繼而經過時鐘網絡到達芯片的各個部件。設計高性能的時鐘樹對於保證芯片的性能有著非常重要的作用。因為時鐘網絡連接著大量的芯片原件（如鎖存器，觸發器和門電路），並且工作在極高的頻率下，它是系統中的主要功耗部件之一。因此，爲了降低功耗并確保芯片的性能，我們需要仔細的設計芯片的時鐘結構。對當今的高性能處理器來講，時鐘樹一般採用全局的時鐘線外加局部的時鐘線后的時鐘樹來提供芯片需要的時鐘信號。早期的研究表明，全局時鐘樹后的時鐘佈線所佔用的電容占到整個時鐘網絡的18.1%。所以全局時鐘樹后的時鐘佈線問題顯得額外重要。在這篇論文中，我們致力於解決有一定時延要求的高性能時鐘佈線問題。在這個問題中，我們給定了待佈線的電路原件，能夠使用的電路線位置以及全局的多時鐘源線。我們提出了線路擴張的算法來解決這個問題。實驗結果表明，和最新的算法相比，我們的算法能夠減少電容使用量24.6%，能夠減少佈線長度23.6%。爲了處理更為複雜的情況，我們還將算法擴展到了非樹結構的時鐘佈線算法來進一步降低時鐘結構的時延。我們提出了一種分割方法，能夠將沒有採納分割方法算法的時間提高26.1%，同時構建出的時鐘樹性能並沒有降低。我們用Hspice 仿真驗證了我們算法的正確性。我們的算法非常高效，對於有兩千多個待佈線原件的芯片，我們的算法運行時間在幾秒鐘之內。

Acknowledgement

First, I would like to express my deepest and most sincere gratitude to professor Evangeline F.Y. Young, my supervisor, for her continuous help, constant encouragement and instructions through my MPhil studies. Without her patient guidance and insightful comments, this thesis could not has reached its present form.

Second, I would like to express my heartfelt gratitude to Mr. Wai-Chung Tang in CSE department in CUHK. He gave me many valuable help and support when I was investigating this clock routing problem. I also want to thank Dr. C.N. Sze from IBM Austin Research Laboratory in USA. It is with his helpful advices and valuable comments that I am able to extend my work with different perspectives and methodologies.

Last but not least, I would thank all my friends for their valuable help and friendship!

This work is dedicated to my family.

Contents

Abstract i

Acknowledgement iii

1 Introduction 1

1.1 Motivations 1

1.2 Our Contributions 2

1.3 Organization of the Thesis 3

2 Background Study 4

2.1 Traditional Clock Routing Problem 4

2.2 Tree-Based Clock Routing Algorithms 5

2.2.1 Clock Routing Using H-tree 5

2.2.2 Method of Means and Medians(MMM) 6

2.2.3 Geometric Matching Algorithm (GMA) 8

2.2.4 Exact Zero-Skew Algorithm 9

2.2.5 Deferred Merge Embedding (DME) 10

2.2.6 Boundary Merging and Embedding (BME) Algorithm 14

2.2.7 Planar Clock Routing Algorithm 17

2.2.8 Useful-skew Tree Algorithm 18

2.3 Non-Tree Clock Distribution Networks 19

2.3.1 Grid (Mesh) Structure 20

2.3.2 Spine Structure 20

2.3.3 Hybrid Structure 21

2.4 Post-grid Clock Routing Problem 22

2.5 Limitations of the Previous Work 24

3 Post-Grid Clock Routing Problem **26**

3.1 Introduction 26

3.2 Problem Definition 27

3.3 Our Approach 30

3.3.1 Delay-driven Path Expansion Algorithm 31

3.3.2 Pre-processing to Connect Critical ports 34

3.3.3 Post-processing to Reduce Capacitance 36

3.4 Experimental Results 39

3.4.1 Experiment Setup 39

3.4.2 Validations of the Delay and Slew Estimation 39

3.4.3 Comparisons with the Tree Grow (TG) Approach 41

3.4.4 Lowest Achievable Delays 42

3.4.5 Simulation Results 42

4 Non-tree Based Post-Grid Clock Routing Problem **44**

4.1 Introduction 44

4.2 Handling Ports with Large Load Capacitances 46

4.2.1 Problem Ports Identification 47

4.2.2 Non-Tree Construction 47

4.2.3 Wire Link Selection 48

4.3 Path Expansion in Non-tree Algorithm 51

4.4 Limitations of the Non-tree Algorithm 51

4.5 Experimental Results 51

4.5.1 Experiment Setup 51

4.5.2 Validations of the Delay and Slew Estimation 52

4.5.3 Lowest Achievable Delays 53

4.5.4 Results on New Benchmarks 53

4.5.5 Simulation Results 55

5 Efficient Partitioning-based Extension **57**
5.1 Introduction 57
5.2 Partition-based Extension 58
5.3 Experimental Results 61
5.3.1 Experiment Setup 61
5.3.2 Running Time Improvement with Partitioning Technique 61

6 Conclusion **63**

Bibliography **65**

List of Figures

2.1 H-tree structure 6
2.2 Recursive partitioning of MMM algorithm 7
2.3 Recursive grouping of the GMA Algorithm. Solid points denote the sink nodes, and empty points denote merging points on the connection. At each level,a geometric matching algorithm is performed on the merging points at the previous level. 8
2.4 Merging Two Subtrees 11
2.5 An example of multiple merging points. For both figure a and b, any point along the segment $P_a P_b$ satisfy the zero-skew requirement. 12
2.6 Construction of merging segments between two nodes in the bottom-up phase of DME. The merging segments are shown in thick solid blue line. 13
2.7 Construction of merging segments between two segments in the bottom-up phase of DME. 13
2.8 An example of the merging region when the skew bound is non-zero. . . 15
2.9 Merging of four sinks S_1, S_2, S_3, and S_4. The shaded part $mr(v1)$ is the merging region of sinks S_1 and S_2, and $mr(v2)$ is the merging region of sinks S_3 and S_4. 16
2.10 Merging of four sinks S_1, S_2, S_3, and S_4. The darker region $mr(v3)$ is the merging region of $mr(v1)$ and $mr(v2)$. 16
2.11 Planar routing of three sinks 18
2.12 Clock mesh structure. The figure is from [14]. 20
2.13 Spine structure of a microprocessor. 21
2.14 (a) A routing graph example of tree growing (TG) algorithm. (b) The final solution contains five trees, T_1, ... , T_5. 23

2.15 (a) Routing graph with sources s_1 and s_2, ports p_1, p_2 and p_3, and via nodes c_1 and c_2. The numbers near the edges denote the wire capacitances. (b) Routing solution of [65]. (c) A better topology with a 36% reduction in wire capacitance. 24

3.1 Typical clock distribution of microprocessors 27
3.2 Post-grid clock network distribution 28
3.3 Post-grid clock routing problem 29
3.4 An overall flow of our approach 32
3.5 Comparisons between computed and simulated delays in a tree structure 40
3.6 Comparisons between computed and simulated slews in a tree structure 40

4.1 General post-grid clock routing algorithm 45
4.2 Example of adding links 48
4.3 Delay comparisons (Non-tree structure) 52
4.4 Slew comparisons (Non-tree structure) 53

5.1 A sample circuit divided into four regions 58
5.2 An overall flow of our partition-based algorithm 59
5.3 An example of partitioning 60

List of Tables

3.1 Comparisons with TG (with topology refinement only for post-processing) 37
3.2 Comparisons with TG (using all techniques) 38
3.3 Lowest achievable delays 42
3.4 Simulation results for tree 43

4.1 Lowest achievable delays (with non-tree technique) 54
4.2 Non-tree algorithm 55
4.3 Simulation results for non-tree 56

5.1 Running time comparisons 62

Chapter 1

Introduction

1.1 Motivations

For most of chips today, data transfer between different function units is synchronized by a single global clock signal. Synchronizing the clock signals is one of the most important tasks in designing a high-performance microprocessor. All the functional units within a chip should be connected to the clock source very carefully to satisfy some delay, skew and/or slew constraints, as those factors directly affect the operating frequency and the performance of the chip. There have been many clock routing algorithms in the literature, such as the H-tree algorithm [6,26,27], the Method of Means and Medians (MMM) algorithm [32], the Geometric Matching Algorithm (GMA) [19, 33], the Exact Zero Skew algorithm [71], the Deferred Merge Embedding(DME) algorithm [10, 38], the Bounded Skew Clock Routing algorithm [18, 20, 35], the Planar-DME algorithm [36,81], the UST/DME algorithm [70], the clock routing algorithms considering process variation [41, 44, 75] and the non-tree clock routing algorithms [40, 52, 77]. From the skew minimization perspective, the above algorithms can be classified into four categories according to the way the skew is handled: (i) clock skew minimization design [6, 19, 26, 27, 32, 33], (ii) exact zero skew design [10, 36, 38, 71, 81], (iii) bounded clock skew design [18, 20, 35, 70], and (iv) process variation-aware clock design [40, 41, 44, 48, 49, 52, 77].

In today's high performance systems, clock signals are usually distributed through a global clock grid [5,39,47,50,59,60,64,65], followed by post-grid routing that connects clock loads to the grid. Early studies show that most of the clock power dissipation

was due to three major categories of capacitances – (i) clock load, (ii) clock twig and clock mesh wires, and (iii) clock grid buffers. The second category of post-grid clock routing wires, i.e., lower mesh wires and clock twig wires, comprises 18.1% of the total capacitance [50]. This post-grid clock routing problem is thus a very important one, although not many previous works have addressed it.

Due to the high complexity of microprocessor design, the clock distribution network is usually synthesized and tuned at the same time when different design teams are working on their logic modules. In this case, the clock distribution between the clock grid and the block-level clock ports is subject to conflict of routing resources for data signals. To resolve this conflict and to facilitate simultaneous work between different design teams, a subset of routing tracks have to be reserved for this post-grid clock routing. As a result, this post-grid clock routing problem assumes a given set of reserved tracks, forming a virtual grid structure. The quality of this routing step is of significant importance as it will directly affect the total power consumption, the clock skews and slews at the input of the ports and finally the quality of the whole chip. These provide motivations to solve this multi-source multi-port post-grid clock routing problem with an objective to minimize the interconnect capacitance while meeting a given delay constraint. Traditionally, this step is done manually and iteratively to satisfy the delay/slew constraints, resulting in a long time to market, especially when the problem size has increased to thousands of clock ports in the layout region. This also motivates the research of a fast algorithm to resolve this clock routing problem effectively.

1.2 Our Contributions

Given the importance of clock network in microprocessor systems, it is crucial to design a good clock distribution scheme in order to ensure a low power consumption and a high quality performance. The contributions of this research [69] can be summarized as follows:

- We have devised an efficient algorithm for the post-grid clock routing problem that can satisfy user given delay bound while minimizing the total wire capacitance. In our problem, the clock skew is optimized by minimizing the maximum delay,

which is an upper bound of the skew. This delay bound is set to be very stringent, e.g., within 5ps, which is in reality the limit for the clock skew. We compared our approach with the previous work [65] and show that compared with [65] with the same delay constraint, our approach can reduce wire capacitance and wire length by 24.6% and 23.6%, respectively.

- We have extended our approach to use non-tree structures in our algorithm, which makes our approach more complete and practical. Our non-tree based clock routing algorithm can satisfy more stringent delay constraints compared with the tree-based approach.
- We have proposed a partitioning-based technique to reduce the running time of our algorithm. We can achieve a 26.1% running time improvement after applying this technique without sacrificing the solution quality.
- We have simulated our clock network using HSPICE and the simulation results confirm the effectiveness and correctness of our approach.

Our algorithm can be applied to high performance microprocessor designs in the $45nm$ technology, and it can also be extended to applications for ASICs with hybrid clock structures.

1.3 Organization of the Thesis

In the following, we will first give a preliminary overview in Chapter 2 on different clock routing algorithms in the literature. Problem definition, motivations and our routing algorithm will be presented in Chapter 3. An non-tree based routing algorithm will be detailed in Chapter 4. Partitioning-based acceleration technique will be introduced in Chapter 5, followed by a conclusion in Chapter 6.

□ **End of chapter.**

Chapter 2

Background Study

2.1 Traditional Clock Routing Problem

Clock routing has been extensively studied in the past. Clock distribution schemes can be classified into two categories: the global networks and the local networks. A survey of different clock construction algorithms is detailed in [11,66]. Tree structure is widely used in global clock networks since it uses less routing resources, consumes less energy and is simpler to implement and simulate [4, 47]. At the same time, mesh structure is proposed to tackle the challenges introduced by process variation, especially when the industry advances into the very deep sub-micron era. Top level trees followed by a mesh structure have been used in some practical microprocessor designs, and improved local clock skews under process variation compared with a pure tree topology [4] are resulted obtained. Tree and mesh structures are also used for the local clock construction in the literature. Basic algorithms include the H-tree algorithm [6, 26, 27], the Method of Means and Medians (MMM) [32], the Geometric Matching Algorithm (GMA) [19,33], the Exact Zero Skew Algorithm [71], the Deferred Merge Embedding(DME) [10, 38], the Bounded Skew Clock Routing algorithm [18, 20, 35], the Planar-DME algorithm [36, 81], the UST/DME algorithm [70] and process variation-aware clock routing algorithms [41, 44, 75]. These basic clock structures can be further extended to hybrid clock distribution schemes by inserting crosslinks [45, 52–55, 74, 79] or using a global mesh [14, 73, 77] to reduce clock skews incurred by process variation.

Capacitance usage and skews are the two major issues traditional clock routing algorithms target at. The purpose of reducing the total capacitance usage is quite straightfor-

ward. We can reduce the power consumption and relieve the heat problem by reducing the chip capacitance usage. According to some previous studies [15, 31, 51], the power usage of a microprocessor can be classified into two categories: dynamic power consumption and leakage power consumption. While the leakage power consumption of a circuit can be estimated as a constant β, the dynamic power is closely related to the capacitance usage of the chip and can be calculated as follows [51]:

$$P(s) = C_{eff} V_{dd}^2 s \tag{2.1}$$

where $s = k\frac{(V_{dd}-V_t)^2}{V_{dd}}$, and C_{eff}, V_t, V_{dd}, and k denote the effective switch capacitance, the threshold voltage, the supply voltage, and a hardware-design-specific constant, respectively. By reducing the total capacitance usage, we can effectively reduce C_{eff} and thus reduce the power consumption of the whole chip.

Synchronous clock strategies continue to be the dominant clock distribution schemes for microprocessors nowadays [5, 39, 58, 59, 64, 65]. We need to control the clock arriving times from the clock source to different units in order to ensure a proper functioning of the chip. The clock skews refer to the clock arrival time difference between two points in the chip. It could be caused by asymmetric routes to different functional units, different interconnect parameters, different threshold voltages or process variation [32]. As frequency increases, the same skew value will correspond to a larger percentage of one clock period and the chip performance will be more adversely affected. This will cause serious problems when designing high performance microprocessors. Uncertainties of those arrival times, especially between nearby points, can limit clock frequency or even lead to functional errors [58]. Therefore, reducing clock skews is very important in constructing a robust clock network.

2.2 Tree-Based Clock Routing Algorithms

2.2.1 Clock Routing Using H-tree

H-tree structure is used to construct balanced and low skew clock network by maintaining the same length from the clock source to all the clocking elements on the chip [6, 26, 27]. We can recursively construct a H-tree structure from the sink nodes

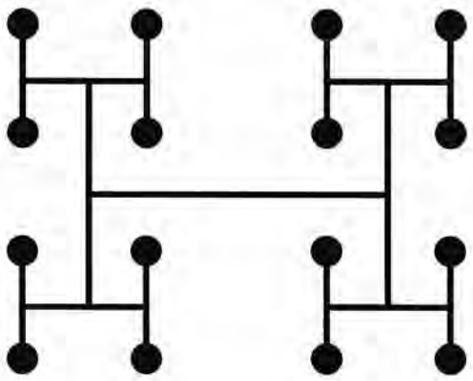


Figure 2.1: H-tree structure

of the chip in a bottom-up style. Ideally, if all the sink nodes bear the same capacitance and there is no process variation, a H-tree constructed as described above becomes a zero-skew tree. H-tree structures can significantly reduce clock skew when all the clock sinks bear the same capacitance and are placed in a symmetric array. Fig. 2.1 shows a simple example of H-tree with 16 sinks. In general the sinks are randomly distributed all over the chip, and bear different capacitances. In such circumstances, using H-tree can yield a clock structure with significantly large clock skews. So for general clock routing problems, more complicated routing algorithms are desired.

2.2.2 Method of Means and Medians(MMM)

The Method of Means and Medians(MMM) [32] algorithm is a generalization of the H-tree algorithm. The MMM algorithm is conceptually simple. Let $S = \{s_1, s_2, ..., s_n\}$ be a set of points representing the clock sinks on a two-dimensional plane. The coordinates of each s_i are denoted as (x_i, y_i). Define

$$x_c(S) = \frac{\sum_{i=1}^{n} x_i}{n} \tag{2.2}$$

$$y_c(S) = \frac{\sum_{i=1}^{n} y_i}{n} \tag{2.3}$$

$(x_c(S), y_c(S))$ represents the center of mass of the set of points. We can divide the set S into two sets S_L and S_R according to the median's x coordinate. Similarly, we can

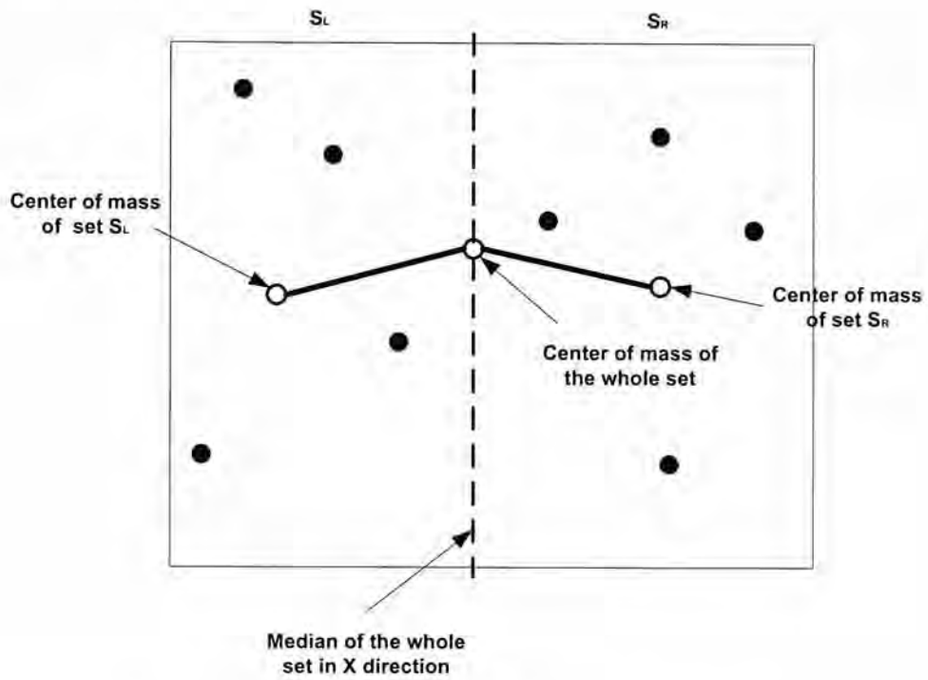


Figure 2.2: Recursive partitioning of MMM algorithm

also divide the set S into two sets S_B and S_T according to the median's y coordinate. If the number of sinks is even, the number of sinks in the two subsets will be equal to each other. Otherwise, they will be differ by one. In fact, $||S_L| - |S_R|| \leq 1$. After that, connections are made from the center of mass of the set S to the centers of mass of the two subsets, ensuring that there is no length skew at the current level of the clock tree. One interesting property of this algorithm is that given a set of points $S = \{s_1, s_2, ..., s_n\}$ when n is an even integer,

$$\begin{aligned} &|x_c(S) - x_c(S_L(S))| + |y_c(S) - y_c(S_L(S))| = \\ &|x_c(S) - x_c(S_R(S))| + |y_c(S) - y_c(S_R(S))| \end{aligned} \tag{2.4}$$

A similar result holds for the sets S_T and S_B, where S_T and S_B are the two sets obtained by dividing S according to the median y coordinate. This property tells us that whenever we divide a set into two subsets with equal number of sinks, the wire lengths from the center of mass to the center of mass of the two subsets are always equal to each other. Thus, at every level of the clock tree, the algorithm can achieve locally zero-skew result (but it is not zero-skew globally.). When choosing the partitioning direction, the algorithm incorporates a delay equalization look-ahead technique. Both x-then-y and

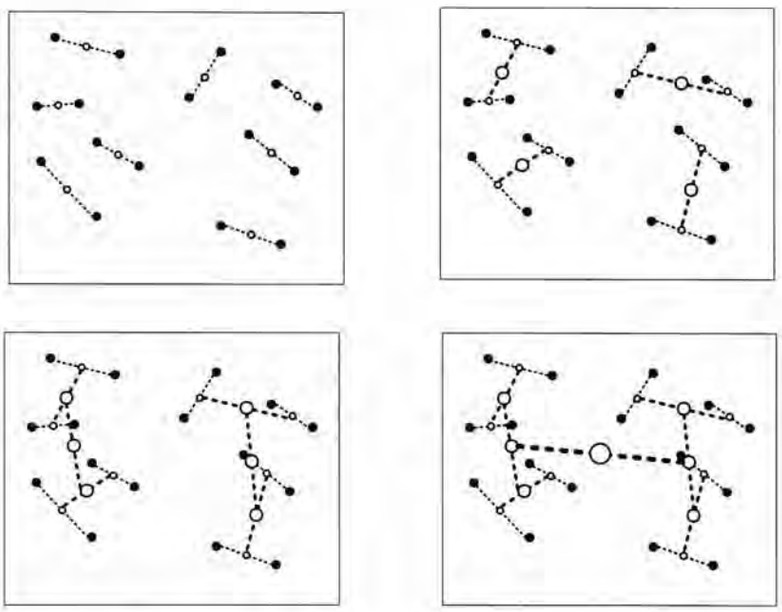


Figure 2.3: Recursive grouping of the GMA Algorithm. Solid points denote the sink nodes, and empty points denote merging points on the connection. At each level,a geometric matching algorithm is performed on the merging points at the previous level.

y-then-x directions are tried, and the one minimizing the clock skew at the end points is selected. The above techniques will be used to recursively construct a clock tree until there is only one sink left in each subset. The time complexity of MMM is $O(nlogn)$, where n is the number of sinks to be routed. Fig. 2.2 shows an example of the MMM algorithm. In this simple example, the eight sink points are divided into two sets S_L and S_R according the median of the whole set in the X direction. The center of mass of the set S_L and the center of mass of the set S_R are connected to the center of mass of the whole set.

2.2.3 Geometric Matching Algorithm (GMA)

Aiming at the same problem as the Method of Means and Medians(MMM) algorithm, geometric matching algorithm (GMA) [19, 33] uses a bottom-up recursive approach to construct a binary clock tree. A geometric matching for k points refers to a set of $\frac{k}{2}$ segments, with no two segments connecting to the same point. At the very beginning, each sink is a subtree itself. The geometric matching algorithm (GMA) will merge these

trees by connecting them in pairs and finding a merging point on the connection in such a way to minimize the maximum difference in path lengths from the merging point to the leaves of the subtrees. Note that when there are only two sinks in the subtree, the optimal merging point is the midpoint of the connection, so that the clock signal will have zero skew[1]. At the next level, merging point is set to be the point that minimizes the maximum path length skew from it to the leaves of the two subtrees being connected. The resulting new subtrees will be recursively merged until a final clock network is constructed for the whole chip. Fig. 2.3 shows an example of the GMA algorithm when there are 16 sinks in the plane. Although the GMA algorithm produces better results compared with the MMM approach according to the experimental results of [33], GMA still cannot guarantee a zero-skew clock structure.

2.2.4 Exact Zero-Skew Algorithm

The underlying assumption behind both the geometric matching algorithm(GMA) and the Method of Means and Medians (MMM) algorithm is that the delay from the source to a sink is linearly proportional to the path length. They mainly focus on balancing the path lengths from the source to different sinks on the chip. An exact zero-skew algorithm based on a more accurate and desirable Elmore delay model is proposed by Tsay [71] to construct exact zero-skew clock network. The algorithm adopts a hierarchical method for computing the Elmore delays in a bottom-up fashion. It assumes that a tree topology is given beforehand, and it will recursively connect pairs of nodes and tune the new merging points in a bottom-up fashion. The finally constructed tree has exact zero-skew under the Elmore delay model, which has been widely used in practice to estimate delay due to its simplicity and high fidelity.

The algorithm is a recursive one proceeding from the leaves to the root. Assume that at a certain stage, a number of exact zero-skew subtrees have been constructed (this is true at the beginning when there are only sink nodes). To connect two zero-skew subtrees and to ensure exact zero-skew of the combined tree, the algorithm finds a good merging point on the connection of the two subtrees, such that the delays from the new

[1]The skew here refers to path length skew, which is defined to be the maximum difference of the path lengths in the tree from the root to any two leaves.

merging point to the leaf nodes of the two subtrees are the same. Let's consider an example with two subtrees T_1 and T_2 as shown in Fig 2.4. The interconnection of the two subtrees is divided into two segments, W_1 and W_2. Each wire is represented by a π-model as shown in the figure. Let t_1 and t_2 denote the internal delays (i.e.,delay from the root to the leaves of the subtree) of subtrees T_1 and T_2. If we want to merge these two subtrees to form a new zero-skew tree, it requires that

$$r_1(c_1/2 + C_1) + t_1 = r_2(c_2/2 + C_2) + t_2 \tag{2.5}$$

where C_1 denote the total capacitance of subtree T_1, C_2 denote the total capacitance of subtree T_2, r_1 and c_1 denote the total wire resistance and capacitance of wire segment W_1, and r_2 and c_2 denote the total wire resistance and capacitance of wire segment W_2 respectively. Assume that the total wire length connecting the two subtrees is l, the length of wire segment W_1 is xl and length of wire segment W_2 is $(1-x)l$. Denote α as the resistance per unit length of the wire and β as the capacitance per unit length of the wire. We have $r = \alpha l$, $r_1 = \alpha x l$, $r_2 = \alpha(1-x)l$, $c = \beta l$, $c_1 = \beta x l$, and $c_2 = \beta(1-x)l$ respectively. After solving equation 2.5, we will get

$$x = \frac{(t_2 - t_1) + (\alpha l(C_2 + \frac{\beta l}{2}))}{\alpha l(\beta l + C_1 + C_2)} \tag{2.6}$$

Thus, the exact location of the merging point can be determined when $0 \leq x \leq 1$. If $x \leq 0$ or $x \geq 1$, detouring, e.g., snaking of wires, is needed to ensure that the constructed clock network is an exact zero-skew tree. If the two subtrees are too much out of balance and detouring will significantly degenerate the routability of the chip, adding buffers, delay lines or capacitance terminators can be used to handle such extreme cases. To minimize the total wire length, pattern routing, e.g., one-bend connection, is used. The pattern that gives shorter wire length in the next higher level will be selected.

However, the algorithm assumes that the tree topology is given beforehand. To construct a zero-skew tree, we can first pair up all sink nodes using MMM or GMA before using the exact zero skew approach to construct the tree.

2.2.5 Deferred Merge Embedding (DME)

The Deferred Merge Embedding (DME) algorithm proposed in [10, 25, 38] makes use of the fact that for each pair of sink nodes, there can be *multiple choices* in selecting the

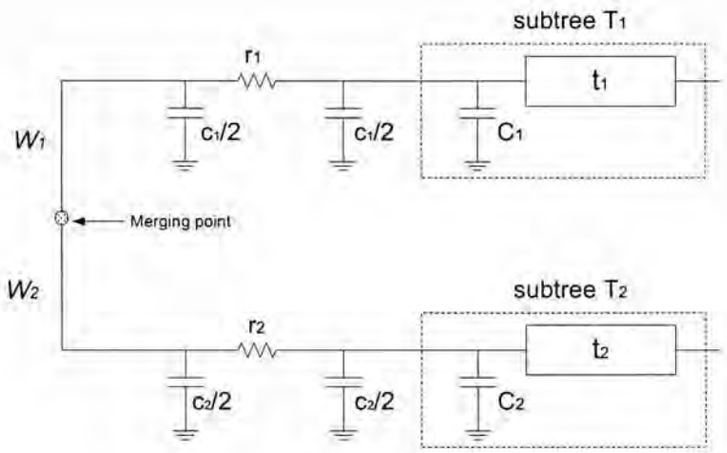


Figure 2.4: Merging Two Subtrees

merging point in order to construct a better zero-skew clock tree. An example is shown in Fig. 2.5 to illustrate that point. Consider Fig. 2.5(a). Let P_1 and P_2 be the merging points of two previously constructed subtrees T_1 and T_2 respectively, and now we are trying to connect T_1 and T_2 together. After considering the delay from P_1 and P_2 to the leaves in T_1 and T_2 respectively, we can compute the required distances from the new merging point x to P_1 and P_2 respectively. Let the required distance from x to P_1 be 5 units and that from x to P_2 be 4 units respectively. We can identify two potential merging points, which are P_a and P_b respectively. However, one can easily see that any point on the segment P_aP_b will satisfy the exact zero skew requirement, as any point on the segment P_aP_b will have a distance of 4 units from node P_2 and a distance of 5 units from node P_1. Similar result holds for the example in Fig. 2.5(b). The Deferred Merge Embedding (DME) algorithm is proposed to construct better zero-skew clock networks by fully exploring all the merging points. We use a *merging segment* to represent all possible placement locations for each merging point. The DME algorithm consists of two phases. First, a bottom-up phase to find all the merging segments. Second, a top-down embedding phase fixing the location of each merging point.

Denote $ms(v)$ as the merging segment that represents all possible placement locations of a merging point v. We use a simple example to illustrate the bottom-up phase of the DME algorithm. Suppose that we are now connecting four sinks, A, B, C, and D using the DME algorithm as shown in Fig. 2.6. Consider the merging segment $ms(E)$

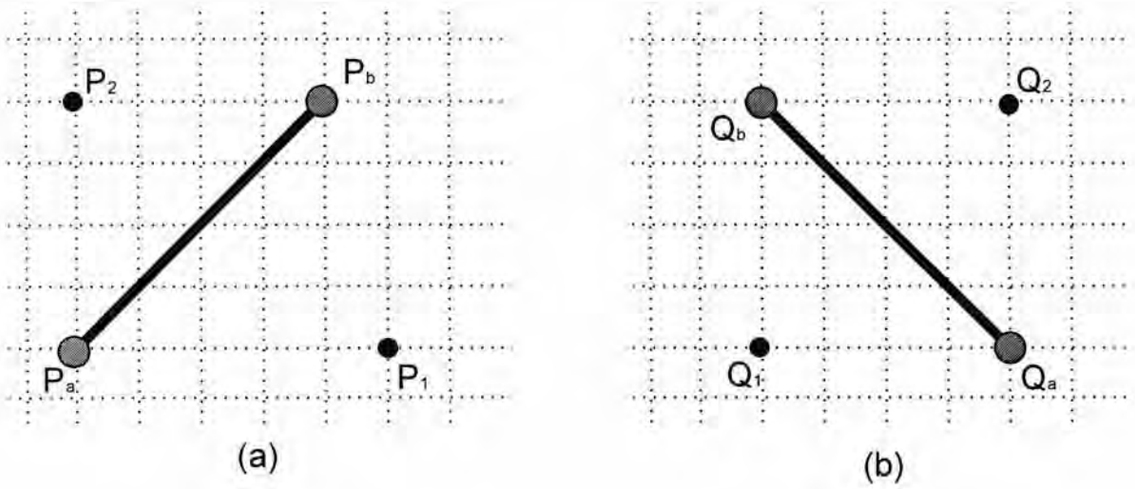


Figure 2.5: An example of multiple merging points. For both figure a and b, any point along the segment P_aP_b satisfy the zero-skew requirement.

of node A and B. According to some delay models, i.e., the Elmore delay model, we can first compute the distance d_{AE} from A to the merging at E and the distance d_{BE} from B to E such that the skew is zero and $d_{AE}+d_{BE}$ is the shortest Manhattan distance between A and B. Then $ms(E)$ is set to be the set of all points at a distance d_{AE} from A and at a distance d_{BE} from B. The merging segment $ms(F)$ of sinks C and D can be found similarly. Next, we will merge the two merging segments, $ms(E)$ and $ms(F)$, to form a new merging segment $ms(G)$ as shown in Fig. 2.7. The distance between $ms(E)$ and $ms(F)$ is set to be the minimum distance between any point on $ms(E)$ and any point on $ms(F)$. After determining the distance between $ms(E)$ and $ms(F)$, d_{EG} and d_{FG} can be calculated similarly. Finally, we set $ms(G)$ to be the set of all points at a distance of d_{EG} from some points on $ms(E)$ and at a distance d_{FG} from some points on $ms(F)$. Each merging segment can be found in constant time. The whole bottom-up phase requires linear time.

After all the merging segments are determined, a top-down phase will be invoked to determine the exact locations of all the merging points. For merging point v, its location will be selected as follows:

- If v is the root node, any point on the corresponding merging segment can be chosen.

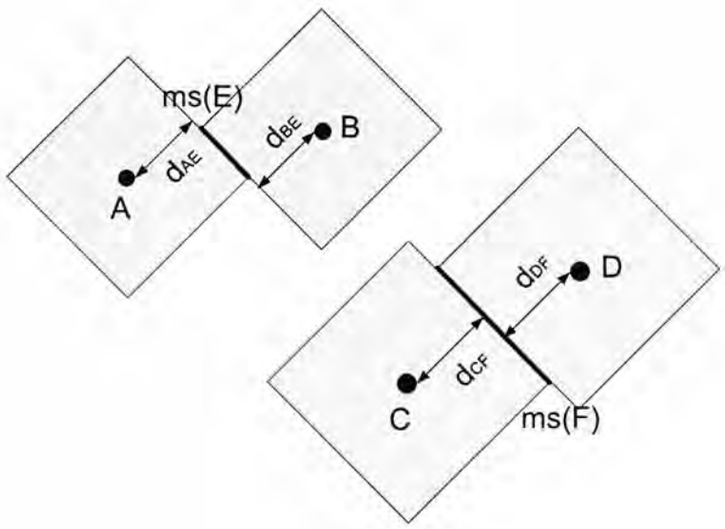


Figure 2.6: Construction of merging segments between two nodes in the bottom-up phase of DME. The merging segments are shown in thick solid blue line.

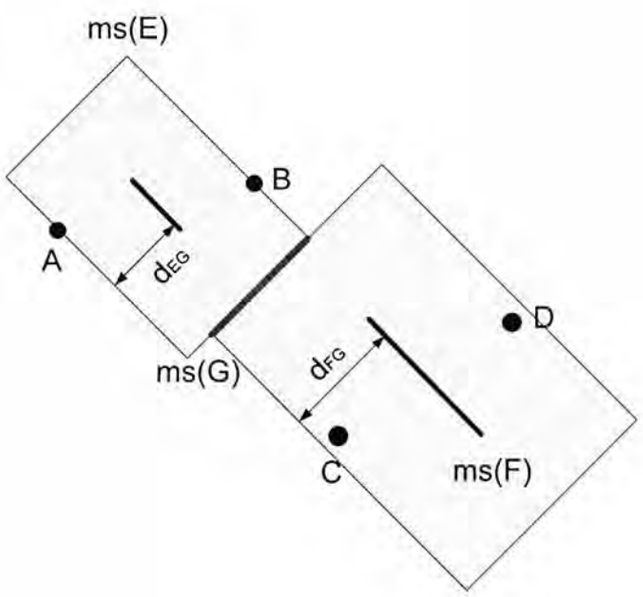


Figure 2.7: Construction of merging segments between two segments in the bottom-up phase of DME.

- if v is an internal node other than the root, v can be embedded at any point in the corresponding merging segment that is at distance d_{vp} (determined in the bottom-up phase) from p, where p is the parent of v.

The top-down phase also takes linear time. Therefore, the overall DME algorithm is a linear time algorithm. The DME algorithm can achieve optimal wire length for linear delay models. It could be further extended to handle problems with general skew constraints.

2.2.6 Boundary Merging and Embedding (BME) Algorithm

Some generalizations of the DME algorithms have been proposed, such as the bounded-skew clock and steiner routing algorithm (BST/DME) in [20, 35] and the Boundary Merging and Embedding (BME) algorithm in [18] to solve the bounded skew clock routing problem. "Exact zero skew" can typically be obtained at the expense of increased wiring area and higher power consumption. However, in practice a chip can still function correctly within a given skew tolerance, and exact zero skew is sometimes not a realistic design requirement [34]. For such cases, constructing a clock tree satisfying a given skew bound is of interest. The *Minimum-Cost Bounded Skew Routing Tree (BST) Problem* [18] is defined as follows: given a set $S = \{s_1, ..., s_n\} \subset \Re^2$ of sink locations and a skew bound B, find a routing topology G and a minimum-cost clock tree $T_G(S)$ that satisfies $skew(T_G(S)) \leq B$. In [20,35], the bounded skew tree problem is solved under the path length delay model, while in [18] a more accurate Elmore delay model in considered. In the following, we mainly introduce the Boundary Merging and Embedding (BME) algorithm proposed in [18].

Let $mr(v)$ denote the merging region of two subtrees being merged, which contains all possible locations of the merging point that satisfy the skew requirement. We first illustrate the point that when the required skew is non-zero, the "merging segment" in the DME algorithm becomes a merging region. Consider the example shown in Fig. 2.8, S_1 and S_2 denote the sinks to be merged. When the skew is zero, the merging segment will be P_1P_2 as shown in the figure. Note that P_1P_2 is indeed the merging segment used in the DME algorithm. When we have a non-zero skew bound, any node within the shaded region $mr(v)$ in Fig. 2.8 can be the merging point of the two sinks, as any node

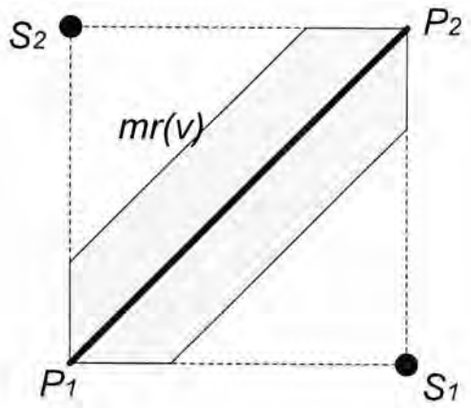


Figure 2.8: An example of the merging region when the skew bound is non-zero.

within $mr(v)$ connecting S_1 and S_2 will have a skew within the required skew bound. Thus, the merging segment becomes a merging region when we have non-zero skew requirement.

The BME algorithm incorporates two phases: a bottom-up phase finding all merging regions and a top-down phase fixing the location for each merging point within the merging regions. We use a simple example to illustrate how the BME algorithm works. Suppose we are now connecting the four sinks, S_1, S_2, S_3 and S_4 as shown in Fig. 2.9. Assume that the required skew bound B is non-zero, we can compute the merging region $mr(v1)$ for sinks S_1 and S_2 according to the technique proposed in [18]. Similarly, we can compute the merging region $mr(v2)$ for sink S_3 and S_4. Note that all points within the merging region satisfy the skew bound B. In the next step, we will merge the two merging regions $mr(v1)$ and $mr(v2)$. We first determine the closest boundary segment between the two merging regions $mr(v1)$ and $mr(v2)$, which are l_a and l_b in Fig. 2.10. Based on the locations of l_a and l_b, we can compute the merging region $mr(v3)$, the darker region, in Fig. 2.10. The bottom-up phase is a recursive process. After determining all the merging regions, a top-down phase, which is similar to the original DME algorithm, is invoked to find the exact locations of all the merging points. In the BME algorithm, only boundaries of the current merging regions are considered when computing the next level merging regions. To overcome the drawback of the BME algorithm, the author in [18] also proposed an Interior Merging and Embedding (IME) approach utilizing the internal regions as the merging points to fully exploit the skew

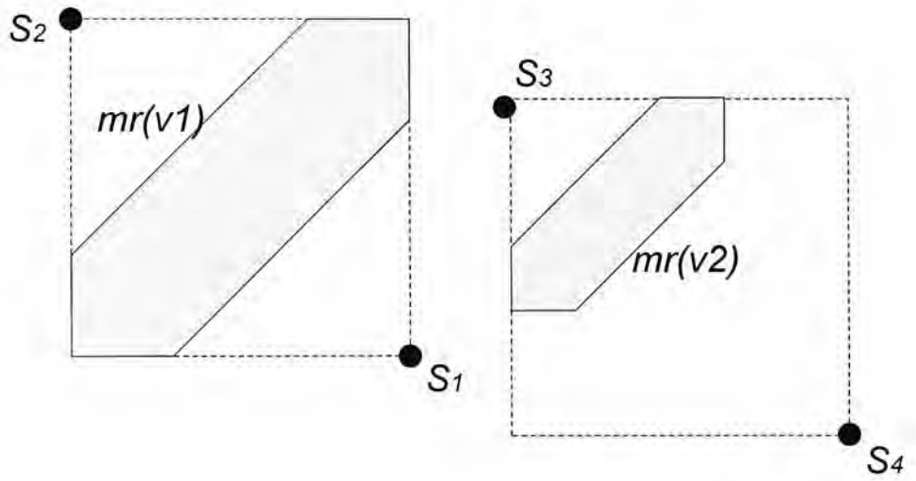


Figure 2.9: Merging of four sinks S_1, S_2, S_3, and S_4. The shaded part $mr(v1)$ is the merging region of sinks S_1 and S_2, and $mr(v2)$ is the merging region of sinks S_3 and S_4.

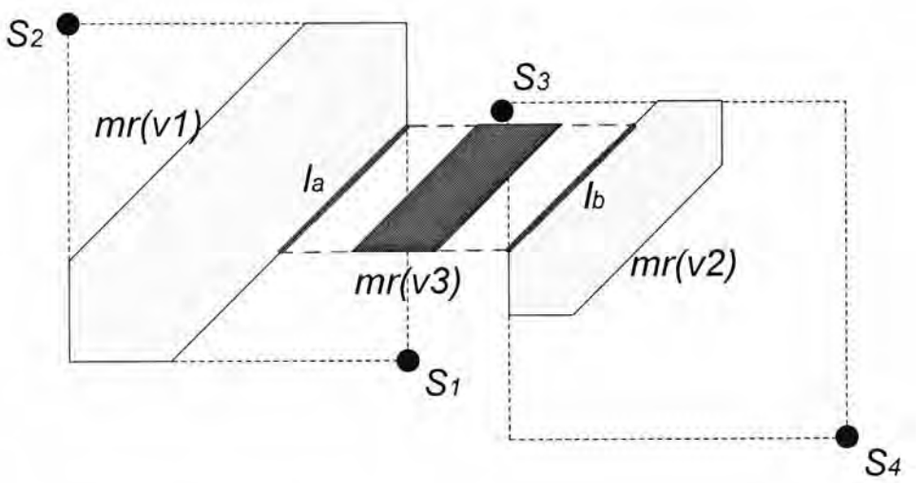


Figure 2.10: Merging of four sinks S_1, S_2, S_3, and S_4. The darker region $mr(v3)$ is the merging region of $mr(v1)$ and $mr(v2)$.

bound. Better experimental results are reported in [18] compared with the results of the BME approach.

2.2.7 Planar Clock Routing Algorithm

Routing of planar clock trees was studied in [36, 81]. The *Planar Equal Path Length Steiner Tree Problem* can be formally defined as follows: given a source point and a set of sink points, find a planar Steiner tree T with minimum total cost such that the lengths of the paths from the source point to all the sink points are exactly the same. Planar intuitively means that we can draw the trees on a plane without edge crossing. In [81], the authors proposed an algorithm to construct a planar clock tree which can be embedded on a single metal layer. In this clock tree construction algorithm, the length of the paths from the clock source to the clock terminals are all the same. The algorithm also guarantees that the path length from the source to the clock terminals is minimum. Let T_i denote the partial tree being constructed in which the first i sinks are connected. The algorithm works as follows. At the very beginning, the sink that has the largest Manhattan distance to the source will be selected and connected to the source. This will generate the first partial tree T_1. All the sinks are classified into two types by this partial tree: (i) free sinks which are the sinks that have not been connected to the partial tree, and (ii) connected sinks that have been added to the tree. In the following steps, a free sink will be selected and connected to a branch of the tree maintaining the same length from the source to the sink. Assume that the current partial tree has already included i sinks, and we now want to connect it with one more sink. The point on T_i that has the same Manhattan distance to a free sink and to a connected sink in tree T_i is called a balance point. For a particular sink, it may have several balance points on tree T_i. The balance point with the minimum Manhattan distance will be chosen. The order in which the sinks being processed will affect the final solution. In the algorithm, the following two rules are used to select the balance point and to determine the order of processing the sinks:

- Min-rule: Always connect a free sink to the balance point which has the minimum Manhattan distance to the free sink, called *minimal balance point*.

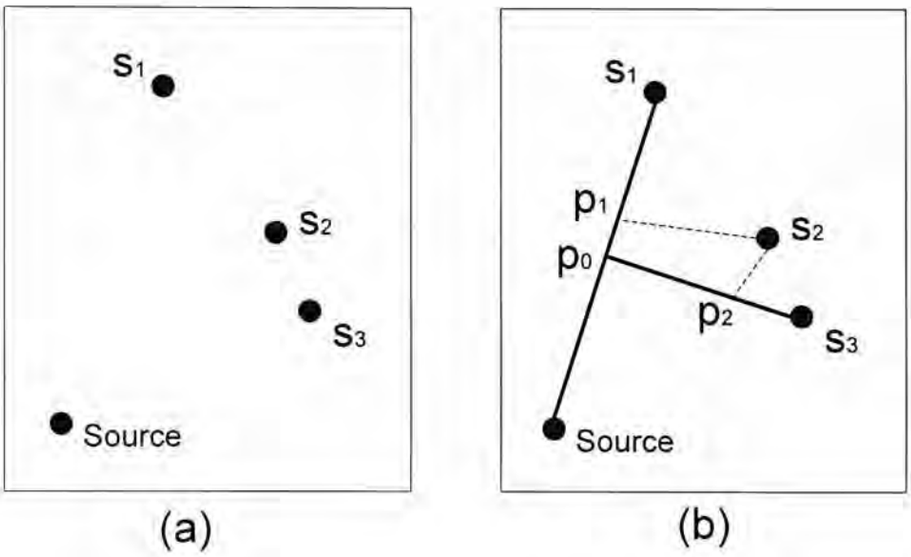


Figure 2.11: Planar routing of three sinks

- Max-rule: At each stage, the free sink that has the largest minimal balance point will be selected.

Fig 2.11 shows an example with three sink nodes. For sink s_3, its balance point is p_0. For sink s_2, there are two balance points p_1 and p_2, and the one with the minimum Manhattan distance, i.e., p_2, will be selected.

The partial tree under construction will divide the free sinks which are not connected yet into several clusters by its branches. Parallel processing is applied to speed up the method by applying the algorithm on different clusters of free sinks simultaneously. Since the clock tree solutions given by other DME algorithms cannot be easily embedded on a layout plane, this planar zero-skew clock routing algorithm can be applied if we want to route on one layer and eliminate the usage of vias.

2.2.8 Useful-skew Tree Algorithm

The underlying assumption behind the Boundary Merging and Embedding (BME) algorithm is that for all pairs of sinks, we have the same global skew range B. However, in practice we may have different skew requirements among different sink pairs and we may even be able to make use of such differences for further optimization. An useful-skew tree (UST/DME) algorithm is proposed in [70] to deal with the problem that dif-

ferent skew requirements exist among different sink pairs. The global skew bound B is used at all levels when we merge the subtrees in the BME algorithm. In the useful-skew algorithm, B will be changed. The UST/DME approach also incorporates two phases: a bottom-up phase to construct a binary tree of merging regions (or segments), and a top-down phase to determine the exact locations of the merging points. In the bottom-up phase, the skew requirements between different node pairs (they could be the sinks or the roots of subtrees.) are formulated as a constraint graph $G(V, E)$, in which the vertices V correspond the nodes and the edges in E denote the skew constraint between the nodes. At the very beginning, a subtree contains the sinks only. We can merge two nodes using a similar approach as the BST/DME algorithm, except that the skew bound B is now calculated using the constraint graph $G(V, E)$ we have previously constructed. As we are incrementally merging the subtrees, new trees are generated and the constraint graph will be updated accordingly. Finally, we can construct a tree of merging segments (or merging regions) as in the BST/DME algorithm. After that, the top-down phase which is the same with that used in [18,38] will be invoked to locate the exact locations of the merging points. According to the experimental results in [18,38], the UST/DME approach achieves better results in terms of wire length compared with the BST/DME algorithm.

2.3 Non-Tree Clock Distribution Networks

All the above algorithms focus on how to a clock tree with small or zero skew can be constructed. In practice, the clock network constructed using the above algorithms still cannot guarantee to meet the delay/slew requirements of the circuit. The undesirable process variation can seriously affect the timing of the circuit. According to the study of [43], interconnection variation alone can cause up to 25% change on clock skew. Those algorithms without considering process variation can hardly satisfy the stringent delay/skew constraints in real designs today. Tree structures have been known to be very sensitive to process variation, as for each element on the circuit, there is only one path delivering the clock signal. Mesh structures are known to be more robust to process variation due to its inherent interconnect redundancies. To combine the virtues of different clock topologies and to combat the undesirable effects due to process variation,

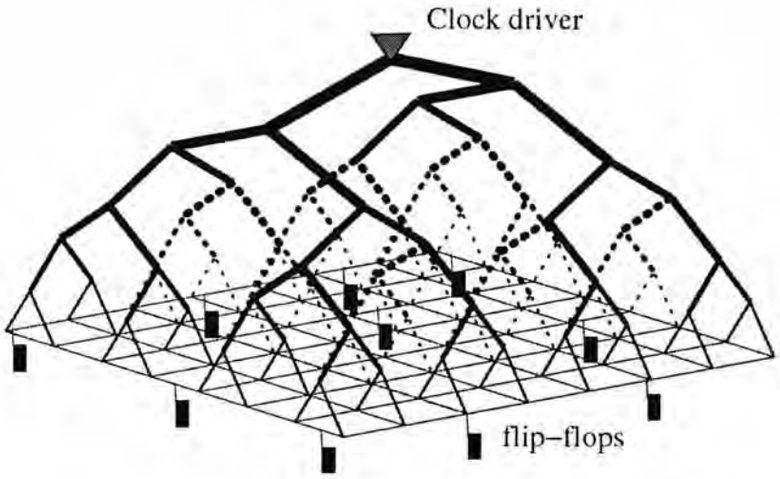


Figure 2.12: Clock mesh structure. The figure is from [14].

extensive research works have been conducted on non-tree clock distribution networks. In the following, we mainly introduce three categories of non-tree clock distribution schemes.

2.3.1 Grid (Mesh) Structure

Grid (or mesh) has been used in the clock networks in some practical chip designs [5,7,23,29,57,60] and has been widely studied in the literature [8,14,17,60,72,77,80]. Clock grid is very effective in reducing local clock skew, as two points can be directly connected. The inherent redundancy in a mesh structure smooths out undesirable variations between signal nodes spatially distributed over a chip. Some analysis on the mesh structure are conducted in [14]. Unlike the tree structures, clock mesh is quite robust and insensitive to placement details, which makes chip design easier. The main concern over clock grid is the excessive wire resources it uses. This will inevitably lead to a significant increase in power consumption of the chip. A mesh structure is shown in Fig. 2.12.

2.3.2 Spine Structure

The spine structure proposed in [42] also can provide a stable clock signal, and has been used in a few practical Intel microprocessor designs [39, 62, 63]. Each spine can

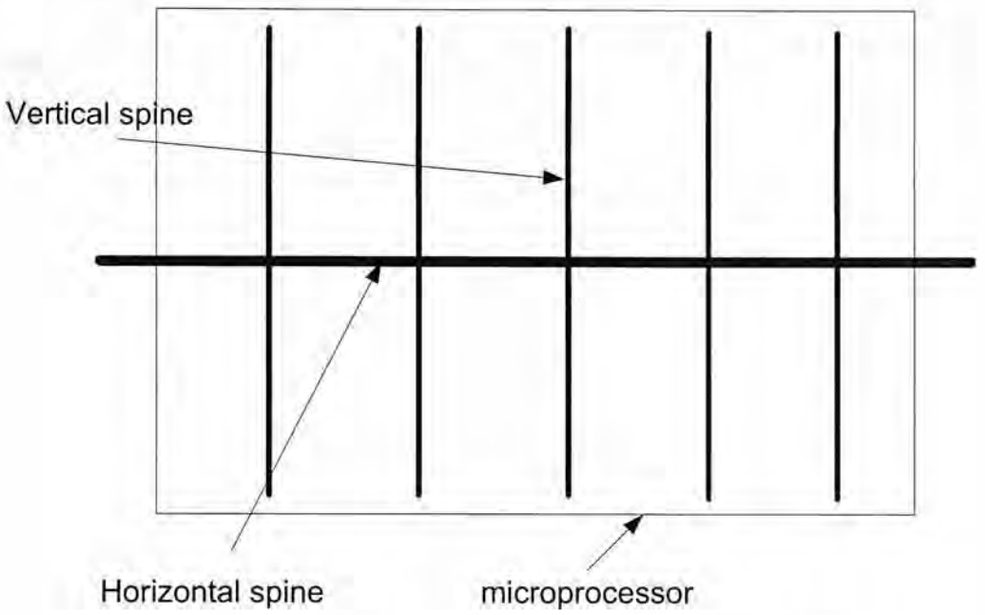


Figure 2.13: Spine structure of a microprocessor.

be tuned and gated independently without affecting others. However, if the system bears a large amount of clock elements, it will require many spine routes, which will in turn introduce a high burden of wire resources and power consumption. A spine structure is shown in Fig. 2.13.

2.3.3 Hybrid Structure

Hybrid structures combining tree and mesh are also a popular way to distribute clock signals for a chip [46, 64, 65, 67, 76, 77]. Several practical designs using stable hybrid clock distribution schemes have been discussed in [60]. For some chips, the performance requirement may be too stringent for a pure tree structure or a pure mesh structure. Using a hybrid clock schemes, we can combine the virtues of tree structures in minimizing the power consumption and mesh structures in reducing skew variability together to achieve a high quality clock distribution scheme. Mesh with local trees was studied in [67] and tree with local meshes (TLM) was studied in [76]. Experimental results in these works show that the hybrid architecture offers significant advantages over both a pure mesh – lower power and faster analysis at an expense of slightly worse skew, and over a tree – smaller skew and more robust to parameter variation.

Tree with crosslinks [45, 52–55, 74, 79] has also been extensively studied in the literature. Aiming at reducing clock skews (global skew or local skew), cross links are either inserted between the leaf nodes or the internal nodes within the clock tree. The major issue of these link insertion algorithms is to find good positions to insert links, which can significantly reduce clock skews of the chip. Some analysis are conducted in [45, 52, 79] to explain the effects when a particular link is inserted into the clock network. Experiment results in [45] show that if we properly insert crosslinks, comparable local skew results can be achieved compared with the results produced by mesh structures. However, the algorithm in [52] only considers inserting crosslinks between the terminal nodes (sink pairs), which can limit the potentials of reducing clock skews with this crosslink insertion technique. In [45], the authors proposed an algorithm considering crosslinks insertion between the internal tree nodes, and experiment results show the superiority of this approach by reducing more clock skews than the approach in [52].

2.4 Post-grid Clock Routing Problem

The post-grid clock routing problem investigated in this thesis bears a fundamental difference with those previous works on constructing clock trees, since the available routing tracks on different metal layers are given and can be very scarce. Besides, in our problem, there are multiple ports and multiple sources in the layout region. There is one very recent work addressing the same problem by Shelar [64, 65] and he proposed a tree growing algorithm to solve the problem with delay and slew constraints. In the algorithm, the clock network is generated by expanding from the sources step by step, and the frontier edge with the smallest wire capacitance is added into the clock network every time. Checking against delay and slew constraints is done whenever a port is being connected.

The algorithm described in [64, 65] formulates the problem on one routing graph $G(V, E)$. The nodes V in this graph correspond to ports and via nodes, while the edges E denote the reserved tracks. Trees, which correspond to clock routes, are initialized by assigning source nodes as their roots. At the very beginning, a pool F of frontier nodes, which is initialized to be all source nodes, is used to store all the current nodes to be expanded. The following steps are performed recursively until all the ports are

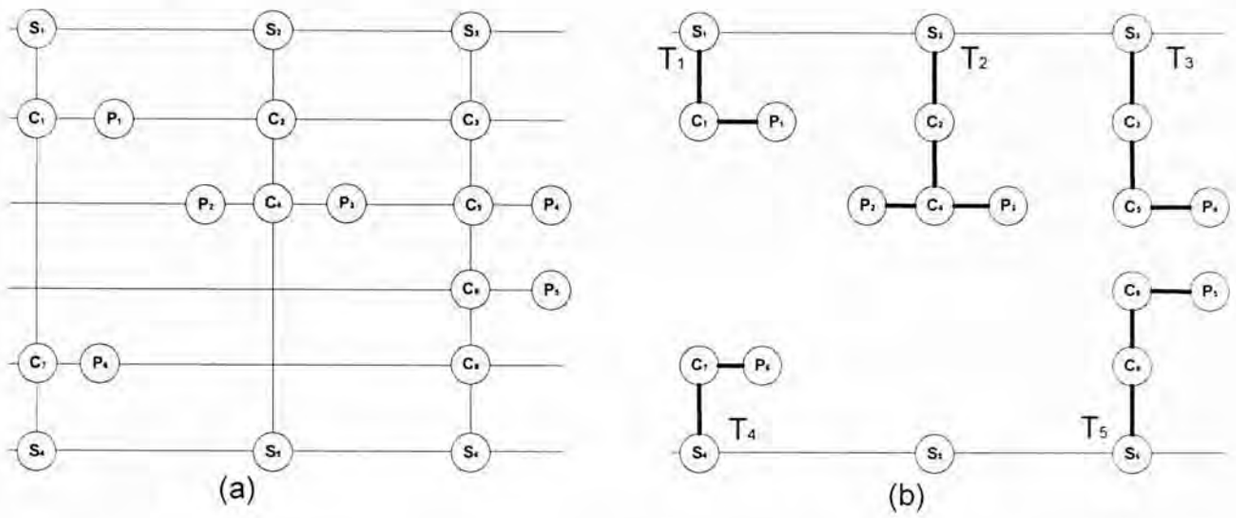


Figure 2.14: (a) A routing graph example of tree growing (TG) algorithm. (b) The final solution contains five trees, T_1, ... , T_5.

connected to the sources. First, all unexplored edges adjacent to a frontier node in F are stored in an edge pool E_f and sorted in an ascending order of their edge capacitances. Then, the clock network is built by a greedy edge expansion process, in which all the edges in E_f are sequentially added into the clock network. Whenever an edge is to be added in the network, the algorithm will ensure that (i) it is not in some other trees already, (ii) adding the edge does not violate any delay or slope constraint. As a result, the routes created by the algorithm are always correct-by-construction. Furthermore, the frontier node pool F will also be updated with the end nodes of the newly added edges. If a port is added into the tree, it will be removed from the set of ports to be connected to the grid wires. After all the ports are connected to the sources, the final clock network is obtained by deleting redundant edges in the trees. The time complexity of this algorithm is $O(|E|^2 log|E|)$, where $|E| = O(n + l + m)$, and n, l, m are the number of port nodes, the number of candidate via nodes, and the number of source nodes respectively. Delay and slew constraints are considered in the algorithm. The delay from a source to a port is computed using the Elmore delay model. The slope, i.e., 10-90% transition time, is computed using the metric in [37]. According to the study in [37], the slope of a signal at the end of a wire segment with resistance R and capacitance C is given by $\sqrt{(2.2RC)^2 + (S_I)^2}$, where S_I is the slope of the waveform at

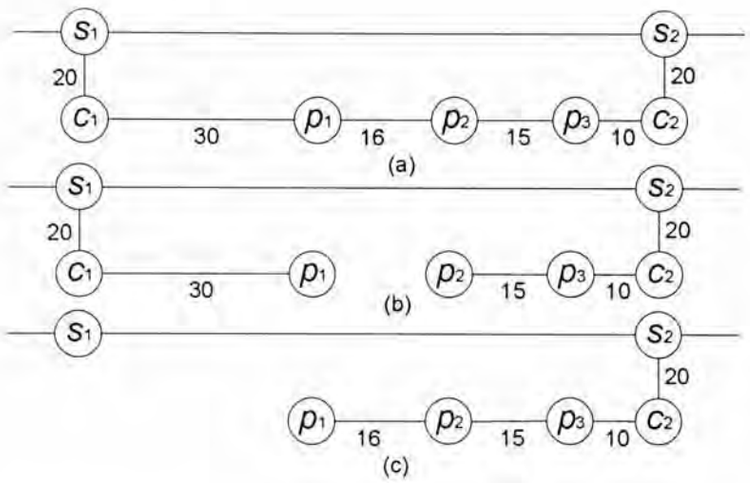


Figure 2.15: (a) Routing graph with sources s_1 and s_2, ports p_1, p_2 and p_3, and via nodes c_1 and c_2. The numbers near the edges denote the wire capacitances. (b) Routing solution of [65]. (c) A better topology with a 36% reduction in wire capacitance.

the input of the segment. In the algorithm, they will lump the resistance and capacitance of the unique path from the root to a node in the tree to compute the slope. A routing example which contains six ports (P_1 to P_6), six sources (S_1 to S_6), and several via nodes (denoted as C_i) is shown in Fig. 2.14. Using this tree growing (TG) approach, we can connect the six ports using five trees, T_1 to T_5, as shown in the right sub-figure of Fig. 2.14.

2.5 Limitations of the Previous Work

As this post-grid clock routing problem is different from traditional clock routing problem, algorithms such as H-tree, MMM, GMA and exact zero skew routing discussed above cannot be directly employed here. Furthermore, the routing method in [65] which deals with the post-grid clock routing problem has a couple of intrinsic drawbacks. First of all, it uses a top-down tree growing heuristic in which the downstream capacitance information is not available when the trees are being constructed, and it thus can hardly optimize the delay. In addition, its slew calculation is based on the lumped-RC model instead of the distributed RC model, and this may lead to accuracy and fidelity problem. All these problems make the heuristic in [64, 65] oversimplified for clock network

designs of high performance microprocessors.

In Fig. 2.15, we show a simple example to illustrate the deficiency of Shelar's approach. In this example, three ports p_1, p_2 and p_3 are to be connected to two sources s_1 and s_2. c_1 and c_2 denotes the via nodes. Fig. 2.15(b) shows the routing topology obtained by the tree growing approach in [65]. Fig. 2.15(c) shows a better clock topology in terms of wire length and capacitance usage. In this example, the topology of Fig. 2.15(c) can achieve a 36% reduction in wire capacitance compared with the result of the tree growing approach.

□ **End of chapter.**

Chapter 3

Post-Grid Clock Routing Problem

3.1 Introduction

An overall clock distribution scheme for high performance microprocessors is shown in Fig. 3.1. The external clock signals are typically generated by a phase locked loop (PLL) and reach the global grid through grid buffers. The grid, typically lying on the topmost metal layer, is usually implemented using spines and will distribute clock signals to different regions of the chip. The grid and PLL are usually designed manually. The grid clock signals will be further routed through a set of pre-reserved tracks on the lower metal layers to the clock ports of different blocks, and this step is called post-grid clock routing. The block-level ports will be created in such a way to align with the reserved tracks and the clock signals will be further sent to different sequentials inside the blocks using buffered clock trees. Thus, the buffered trees are connected at some specific locations (ports), which are connected to the clock grid by routing along the reserved tracks on the lower metal layers. During the block-level synthesis when designing a chip, replicating, placing, sizing of clock cells and routing of clock wires will be performed and this will create the block-level ports, aligned with the reserved tracks for post-grid clock routing. There can be thousands of ports in each layout region in a real post-grid clock routing problem. As shown in Fig. 3.2, the global grid wires are driven by multiple grid buffers and deliver the clock signals to block-level ports by routing along the reserved tracks on the lower metal layers.

A simple example of this post-grid clock routing problem is shown in Fig. 3.3(a). In this example, there are five metal layers (from layer 3 to layer 7) with six ports lying

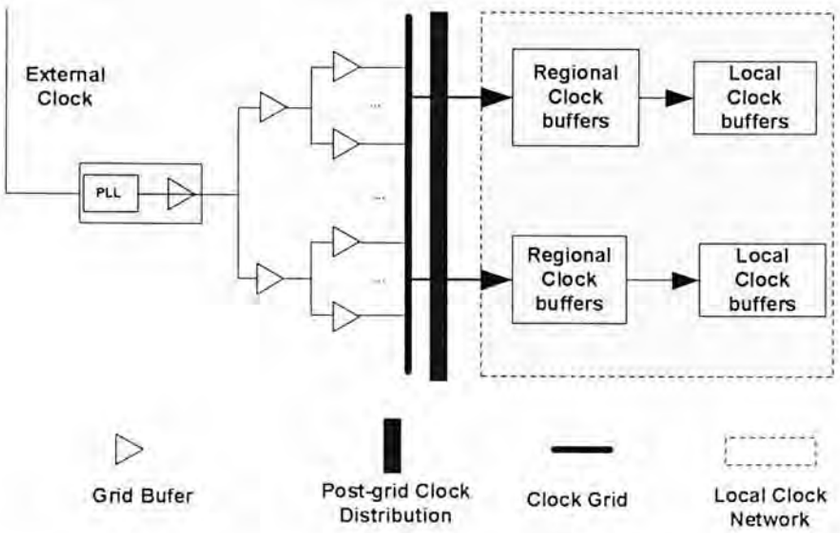


Figure 3.1: Typical clock distribution of microprocessors

on metal layer 3 and the source grid is on metal layer 7. Routing can only be done on those reserved tracks (dashed lines). A sample routing solution is shown in Fig. 3.3(b). The target is to connect all the ports to the sources without exceeding a very stringent delay bound (which is also an upper bound of the skew), and to minimize the total wire capacitance. Note that for this particular instance, our algorithm gets the *optimal* solution as shown in Fig. 3.3(b).

3.2 Problem Definition

In this post-grid clock routing problem, we are given (1) a set of reserved tracks (including the source grid which is always on the topmost metal layer) on different metal layers which have alternate routing directions, (2) the locations and capacitances of n ports $P = \{P_1, P_2, ..., P_n\}$ on some lower metal layers, and (3) the types of wires (with different capacitance/resistance tradeoffs) available on each metal layer. We assume that the clock grid on the topmost layer provides zero-skew clock signals. The objective of this post-grid clock routing problem is to connect all the ports to the sources[1] by making use of the reserved tracks and different wire types so as to satisfy the constraints

[1]These sources are vias to the source grid on the topmost metal layer.

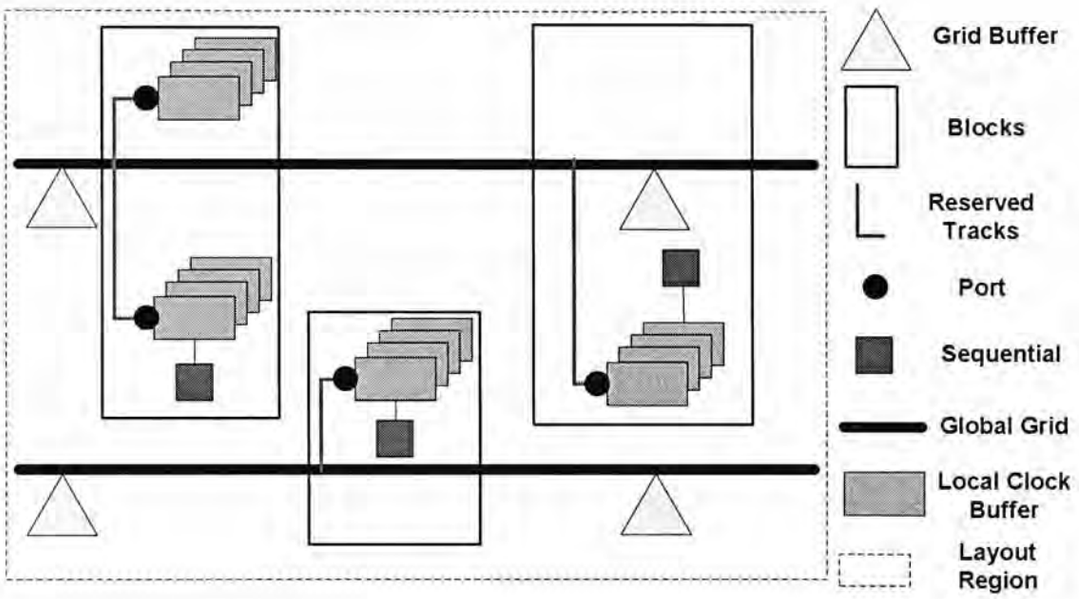
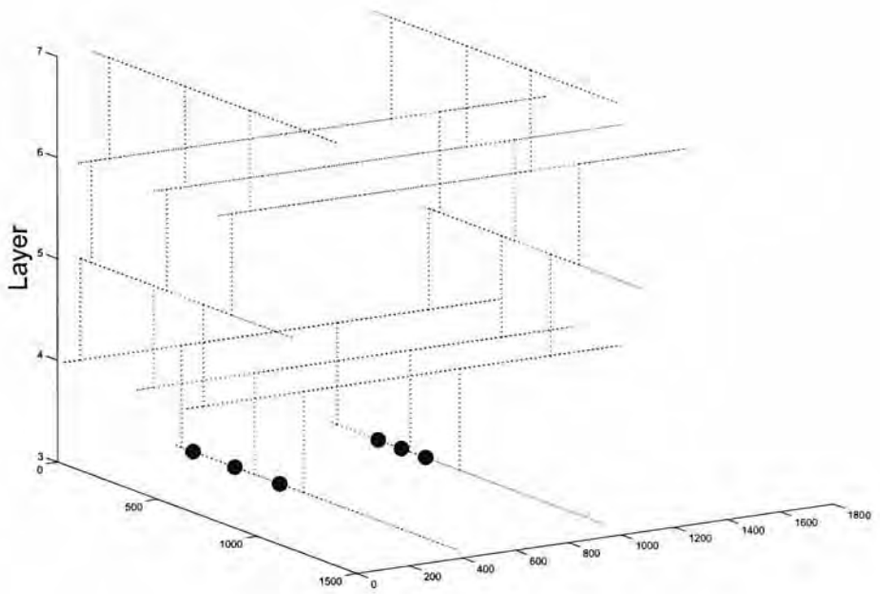


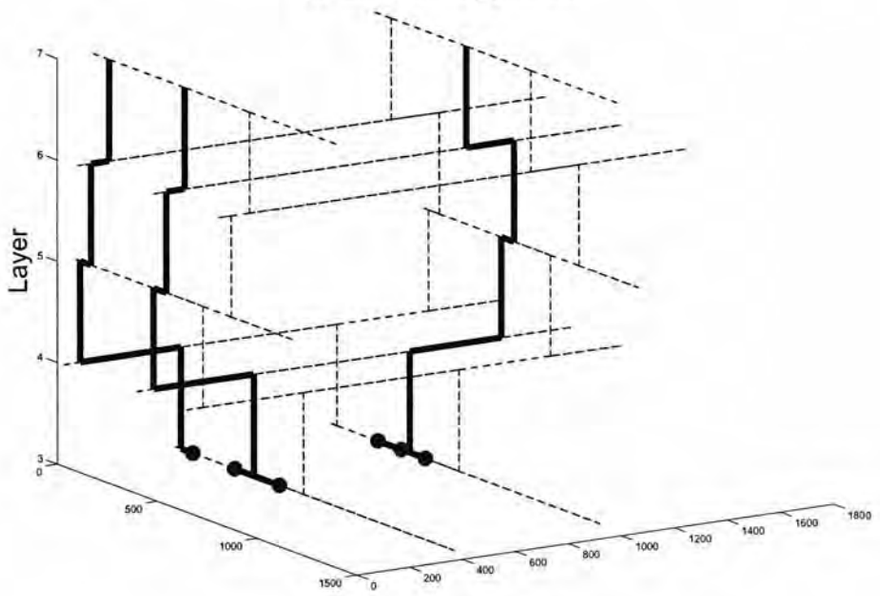
Figure 3.2: Post-grid clock network distribution

on the maximum delay bound D, and to minimize the total wire capacitance. The delay here is computed according to the Elmore delay model due to its simplicity and high fidelity.

Similar to the previous work [65], we do not optimize the skew directly. This is because the grid-to-ports delay bound (also upper bound the skew) is very stringent and is set to be within 5ps for all the data sets, which is very small compared with the overall circuit skew budget. Therefore, it is not necessary to put the skew as another optimizing objective specifically. We can estimate the slew of signals using $\sqrt{(2.2RC)^2 + (S_i)^2}$ according to [37], where R and C denote the resistance and capacitance of the wire segment respectively, and S_i denotes the input slew. To estimate the slew at a particular port, we can simply replace RC by the Elmore delay of the particular port. In the experiment part, we can see that the estimated slew and the simulated slew correlate to each other quite well, and are also very close to each other in their absolute values. In addition, similar to [65], we do not consider buffer insertion in this post-grid clock routing. A very detailed explanation is provided in [65]. In fact, the well-defined grid and reserved tracks make buffer insertion unnecessary for this post-grid clock routing problem.



(a) Initial routing problem



(b) Sample Routing Solution

Figure 3.3: Post-grid clock routing problem

Algorithm 1: Wire replacement

```
1  begin
2      T_r ← all trees;
3      while T_r is not empty do
4          T_i ← select one tree in T_r;
5          P_l ← port with the largest Elmore delay in T_i;
6          P_x ← all terminal ports in T_i except P_l;
7          while P_x is not empty do
8              P_i ← port node in P_x with the smallest Elmore delay;
9              P_a ← lowest common ancestor of P_l and P_i;
10             repeat
11                 Replace e(P_i) using the second type of wire if no violation occurs;
12                 P_i ← parent(P_i);
13             until ∃P_k ∈ T_i where d(P_k) > D or P_i = P_a;
14             P_x ← P_x − P_i;
15         end
16         P_i ← P_l, P_a ← tree root of T_i;
17         repeat steps 10-13;
18         T_r ← T_r − T_i;
19     end
20 end
```

3.3 Our Approach

This post-grid clock routing problem can be seen as a multi-source multi-sink[2] tree construction problem with a delay bound and an objective to minimize the total wire capacitance. We first model the virtual grid of reserved routing tracks by a graph G. The set of vertices contain (1) the block-level clock ports (i.e., the sinks), (2) the possible via positions between reserved tracks on adjacent metal layers, and (3) the clock sources (which are the vias connecting to the source grid). The edges in G represent the wire segments on the reserved tracks connecting ports, vias or sources. Our approach includes a pre-processing step that performs segment merging, finds segment intersections and construction of the graph G and uses some techniques in [12, 13] and it will not be detailed here.

We devise a delay-driven *path expansion* algorithm to solve this clock routing problem. To make our illustration more clear, we define a new term *path* in our approach

[2]These "sinks" are block-level clock ports in our problem and are different from the "sinks", which are flip-flops or latches, in traditional clock routing problems.

Algorithm 2: Topology refinement

```
1  begin
2      P_y ← all terminal ports;
3      sort(P_y) in a non-increasing order of their Elmore delays;
4      while P_y is not empty do
5          P_i ← a port in P_y;
6          modified path_expansion() on P_i;
7          // Paths expand toward all directions, and the
8          // path with smallest wire capacitance will be
9          // expanded first
10         P_y ← P_y − P_i;
11     end
12 end
```

as follows: a path is a routing between an intermediate node (a via node or a source node) and a block-port along the reserved tracks. During the expansion process, we will always select the path with the smallest Elmore delay (note that it is the total delay from the last node of the path to the first node of the path) in the current path pool to be further processed. A path p will be *taken* when it reaches a source. Then, all the paths that intersect with p will also be considered and *taken* if no delay violation occurs. This path expansion step will be repeated until all the ports are connected, or no more ports can be connected without violating the delay constraint. These are the basic steps of our partition-based delay-driven path expansion algorithm. It will be invoked repeatedly (except that the partition of ports will be performed only once) with a pre-processing step that will connect up some critical ports first. Finally, some post-processing techniques are performed to further reduce the total wire capacitance. A flow of our approach is illustrated in Fig. 3.4.

3.3.1 Delay-driven Path Expansion Algorithm

In this delay-driven path expansion algorithm, we will propagate from all the ports simultaneously along the reserved tracks to reach a source. A heap data structure H is used to store all the currently expanding paths sorted according to their Elmore delays. At the beginning, the heap H is initialized with all the ports, which can be regarded as zero length paths with zero delay.

In each step, we will pick a path p from the top of the heap, which has the smallest

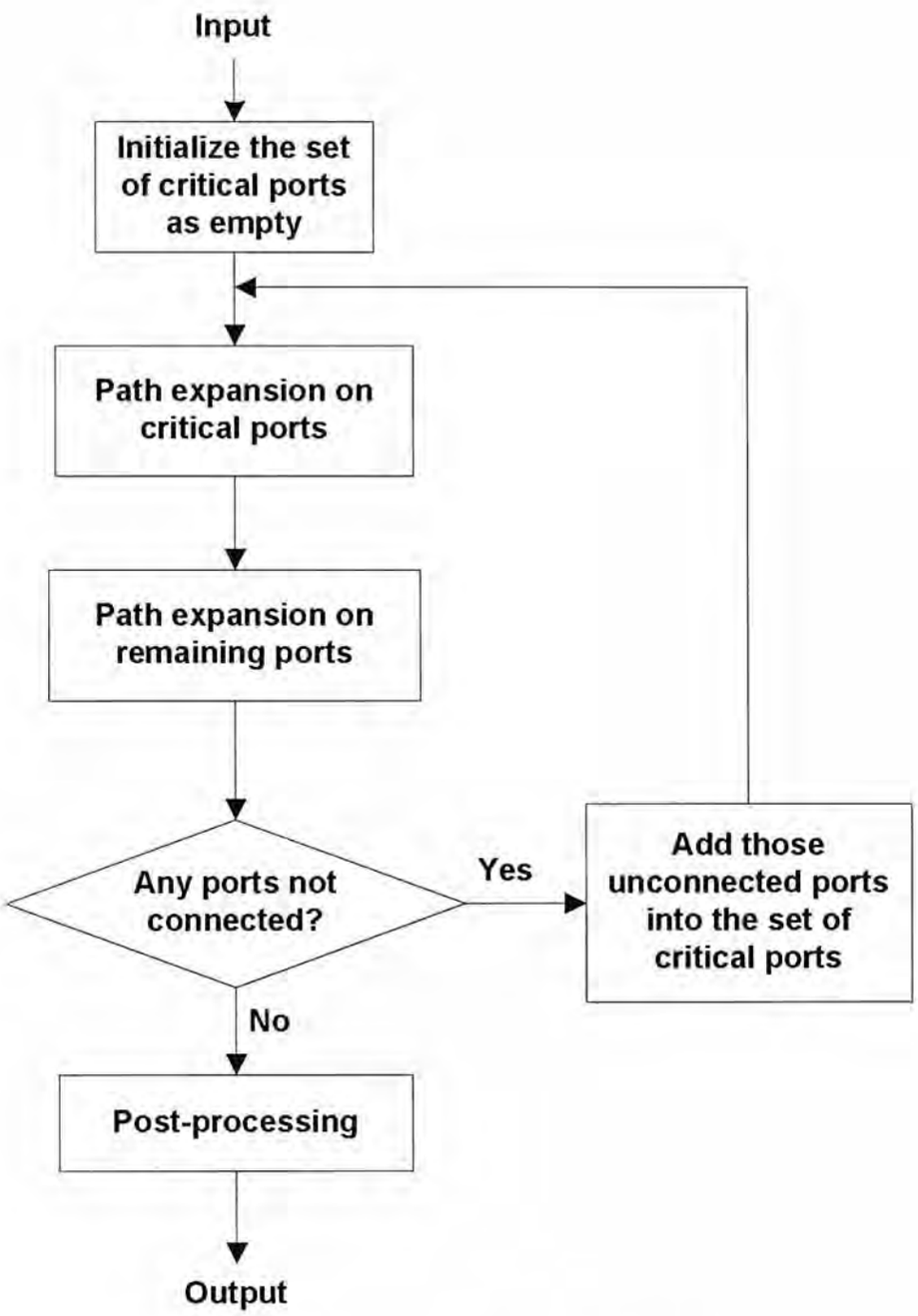


Figure 3.4: An overall flow of our approach

Algorithm 3: Main program

```
1  begin
2      P ← all ports;
3      P_c ← φ; // critical ports
4      k=0;
5      repeat
6          Initialize H as P_c;
7          path_expansion() with H initialized as P_c;
8          Initialize H as P − P_c;
9          path_expansion() with H initialized as P − P_c;
10         P_c ←P_c + ports that fail to be connected to a source;
11         k ← k + 1;
12     until all sinks are connected or k > K;
13     if all sinks are connected then
14         Post-process;
15         // wire replacement and topology refinement
16     else
17         No solutions under current constraints;
18     end
19 end
```

Elmore delay among all the paths in H. We will then check whether p has reached a source. If not yet, we will expand p vertically up if a via [3] exists at the endpoint $last(p)$ of p or will otherwise expand sideways (horizontally or vertically, depending on the track direction of the metal layer the last node of p is lying on) along the reserved tracks. We will first compute the Elmore delays of these new paths. Those new paths with Elmore delay smaller than the delay limit D will be inserted into the heap H. The path p will then be removed from H.

However, if the path p has reached a source, we will first check against the delay constraint. If no violation occurs, we will take this path p into our routing solution. Suppose that the path p is expanded from a port $port(p)$, all the paths originating from $port(p)$ will be removed from H. Furthermore, we will process every path q where q intersects with p. All these paths will be considered in a non-decreasing order of their Elmore delays. For each of these paths q, we will check whether connecting q to p in the routing solution will violate the delay constraint at $port(q)$ as well as at any port in

[3] Note that the capacitance and resistance of the vias are neglected here for simplicity. The same assumption was made in the previous work [65]. However, the via capacitance and resistance can be easily incorporated into our framework by considering them when computing the delay of a path.

the current clock tree under construction. If any violation occurs, we will just neglect q and consider the next candidate. Otherwise, we will take q into the routing solution and connect it to p. We call these paths which do not come to the top of the heap but are processed *chain paths*. Note that once a path is taken into the routing solution, all the nodes on it will be regarded as "sources" for later expansions, and all the paths originating from its port will be removed from H.

Wire length reduction is not directly addressed in our algorithm. But as we always choose a path with the minimum delay to expand and delay is closely related to wire length, paths with shorter wire lengths will have a higher chance to be selected and processed. Therefore, we can expect a reduction in wire length using our approach. A pseudo-code of this path expansion algorithm is shown in Algorithm 4.

Processing of Chain Paths

In the above path expansion algorithm, after a path p is taken into the routing solution, we will process all the paths that intersect with p in the algorithm. First of all, we will initialize a current routing tree T_p as the single path p and initialize a set $chain(p)$ with all the paths in H that intersect with p. The paths in $chain(p)$ are sorted according their Elmore delays in a non-decreasing order. We will then do the following recursively until the set $chain(p)$ becomes empty. First, we will pick and remove a path p_1 from $chain(p)$ that has the smallest Elmore delay. We will then check if connecting p_1 to T_p will violate the delay constraint for $port(p_1)$ as well as for all the existing ports in T_p. If yes, p_1 will be neglected and the next path in $chain(p)$ will be considered. Otherwise, p_1 will be added into T_p and all the paths originating from $port(p_1)$ will be removed from H. Furthermore, all the paths in H that intersect with p_1 will be added into $chain(p)$ recursively.

3.3.2 Pre-processing to Connect Critical ports

The path expansion algorithm does not guarantee connecting all the ports to the sources successfully, especially when the user specified delay constraint is too stringent. If there are critical ports (far away from sources or with very large port capacitance) which are harder to satisfy the requirement, it will be better to generate smaller trees for them

Algorithm 4: Path expansion algorithm

```
1  begin
2      while H is not empty do
3          p = delete_min(H);
4          if p connects to source and d(p) ≤ D then
5              T_p ← p;
6              clean up H;
7              // remove all paths in H that originate from port p
8              foreach p' intersects with p do
9                  chain(p) ← p';
10             end
11             while chain(p) is not empty do
12                 q = delete_min(chain(p));
13                 if adding q to T_p does not violate D constraint then
14                     connect q to T_p;
15                     foreach p' intersects with q do
16                         chain(p) ← p';
17                     end
18                     clean up H;
19                     // remove all paths in H that
20                     // originate from port q
21                 end
22             end
23             Store T_p as one clock tree in the solution;
24         else
25             H ← expansion of p in selected directions;
26         end
27     end
28 end
```

first before handling others. Therefore, our post-grid clock routing algorithm involves iterations of the path expansion algorithm and will identify critical ports that fail to be connected to a source in the previous iteration. Those critical ports will be given higher priority to be processed in the next path expansion iteration such that smaller clock trees are more likely to be generated to connect them.

The pseudo-code in Algorithm 3 summarizes the overall flow of our approach. We create a set of critical ports P_c which is initialized as empty ϕ. We then enter the path expansion iterations in which we first execute the path expansion algorithm on the set of ports in P_c. This gives the critical ports a higher priority to be routed to the sources. We will then invoke the path expansion algorithm on the remaining ports $P - P_c$. Note

that these remaining ports may also be connected to the trees constructed for the critical ports. After that, all the ports that cannot be routed to a source in this round will be added to P_c. Priorities also exist in P_c in which a higher priority is given to those most recently added ports. We repeat these steps until all the ports are connected or the number of iterations exceeds a user defined limit K.[4]

3.3.3 Post-processing to Reduce Capacitance

For all the data sets, there are two types of wires on each layer with capacitance and resistance tradeoffs[5]. The first type has higher capacitance but lower resistance per unit length, while the second type has lower capacitance but higher resistance per unit length. The per unit length delay of type-one wire is less than that of type-two wire on all the layers. In our path expansion algorithm, we will first use type-one wire on all layers to optimize delay as much as possible. A post-processing step is then performed to reduce the total wire capacitance as long as the delay constraint is maintained by replacing the wire types. Two techniques, *wire replacement* and *topology refinement*, are invoked in this post-processing step.

Wire Replacement

This refinement process is done for all trees in the clock network one after another with the following steps. First, all the terminal ports in the current tree are stored in a port pool P_x in which they are sorted in a non-decreasing order of their Elmore delays, and the port P_l with the largest delay in the tree will be recorded. We will then sequentially explore all the ports in P_x. Without loss of generality, lets assume that the currently processing port is P_i, and node P_j is the parent node of P_i in the tree. We use $e(P_i)$ to denote the edge connecting P_i and P_j. We will then check whether any violation occurs if $e(P_i)$ is replaced by the second type of wire. If not, we will replace it with the second type of wire and set $P_i = P_j$. This step is repeated until the delay constraint is violated at any port in the current tree, or when P_i becomes an ancestor of the node P_l (since we do not want to increase the largest delay in this tree). Port P_l will be finally

[4] In this case, the algorithm fails to converge to a feasible solution. Note that this may happen when the delay constraint is too stringent.

[5] Our algorithm can also handle the case that multiple types of wire are available on each layer.

explored after all other ports in the tree have been processed. In our implementation, the above process is repeated three times, as we find that for most test cases, running more iterations of this wire replacement process brings little or no capacitance reduction. The pseudo-code in Algorithm 1 details the flow of this wire replacement process.

Table 3.1: Comparisons with TG (with topology refinement only for post-processing)

Test Cases	No. Sinks	Capacitance (pf)			Wire Length (mm)			Delay (ps)	Runtime (s)	
		TG x_1	PE* x_2	Improvement $\frac{x_1-x_2}{x_1}$ %	TG y_1	PE* y_2	Improvement $\frac{y_1-y_2}{y_1}$ %		TG	PE*
test1	300	3.3	2.6 (2.8)	20.9 (16.0)	12.6	10.0 (10.6)	20.1 (15.5)	0.45	0.02	0.27
test2	1846	13.7	9.7 (10.6)	29.2 (22.4)	42.9	32.3 (34.9)	24.8 (18.6)	1.15	0.10	2.72
test3	836	8.1	5.2 (5.8)	36.7 (28.2)	32.2	20.6 (23.1)	36.7 (28.5)	0.80	1.35	3.01
test4	502	5.3	4.0 (4.5)	23.8 (14.6)	12.4	9.5 (11.0)	22.8 (11.0)	1.35	0.03	2.89
test5	137	1.4	1.1 (1.2)	21.0 (15.7)	3.4	2.7 (3.1)	19.4 (10.5)	1.10	0.01	0.09
test6	724	7.9	5.7 (6.2)	27.0 (21.7)	18.8	14.2 (15.5)	24.6 (17.4)	1.25	0.05	0.68
test7	981	9.9	7.5 (8.2)	23.8 (17.2)	23.2	17.9 (19.9)	22.9 (14.1)	1.45	0.05	1.00
test8	538	5.9	4.5 (4.8)	24.6 (18.0)	14.1	10.8 (12.2)	23.8 (13.3)	1.80	0.04	0.49
test9	1915	19.9	14.3 (15.6)	28.3 (21.5)	46.1	33.1 (37.0)	28.0 (19.7)	2.75	0.13	3.27
test10	1134	10.7	8.6 (9.4)	19.6 (12.4)	25.8	20.1 (22.0)	22.2 (14.8)	1.90	0.09	6.92
test11	724	6.6	4.9 (5.3)	25.0 (18.9)	13.5	10.5 (11.3)	23.3 (16.5)	1.05	0.04	2.95
test12	225	2.5	2.0 (2.1)	20.1 (13.8)	6.3	4.9 (5.4)	21.9 (13.7)	1.30	0.01	0.17
test13	859	9.5	7.2 (7.6)	24.1 (19.3)	24.1	18.8 (20.4)	22.2 (15.4)	1.10	0.06	0.93
test14	366	3.9	3.1 (3.3)	20.7 (15.9)	9.5	7.8 (8.5)	18.3 (10.8)	0.95	0.04	0.30
Ave.	792	7.7	5.7 (6.2)	24.6 (18.2)	20.4	15.2 (16.8)	23.6 (15.7)		0.14	1.84

Note 1: PE* denotes our *delay-driven path expansion algorithm* with topology refinement only for post-processing.
Note 2: Both TG and PE* use just type one wire on every layer.
Note 3: The figures inside brackets denote the results before the post-processing techniques.

Topology Refinement

In the path expansion algorithm, we will expand a path p upwards as long as the end node of p is at a via connecting to the upper layer. Besides, chain paths are greedily

Table 3.2: Comparisons with TG (using all techniques)

Test Cases	No. Sinks	Capacitance (pf)		Wire Length (mm)		Delay (ps)	Runtime (s)	
		TG	PE	TG	PE		TG	PE
test1	300	3.3	2.3	12.6	10.6	0.45	0.02	0.23
test2	1846	13.7	4.9	42.9	33.4	1.15	0.10	2.85
test3	836	8.1	4.2	32.2	22.4	0.80	1.35	2.78
test4	502	5.3	1.6	12.4	9.9	1.35	0.03	2.99
test5	137	1.4	0.5	3.4	2.8	1.10	0.01	0.10
test6	724	7.9	2.4	18.8	14.5	1.25	0.05	0.78
test7	981	9.9	3.0	23.2	17.9	1.45	0.05	1.14
test8	538	5.9	1.8	14.1	10.8	1.80	0.04	0.58
test9	1915	19.9	5.2	46.1	32.6	2.75	0.13	3.68
test10	1134	10.7	3.2	25.8	19.6	1.90	0.09	7.08
test11	724	6.6	1.8	13.5	10.4	1.05	0.04	3.02
test12	225	2.5	0.9	6.3	5.0	1.30	0.01	0.20
test13	859	9.5	3.2	24.1	19.1	1.10	0.06	1.07
test14	366	3.9	1.3	9.5	7.9	0.95	0.04	0.34
Ave.	792	7.7	2.6	20.4	15.5		0.14	1.92

Note 1: "PE" denotes our regular approach of having two choices of wires on each layer

processed as long as the delay bound is maintained. Thus, there are still chances to bring down the capacitance by changing the topology of the initially constructed trees. To achieve this, we will employ a topology refinement step on all the terminal ports as follows. First, we will sort all the ports that are terminal nodes in the trees in a non-increasing order of their Elmore delays in a port pool P_y. These ports will be processed sequentially in the algorithm. For any port P_i being processed, we will first disconnect P_i from the tree it is currently connecting to, and record the total wire capacitance C_b of the removed path p_i. A new path expansion algorithm will then be invoked at P_i which is different from the previous path expansion algorithm that (1) only the second type of wire will be used during the path expansion process, (2) paths will be expanded in all possible directions and (3) the path with the minimum wire capacitance (instead of

the minimum wire delay) will be selected and processed first in the expansion process. New paths with wire capacitance less than C_b will be inserted into the heap. Once a path reaches a source or a tree (note that all trees are connected to sources now), we will check whether delay violation occurs if the new path is taken. This new path will be taken if no violation occurs. Otherwise, we will continue the modified path expansion algorithm until another path reaches a source or a tree, or when all the paths are exhausted. If all the paths are explored but no path is successfully connected, we will simply restore the original path p_i. The above steps are repeated three times in our implementation. Algorithm 2 shows the flow of this topology refinement process.

3.4 Experimental Results

3.4.1 Experiment Setup

The path expansion algorithm proposed in this paper is implemented in C++ and all the experiments are carried out on a Linux machine with 4GB RAM and a Pentium 4 microprocessor running at 3.2GHz. We have also implemented the tree growing approach (TG) in [65] using C++ for comparisons. In the experiments, we assume that the slew of the source signals is 10ps. In the simulation, we are observing the 50% delays to compare with the Elmore delays calculated using our algorithm. It has been shown in [28] that the Elmore delay serves as a 50% delay upper bound with respect to any ramp input signals of the chip. The first three test cases (test1-3) are provided by industry. The remaining eleven test cases are obtained from the circuits used in the ISPD 2010 Clock Network Synthesis Contest [1]. For the ISPD test cases which have no layer information given, five layers of reserved tracks are added according to the track conventions used in test 1-3.

3.4.2 Validations of the Delay and Slew Estimation

As we have simplified our problem to consider only maximum delay constraint, we want to validate the assumption that delay and slew of the circuit are closely related. We have constructed a simple circuit containing only one port, and run our path expansion algorithm to construct a tree structure for this port. For the tree we have constructed,

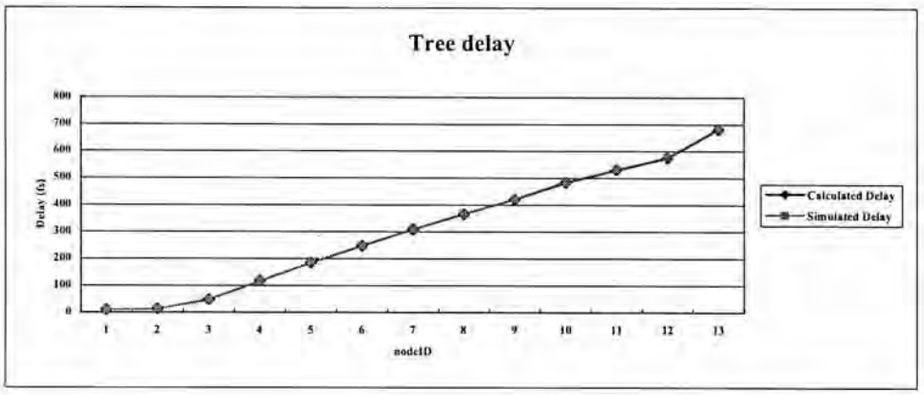


Figure 3.5: Comparisons between computed and simulated delays in a tree structure

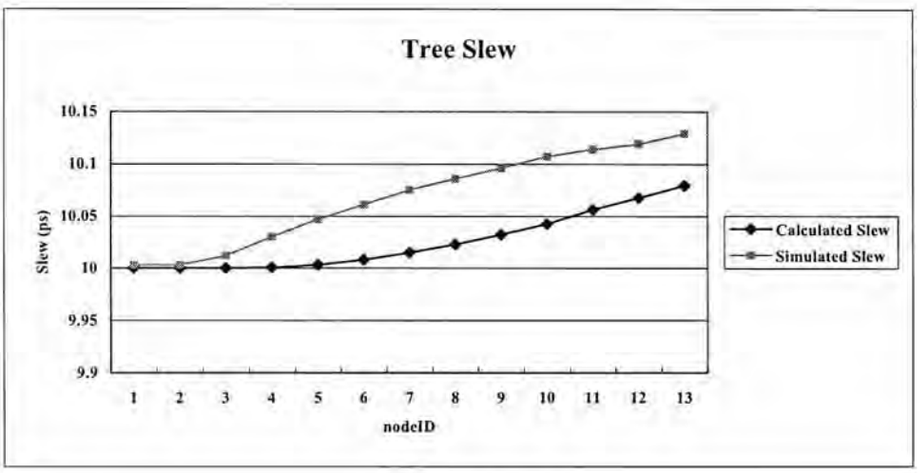


Figure 3.6: Comparisons between computed and simulated slews in a tree structure

there are totally 13 nodes (including the source node and the port). We compare the delays and slews we have calculated with the delays and slews we got by running hspice simulations. The detailed results for the tree structure are shown in Fig. 3.5 and Fig. 3.6.

We can see that for all the 13 nodes, the delays we calculated and the delays we obtained from simulations are closely correlated to each other. For the slew, there is a small gap between the estimated ones using the formula $\sqrt{(2.2RC)^2 + (S_i)^2}$ and the ones using hspice simulation. They are still very close to each other, and are highly correlated with each other. We can also see that the differences between the simulated slew and the calculated slew are very small, which confirms the correctness of the above

slew estimation.

3.4.3 Comparisons with the Tree Grow (TG) Approach

Since the approach in [65] considers only one type of wire on each layer, for fair comparison, we compare the result of our approach using just the first type of wire on every layer (i.e., without the wire replacement step and use only type one wire in all the other steps) with the result of [65] using the first type of wire on every layer. "PE*" denotes the approach using just the first type of wire on every layer and using topology refinement only in post-processing step. In these experiments, we first get the lowest achievable delays obtained by TG empirically on all the test cases and use these delays as our delay bounds. The results are shown in Table 3.1. Column 3 and 6 show the total wire capacitance and the total wire length generated by TG. The results of our approach are shown in column 4 and 7. On average, our approach provides a 24.6% improvement in the total wire capacitance and a 23.6% improvement in the total wire length compared with TG respectively. The running times of both algorithm are shown in the last two columns. As we can see that though our approach is slower, the runtimes are still very practical. For all the test cases, the running times of our approach are within seconds.

If we allow both types of wires on each layer, further reduction in wire capacitance can be obtained and the results are shown in Table 3.2. As we can see from the result, our approach can make good use of the availability of different wire types to further reduce the capacitance. For example, in test2, the wire capacitance can be reduced significantly by 49% (from 9.68pf to 4.95pf) with the wire replacement step. On average, the major path expansion algorithm, the topology refinement step and the wire replacement step take 44%, 31% and 25% of the total running time respectively. Note that in some cases, the running time of "PE" is even larger than that of "PE*" although "PE" does not perform the wire replacement step. This is because the inputs to the topology refinement procedure in "PE*" and "PE" are different as "PE" does not perform wire replacement. There are thus variations in the running times of the topology refinement step.

3.4.4 Lowest Achievable Delays

Our approach can actually produce solution with better delay than the TG approach. We have run our algorithm on all the test cases to get the smallest achievable delays. The results are shown in Table 3.3. For almost all the test cases, we can further reduce the delays generated by TG. Take *test3* as an example, we can significantly reduce the delay from 0.80ps to 0.55ps, which shows an advantage of using our method in satisfying stringent user specified delay limits. In practice, designers may not know whether a delay limit is achievable for a circuit. Our approach can help in determining the lowest achievable delay by embedding the algorithm in a binary search loop. This is possible since our approach will take the delay limit as an input constraint.

Table 3.3: Lowest achievable delays

Test Cases	Capacitance (pf)	Wire Length (mm)	Delay (ps)	Runtime (s)
test1	2.27	10.6	0.45	0.20
test2	6.08	34.6	0.47	4.69
test3	5.06	24.6	0.55	4.56
test4	1.75	10.2	1.00	2.12
test5	0.55	3.0	0.86	0.12
test6	2.83	15.5	0.83	1.01
test7	3.02	18.0	1.35	1.60
test8	1.86	11.2	1.32	0.50
test9	5.45	33.7	1.95	28.22
test10	3.28	20.1	1.67	22.27
test11	1.92	10.7	0.89	2.87
test12	0.91	5.1	1.12	0.22
test13	3.43	19.9	0.90	1.47
test14	1.48	8.4	0.67	0.40

3.4.5 Simulation Results

We further validate our results using hspice simulations. The slew of the input signals are set to be 10ps. The estimated slew of the circuits are shown in column three us-

ing $\sqrt{(2.2RC)^2 + (S_i)^2}$ according to [37], where RC is replaced by the largest Elmore delay of the circuit. Detailed results are shown in Table 3.4. As we can see from the simulation results, The delay and slew we calculated is very close to the simulation results.The correlation coefficient is over 99% between the simulated delay and calculated delay while it is over 98% between the simulated slew and calculated slew. This verifies the correctness of our method.

Table 3.4: Simulation results for tree

Test Cases	Calculated Results		Simulation Results	
	Delay (ps)	Slew (ps)	Delay (ps)	Slew (ps)
test1	0.45	10.05	0.45	10.07
test2	1.14	10.32	1.14	10.24
test3	0.80	10.15	0.80	10.14
test4	1.35	10.43	1.35	10.36
test5	1.09	10.29	1.09	10.25
test6	1.24	10.37	1.24	10.32
test7	1.43	10.50	1.43	10.51
test8	1.78	10.76	1.78	10.90
test9	2.75	11.69	2.70	11.41
test10	1.90	10.84	1.90	11.02
test11	1.05	10.26	1.05	10.24
test12	1.29	10.40	1.28	10.31
test13	1.09	10.29	1.09	10.24
test14	0.95	10.22	0.95	10.20

Chapter 4

Non-tree Based Post-Grid Clock Routing Problem

4.1 Introduction

All the above techniques focus on constructing trees for the clock distribution network. To make our approach more practical and to handle some difficult cases, we have also considered non-tree structures in our algorithm. Our main objective with non-tree topology is to reduce the maximum delays of the ports. In practice, there are cases in which a small number of ports have exceptionally large capacitances that even its shortest direct connection to the nearest source will have a delay exceeding the limit D. Since the shortest path delay is a lower delay bound one can achieve using tree structure, we can never satisfy the delay requirement for this kind of *problematic* ports without using non-tree topologies. For our general post-grid clock routing algorithm, we will first identify those problematic ports and construct some non-tree structures to bring down their delays to be within the delay limit D. After connecting all those problematic ports, the remaining ports will be connected using the previous post-grid clock routing algorithm in chapter 3. Note that for the remaining ports, they are also allowed to connect to the non-tree structures constructed for the problematic ports, as long as no delay violation occurs. Fig. 4.1 shows the overall flow of our general post-grid clock routing algorithm.

A general RC network can be denoted by a graph $G = (V, E)$, where V denotes all the sources, sinks and internal nodes, and E denotes the interconnections between all

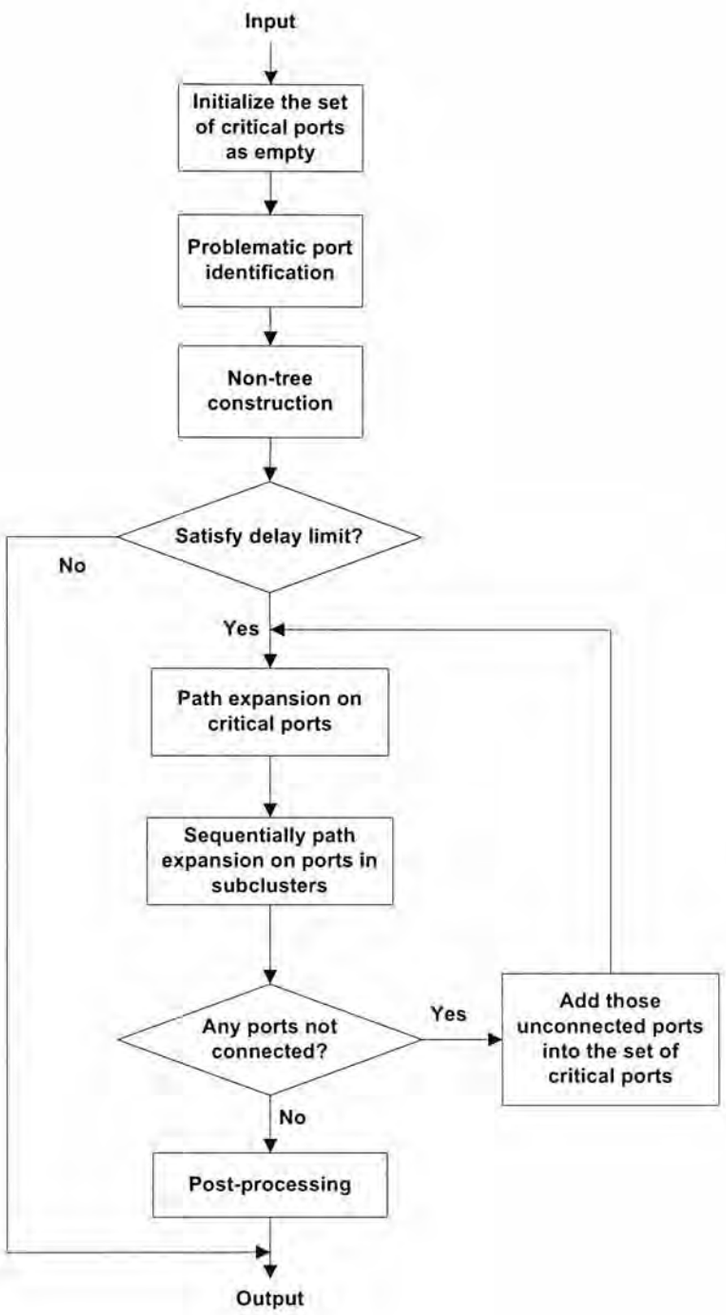


Figure 4.1: General post-grid clock routing algorithm

the nodes in V. The Elmore delay at node i, denoted as t_i, can be expressed as

$$t_i = \sum_j (R_{i,j} C_j) \tag{4.1}$$

where C_j is the ground capacitance at a node j in V. The transfer resistance $R_{i,j}$ is equal to the voltage at node i when 1A current is injected into node j while all other node capacitances are zero [61, 78]. In the special case where we have a tree topology, $R_{i,j}$ is simply the total resistance along the common path shared by node i and j [61,78]. We can decompose a general RC network into a spanning tree $T = (V, E_t)$ and several crosslinks E_l [9]. The computational model in [9] can be used to evaluate the delay at different nodes when crosslinks exist in the clock network. Consider a particular sink i and assume that its original delay is t_i. For a link connecting node u and w with capacitance C_l and resistance R_l, its effects on the delay of sink i can be analyzed as follows.

To consider the effect of the link capacitance, we first add a load capacitance of $\frac{C_l}{2}$ at node u and w. The delay of sink i after adding the load capacitance, denoted as $\tilde{t_i}$, becomes

$$\tilde{t_i} = t_i + \frac{C_l}{2}(R_{i,u} + R_{i,w}) \tag{4.2}$$

According to [9] the delay at node i, denoted by $\hat{t_i}$, after considering the link resistance and can be computed as

$$\hat{t_i} = \tilde{t_i} + \frac{\tilde{t_u} - \tilde{t_w}}{R_l + r_u - r_w} r_i \tag{4.3}$$

where r_i, r_u, and r_w are the Elmore delay at node i, u, w respectively, when C_u = 1, C_w = -1 and all other node capacitances are set to be zero.

Using the above technique, we can evaluate the effects of a link on the Elmore delays of the ports in a non-tree structure. We can further decide whether to accept or reject a link based on the computed delays.

4.2 Handling Ports with Large Load Capacitances

For problematic port with its shortest delay to the closest source larger than the delay limit, there is no way to satisfy the delay constraint using a tree-based clock tree. We

must resort to non-tree topology for the clock network in order to solve the problem successful. Details are shown in the following sections.

4.2.1 Problem Ports Identification

Here, we define *problem ports* as those ports whose smallest Elmore delays from the source are still beyond our target delay limit D. Identifying the problematic ports is quite straightforward. A modified *path expansion algorithm* will be utilized on all the ports individually to first identify those problems ports, in which the ports can expand toward all possible directions. When a port reaches a first source, the corresponding delay D_{min} will be the lower bound delay we can achieve using tree structures. All those ports with D_{min} larger than our delay limit D will be classified as problematic ports. Those problematic ports will be selected to be processed first in our general post-grid clock routing algorithm.

4.2.2 Non-Tree Construction

To handle these problematic ports whose shortest path delays exceed the delay limit D, we have extended our algorithm to first connect those problematic ports by a non-tree structure to several sources to bring down the delay to within the limit D, and then connect the remaining ports to the sources. The non-tree structure is constructed by connecting the problematic port to more than one sources by several paths and by adding crosslinks between those paths.

Consider a particular problematic port P_e, after we make a shortest path connection p_1 for port P_e we will do the following steps to create a non-tree structure. First, we will expand from the first node n_l of p_1 in the opposite direction to find another nearest source. Let p_2 be the new path. p_2 will be taken into the routing solution if it helps in reducing the delay of P_e. Then, all the crosslinks between p_1 and p_2 (note that crosslinks can only exist at locations with reserved tracks) will be recorded and examined. The computational model in [9] is used to calculate the delays at the ports when crosslinks exist. All the crosslinks that can reduce the delay of P_e will be taken into the routing solution one by one until the delay constraint is met, or when all the crosslinks are exhausted. If the delay is still violated after adding all the crosslinks, we will set $n_l =$

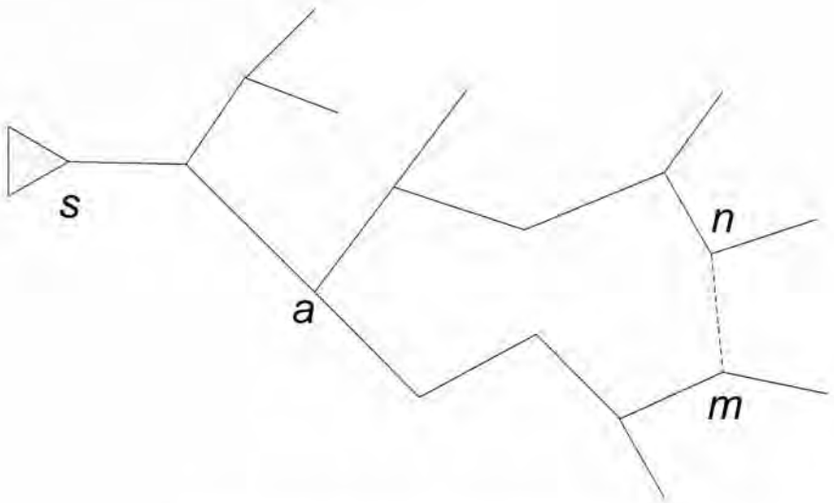


Figure 4.2: Example of adding links

$parent(n_l)$ and repeat the above steps recursively with one edge up the original path p_1 to find more sources and crosslinks.

After handling all the problematic ports, other ports will be handled as usual according to Algorithm 3. Note that we also allow other ports to connect to the non-tree structures, as long as the delay constraint is not violated.

4.2.3 Wire Link Selection

Assume that we are dealing with the topology as shown in Fig. 4.2, the resistance and capacitance of the currently inserted link mn are r_l and c_l respectively. The impact of the link capacitance can be analyzed by adding a capacitance of $c_l/2$ at node n and node m. Define r_{ij} as the lumped resistance along the unique path from node i to node j in the tree. Let the vector $\tilde{t}$ denote the delays of all the nodes in the tree after we added the link capacitance, and $\tilde{t}_i$ denote the delay at node i after we added the link capacitance.

The vector $\tilde{t}$ can be expressed as

$$\tilde{t} = \begin{bmatrix} \cdots \\ t_m + \frac{c_l}{2}(2r_{sa} + r_{am}) \\ \cdots \\ t_n + \frac{c_l}{2}(2r_{sa} + r_{an}) \\ \cdots \end{bmatrix} \tag{4.4}$$

where t_m and t_n are the original delays at node m and n

The Elmore delay of the network after inserting the link resistance r_l can be computed using the techniques proposed in [9]. Here, we use the vector $\tilde{p}$ with n elements where n is the number of nodes in the tree. We use p_i to denote the element corresponding to node i in $\tilde{p}$. Let $\hat{t}$ denote the delay vector after considering the resistance of the crosslink. The delay at node i, denoted by $\hat{t_i}$, can be expressed as

$$\hat{t_i} = \tilde{t_i} - \frac{\tilde{t_n} - \tilde{t_m}}{r_l + p_n - p_m}\tilde{p_i} \tag{4.5}$$

where p_i, p_n and p_m are the Elmore delay at node i, n, and m respectively when $C_n = 1$, $C_m = -1$ and all the other nodes have zero capacitance. For $\tilde{p}$, we have

$$\tilde{p} = \begin{bmatrix} \cdots \\ -r_{am} \\ \cdots \\ r_{an} \\ \cdots \end{bmatrix} \tag{4.6}$$

From equation (4.4), equation (4.5) and equation (4.6), the delay vector, $\hat{t}$, can be cal-

culated as

$$\begin{aligned}
\hat{t} &= \tilde{t} - \frac{\tilde{t_n} - \tilde{t_m}}{r_l + p_n - p_m}\tilde{p} \\
&= \begin{bmatrix} \\ \cdots \\ t_m + \frac{c_l}{2}(2r_{sa} + r_{am}) \\ \cdots \\ t_n + \frac{c_l}{2}(2r_{sa} + r_{an}) \\ \cdots \\ \\ \end{bmatrix} - \frac{t_n - t_m + \frac{c_l}{2}(r_{an} - r_{am})}{r_l + r_{am} + r_{an}}\tilde{p} \\
&= \begin{bmatrix} \\ \cdots \\ t_m + \frac{c_l}{2}(2r_{sa} + r_{am}) + \frac{(t_n - t_m) + \frac{c_l}{2}(r_{an} - r_{am})}{R_{loop}} r_{am} \\ \cdots \\ t_n + \frac{c_l}{2}(2r_{sa} + r_{an}) - \frac{(t_n - t_m) + \frac{c_l}{2}(r_{an} - r_{am})}{R_{loop}} r_{an} \\ \cdots \\ \\ \end{bmatrix}
\end{aligned} \tag{4.7}$$

where node a is the lowest common ancestor of node m and n, and R_{loop} is the total resistance along the loop $m \to a \to n \to m$. Denote the value of $-(\frac{c_l}{2}(2r_{sa} + r_{am}) + \frac{(t_n - t_m) + \frac{c_l}{2}(r_{an} - r_{am})}{R_{loop}} r_{am})$ by I. Assume that we are now trying to reduce the delay at node m. For all the links between the two paths $a \to m$ and $a \to n$, we will calculate their values of I and then sort them in descending order according to their values. The value of I gives us some information on how effective a link is in order to reduce the Elmore delay at node m. The larger the I is, the more effective will be the link in reducing the delay at node m and also possibly the delay of the descendant node of m. In our implementation, all the links will be considered sequentially and added to the network temporally to check if it will help in reducing the delays of the problematic ports (Note that when a crosslink is inserted, we will compute the delays of the problematic ports using the approach in [9].). A crosslink will be accepted if it can reduce the delay of the problematic port. The crosslink insertion process will be repeated until the user defined delay limit is satisfied or when all the links are exhausted.

4.3 Path Expansion in Non-tree Algorithm

The *path expansion* algorithm here is basically the same as that in the previous tree construction algorithm. Additionally, we also allow ports to connect to a non-tree structure as long as the delay limit is not violated. In every new iteration, we also keep the non-tree structures constructed in the previous iteration unchanged. For example, if a regular port a connect to a non-tree structure T constructed for the problematic ports, the port a will not be routed again in the next iteration of the main algorithm. All the non-tree structures constructed in the previous round will stay unchanged. Then the next round of the path expansion algorithm starts. By doing this, we want to ensure that the difficult ports are routed first before routing other relatively easier ports.

4.4 Limitations of the Non-tree Algorithm

Due to the limited number of available tracks, our non-tree algorithm does not always converge to a feasible solution, especially when a large number of ports lie within a small region of the chip with less available tracks. In this case, we can hardly find useful sources and crosslinks, and the algorithm will finally fail to find a feasible solution. Moreover, the existence of non-tree structures can "block" the paths of other ports, which is also a contributing factor to possible failure of the algorithm. When such failure occurs, we will increase the user specified delay limit D to further explore possible solutions. When D is larger than the minimum delay among all the ports, which means that there are no problematic ports, the non-tree algorithm automatically degenerates into the path expansion algorithm.

4.5 Experimental Results

4.5.1 Experiment Setup

In the experiments, we use the same settings as used in section 3.4. The slew of the source signal is set to be 10ps. In hspice simulation, 50% delay is measured to compare with the Elmore delay we have calculated. In addition, we have create another 14 test cases based on the original benchmarks we have. We increase the capacitance of a

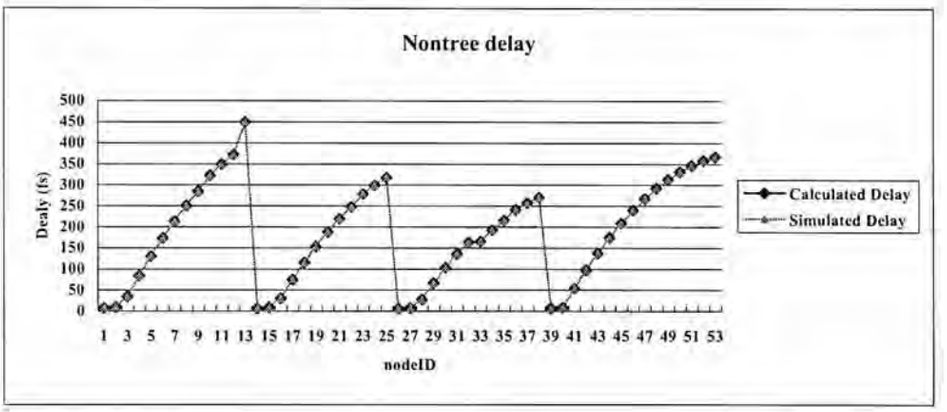


Figure 4.3: Delay comparisons (Non-tree structure)

certain amount of ports (see section 4.5.4) in all the test cases and run our general clock routing algorithm. Detailed experiment results are explained in the following sections.

4.5.2 Validations of the Delay and Slew Estimation

We use the same circuit that contains only one port as used in section 3.4.2 to validate our delay and slew estimations. We have constructed a non-tree structure containing 53 nodes in total for this simple circuit. Detailed results about the delay and slew are shown in Fig. 4.3 and Fig. 4.4.

We can see that the delay computational model can accurately compute the delays of all the nodes within this non-tree structure. The slew predication is similar with the results of the tree path expansion algorithm detailed in section 3.4.2. Although there is a gap between the calculated results and the simulated results, their difference is relatively small. From Fig. 4.4, we can see that they also correlate to each other quite well. This again verifies the correctness of our slew predication method.

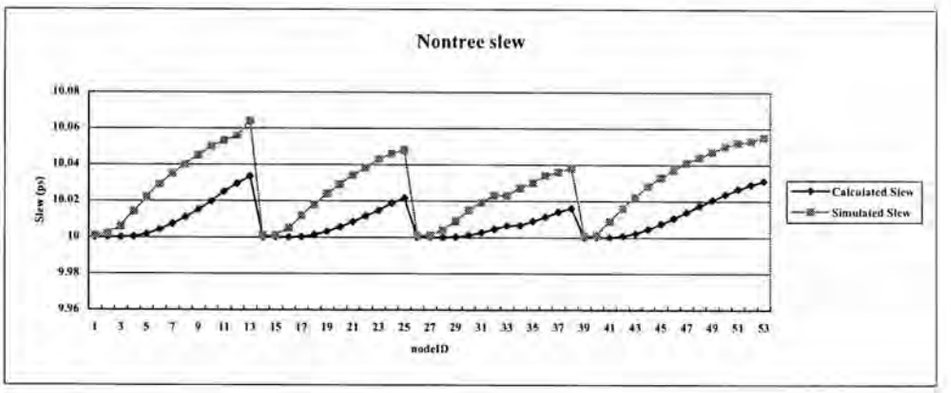


Figure 4.4: Slew comparisons (Non-tree structure)

4.5.3 Lowest Achievable Delays

With the above non-tree techniques, our approach can actually produce solution with even tighter delay bound than that in section 3.4.4. We have run our non-tree algorithm on all test cases to get the smallest achievable delays. The results are shown in Table 4.1. Compared with the original path expansion algorithm, our general post-grid routing algorithm can further decrease the lowest achievable delays by constructing non-tree structures for those problematic ports. Take *test1* as an example, we can further reduce the lowest achievable delay from 0.45ps to 0.37ps. Actually, we can satisfy more stringent delay limits for 10 out of 14 test cases compared with the results in section 3.4.4. This clearly shows the effectiveness of our general post-grid routing algorithm incorporating non-tree structures.

4.5.4 Results on New Benchmarks

To further validate the effectiveness of our proposed non-tree algorithm, we generate 14 test cases from the original ones (the new test cases have their names starting with an "n"). These new test cases are generated as follows. We first sort the ports according to their minimum Elmore delays, which is the delay when a port is connected to its nearest source directly. Then we increase the capacitances of the first three ports in the list so that their minimum delays increase by at least 50%. Detailed results on these new test cases are shown in Table 4.2.

Table 4.1: Lowest achievable delays (with non-tree technique)

Test Cases	Capacitance (pf)	Wire Length (mm)	Delay (ps)	Runtime (s)
test1	2.70	10.8	0.37	0.19
test2	7.60	35.0	0.35	2.40
test3	5.45	23.7	0.55	4.08
test4	2.06	10.0	0.95	1.50
test5	0.74	3.0	0.78	0.10
test6	3.68	16.2	0.68	0.67
test7	3.68	17.6	1.32	1.55
test8	2.47	11.7	0.92	0.51
test9	6.52	32.8	1.95	6.81
test10	3.93	19.5	1.57	7.90
test11	2.41	10.8	0.75	5.21
test12	1.17	5.1	1.00	0.19
test13	4.54	20.6	0.65	3.70
test14	2.02	8.9	0.52	0.74

Total capacitance, total wire length, delay limits, running time and number of problematic ports are shown in column 2-5 respectively. The delay limits D is obtained empirically for all test cases. The second last column D_{min} in Table 4.2 shows the minimum delay of the problematic ports when they are connected to the nearest source *directly*. Therefore, these are the lower bound delays achievable using a tree structure. We can see from the comparsion in the last column that our non-tree approach can reduce further the delay by 23.4% on average. For *ntest3* and *ntest12*, our non-tree algorithm does not help much and it automatically degenerates into the original path expansion algorithm (the result is thus a set of trees) because of the high density of the ports especially in the surroundings of the problematic ports. For all the other test cases, our proposed non-tree approach can successfully generate a solution in which the maximum port delay is less than the lower bound delay shown in the second last column. This clearly demonstrates the effectiveness of our proposed non-tree algorithm.

4.5.5 Simulation Results

We also validate our results using hspice simulation. The slew of the input signals are set to be 10ps. The estimated slew of the circuits are shown in column three using $\sqrt{(2.2RC)^2 + (S_i)^2}$ according to [37], where RC is replaced by the largest Elmore delay of the circuit. Detailed results are shown in Table 4.3. As we can see from the simulation results, The delay and slew we calculated is very close to the simulation results. The correlation coefficient is over 99% between the simulated delay and calculated delay while it is over 96% between the simulated slew and calculated slew. This verifies the correctness of our method.

Table 4.2: Non-tree algorithm

Test Cases	Capacitance (pf)	Wire Length (mm)	Delay (x ps)	Runtime(s)	k	D_{min} (y ps)	Improvement ($\frac{y-x}{y}$%)
ntest1	2.4	11.1	0.45	0.1	3	0.68	33.8
ntest2	6.5	36.7	0.45	1.0	3	0.71	36.6
ntest3	5.1	25.2	0.60	3.3	3	0.51	-18.5
ntest4	2.0	11.3	1.00	1.0	3	1.26	20.8
ntest5	0.7	3.2	1.03	0.1	3	1.29	20.3
ntest6	3.5	17.6	0.66	0.5	3	1.25	47.0
ntest7	3.3	20.0	1.35	0.8	3	2.02	33.3
ntest8	2.1	12.5	1.30	0.2	3	1.98	34.4
ntest9	6.2	38.0	2.00	3.5	3	2.42	17.4
ntest10	3.6	22.1	1.80	3.1	3	2.33	22.6
ntest11	2.3	12.1	0.80	3.2	3	1.24	35.4
ntest12	0.9	5.5	1.70	0.1	3	1.65	-3.0
ntest13	3.5	20.7	1.25	0.4	3	1.35	7.2
ntest14	1.8	9.5	0.58	0.3	3	0.98	40.8
Ave.	3.1	17.5		1.3	3		23.4

Note: k denotes the number of problematic ports in the test case.

Table 4.3: Simulation results for non-tree

Test Cases	Calculated Results		Simulation Results	
	Delay (ps)	Slew (ps)	Delay (ps)	Slew (ps)
ntest1	0.45	10.05	0.45	10.07
ntest2	0.45	10.05	0.45	10.08
ntest3	0.60	10.09	0.60	10.11
ntest4	1.00	10.24	0.99	10.24
ntest5	1.03	10.25	1.03	10.23
ntest6	0.66	10.10	0.66	10.14
ntest7	1.35	10.43	1.35	10.43
ntest8	1.29	10.40	1.29	10.37
ntest9	2.00	10.93	2.00	11.15
ntest10	1.80	10.76	1.80	10.92
ntest11	0.80	10.15	0.80	10.17
ntest12	1.70	10.68	1.68	10.73
ntest13	1.25	10.37	1.25	10.33
ntest14	0.58	10.08	0.58	10.11

Chapter 5

Efficient Partitioning-based Extension

5.1 Introduction

Partitioning is a widely used technique to solve a difficult problem by partitioning the problem into smaller and, usually, easier ones. Moreover, one can expect a running time speed up by partitioning a problem into smaller ones, since it is easier and faster to solve these smaller problems. Partitioning has been successfully used in many placement, floorplanning and routing algorithms [2, 3, 16, 21, 22, 24, 30, 56, 68]. A drawback of this partitioning approach is that the solution quality will usually degenerate, as the algorithm will fail to consider the global information when dealing with individual subproblems. However, in our problem, this technique can be applied successfully without affecting the solution quality because that we can observe from the resulting clock network that a port will be connected to the nearer grid most of the time. Motivated by these works and with an objective of reducing the runtime, we also proposed our partitioning-based clock routing algorithm, in which we partition all the ports into several smaller clusters. For each cluster, we will run our clock routing algorithm to construct clock trees for that particular cluster. After connecting the ports in each cluster, we combine the results to form the final clock network. Fig. 5.1 shows a sample circuit which is divided into four regions using our partition technique.

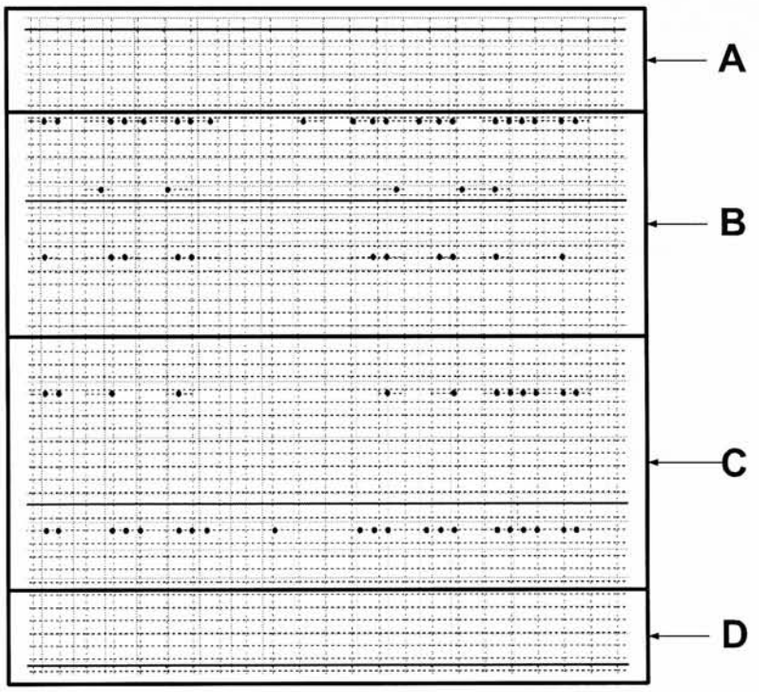


Figure 5.1: A sample circuit divided into four regions

5.2 Partition-based Extension

With the well designed source grid and multi-layer reserved tracks, a port can always be expected to connect to the nearest source grid as long as no delay violation occurs. Actually in the final clock network obtained using the path expansion algorithm described in Chapter 3 and Chapter 4, we found that a port would rarely navigate away from the nearer source grid and connect to a far away one. With this observation, we propose a technique to cluster all the ports into several small clusters, and employ the same path expansion algorithm as described in Chapter 3 and Chapter 4 on these smaller clusters to construct the clock network. Fig. 5.2 shows the flow of our main algorithm incorporating this partitioning technique.

The whole chip is divided into several smaller regions according to the source grid (either horizontal or vertical). The positions of the source grid[1] will be recorded. The intermediate positions between two successive source grids will be used as the guideline to split the chip into smaller partitions. After we get all the partitions, partitioning of

[1] If the source grid is horizontal, y coordinates will be recorded, otherwise x will be recorded.

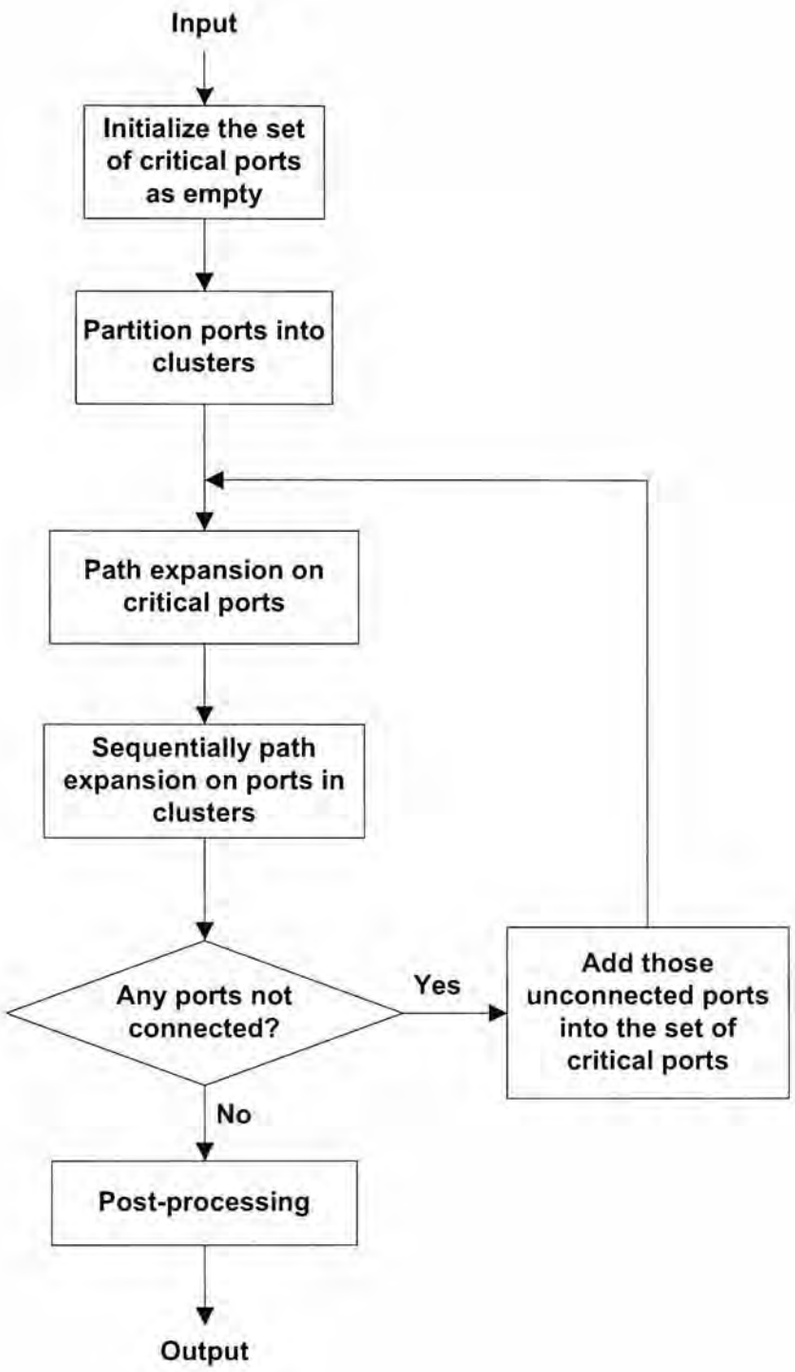


Figure 5.2: An overall flow of our partition-based algorithm

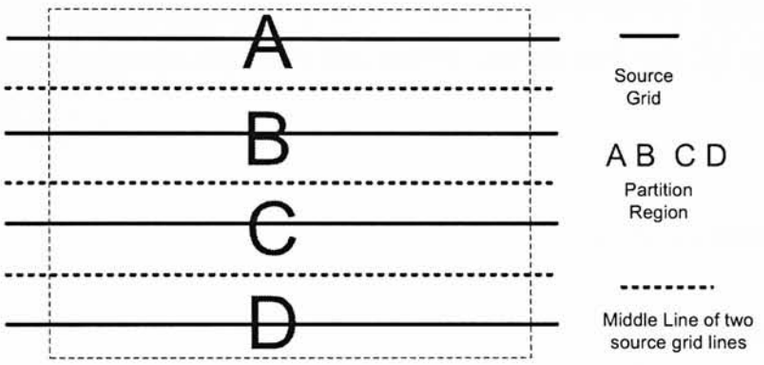


Figure 5.3: An example of partitioning

ports is rather straightforward: all the ports lies in the same partition will be grouped into the same cluster. We can also employ the same idea on these smaller clusters to further divide them into even smaller ones. For example, if the source grid is horizontal, we can further divide the cluster vertically, and vice versa. However, we found that this second level clustering might have adverse effect on the solution quality. This is mainly because we do not have a good guideline on where this second level division should be made. Unlike the first level division that we can safely partition the ports according to the positions of the source grids, there is no natural partitioning of the ports at the second level. This has been verified in our experiments that when incorporating this second level division, the path expansion algorithm usually fails to find a feasible solution under a particular delay limit. However, if we use only one level of partitioning, we can successfully satisfy the delay limit. Therefore in our implementation, we only group all the ports into smaller clusters horizontally or vertically (according to the direction of the source grid) without further dividing the clusters into sub-clusters. As shown in Fig. 5.3, we can partition this simple circuits into sub-regions A, B, C and D using the method described above.

After dividing the ports into clusters, we will sequentially employ the path expansion algorithm on those clusters to connect all the ports. This procedure is embedded into the main algorithm in line 9 while keeping all the other parts of the main algorithm unchanged. Since the original problem is divided into smaller ones, we can see significant improvement in running time compared with the original methods without using

partitioning technique. As detailed in section 5.3, the largest running time improvement is over 48% among all the test cases, while the average run time improvement is over 26%.

5.3 Experimental Results

5.3.1 Experiment Setup

In the experiments, we use the same settings as in section 3.4. The slew of the source signal is set to be 10ps. In the hspice simulation, 50% delay is measured to compare with the Elmore delay we have calculated. Detailed experimentalal results are explained as follows.

5.3.2 Running Time Improvement with Partitioning Technique

To demonstrate the effectiveness of our proposed partition-based path expansion algorithm, we compare its results with that without using this partitioning technique. The running times of both algorithm are shown in Table 5.1.

We can see that our proposed partition-based acceleration technique can further improve the running time by 26.1% on average while maintaining approximately the same solution quality (with minor improvement on average). The largest running time improvement is over 48%. These results clearly prove the effectiveness of this technique.

Table 5.1: Running time comparisons

Test Cases	Capacitance (pf)		Running Time (s)			Delay (ps)
	PE	PPE	PE x_1	PPE x_2	Improvement $\frac{x_1-x_2}{x_1}$ %	
test1	2.61	2.61	0.27	0.24	9.8	0.45
test2	9.68	9.67	2.72	1.39	48.9	1.15
test3	5.16	5.12	3.01	2.67	11.5	0.80
test4	4.01	4.00	2.89	1.51	48.0	1.35
test5	1.09	1.09	0.09	0.08	11.8	1.10
test6	5.73	5.73	0.68	0.48	29.2	1.25
test7	7.50	7.51	1.00	0.61	38.8	1.45
test8	4.45	4.45	0.50	0.37	25.2	1.80
test9	14.28	14.26	3.27	2.04	37.7	2.75
test10	8.60	8.59	6.92	7.15	-3.4	1.90
test11	4.93	4.92	2.95	1.76	40.5	1.05
test12	1.98	1.98	0.17	0.14	16.0	1.30
test13	7.20	7.19	0.93	0.62	33.0	1.10
test14	3.10	3.10	0.30	0.25	17.8	0.95
Ave.	5.74	5.73	1.84	1.38	26.1	

Note 1: PE denotes the original algorithm without this partitioning technique.

Note 2: PPE denotes the partitioning-based path expansion algorithm.

Chapter 6

Conclusion

In this thesis, we review the clock routing problem in the literature and describe many classical clock routing algorithms, such as H-tree, Method of Means and Medians (MMM), Geometric Matching Algorithm (GMA), Deferred Merge Embedding(DME), and Tree Growing (TG) algorithm on this topic. For the problem of post-grid clock routing that appears in high performance microprocessor designs today, we present an efficient delay-driven path expansion algorithm using the heap data structure to construct post-grid clock networks on reserved multi-layer metal tracks. We also propose a partition-based acceleration technique to further speed up the running time based on some key observations of this particular post-grid clock routing problem. Experimental results show the effectiveness of our proposed partitioning-based technique. We have compared our approach with the state-of-the-art algorithm on this problem and show that our algorithm can significantly improve over this work with a 24.6% reduction in wire capacitance and 23.6% reduction in wire length on average while maintaining very practical runtimes. Our algorithm also outperforms the previous method in terms of minimum achievable delays. To make our approach more robust and complete, we have extended the algorithm to allow non-tree structures in order to handle the ports with exceptionally large load capacitances. All our results are verified using hspice simulations. Our algorithm is expected to bring down the energy consumption and improve grid-to-port delay in post-grid clock networks. Our algorithm can be applied to high performance microprocessor designs in 45nm technology, and it may also be extended to applications for ASICs with hybrid clock structures.

□ **End of chapter.**

Bibliography

[1] *ISPD 2010 High Performance Clock Network Synthesis Contest.* http://www.sigda.org/ispd/contests/10/ispd10cns.html.

[2] S. Adya, S. Chaturvedi, J. Roy, D. Papa, and I. Markov. Unification of partitioning, placement and floorplanning. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-aided Design*, pages 550–557. IEEE Computer Society, 2004.

[3] C. Alpert and A. Kahng. Recent directions in netlist partitioning: a survey. *INTEGRATION, the VLSI journal*, 19(1-2):1–81, 1995.

[4] F. Anderson, J. Wells, and E. Berta. The core clock system on the next generation itanium1 microprocessor. In *IEEE International Solid-State Circuits Conference*, volume 1, pages 146–453, 2002.

[5] D. Bailey and B. Benschneider. Clocking design and analysis for a 600-MHz Alpha microprocessor. *IEEE Journal of Solid-State Circuits*, 33(11):1627–1633, Nov 1998.

[6] H. Bakoglu, J. Walker, and J. Meindl. A symmetric clock-distribution tree and optimized high-speed interconnections for reluced clock skew in ULSI and WSI circuits. *Proceedings of IEEE International Confrence of Computer Design*, pages 118–122, 1986.

[7] W. Bowhill, R. Allmon, S. Bell, E. Cooper, D. Donchin, J. Edmondson, T. Fischer, P. Gronowski, A. Jain, P. Kroesen, et al. A 300 MHz 64 b quad-issue CMOS RISC microprocessor. In *IEEE International Solid-State Circuits Conference*, pages 182–183. IEEE, 1995.

[8] P. Camporese, A. Deutsch, T. McNamara, P. Restle, and D. Webber. XY grid tree tuning method, Mar 2001. US Patent 6,205,571.

[9] P. Chan and K. Karplus. Computing signal delay in general RC networks by tree link partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(8):898–902, Aug. 1990.

[10] Chao T.-H. and Hsu Y.-C. and Ho, J.-M. Zero skew clock net routing. In *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pages 518–523, Jun 1992.

[11] Charles J. Alpert, Dinesh p. mehta, Sachin S. Sapatnekar. *Handbook of Algorithms for Physical Automation.* CRC Press, 2007.

[12] B. Chazelle. Filtering search: A new approach to query-answering. In *24th Annual Symposium on Foundations of Computer Science*, pages 122–132, 1983.

[13] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. In *29th Annual Symposium on Foundations of Computer Science*, pages 590–600, 1988.

[14] H. Chen, C. Yeh, G. Wilke, S. Reddy, H. Nguyen, W. Walker, and R. Murgai. A sliding window scheme for accurate clock mesh analysis. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 939–946, Nov. 2005.

[15] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. *IEEE Real-Time and Embedded Technology and Applications Symposium*, 0:408–417, 2006.

[16] T. Chen, Y. Chang, and S. Lin. IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs. In *IEEE/ACM International Conference on Computer-aided Design*, pages 159–164. IEEE, 2005.

[17] E. Chiprout. Fast flip-chip power grid analysis via locality and grid shells. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-aided Design*, pages 485–488. IEEE Computer Society, 2004.

[18] J. Cong, A. Kahng, C. Koh, and C. Tsao. Bounded-skew clock and steiner routing under elmore delay. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-aided Design*, pages 66–71. IEEE Computer Society, 1995.

[19] J. Cong, A. Kahng, and G. Robins. Matching-based methods for high-performance clock routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(8):1157–1169, 1993.

[20] J. Cong and C. Koh. Minimum-cost bounded-skew clock routing. In *IEEE International Symposium on Circuits and Systems*, volume 1, pages 215–218. IEEE, 1995.

[21] J. Cong, M. Romesis, and J. Shinnerl. Fast floorplanning by look-ahead enabled recursive bipartitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(9):1719–1732, 2006.

[22] J. Cong and Y. Zhang. Thermal-driven multilevel routing for 3-D ICs. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 121–126. ACM, 2005.

[23] D. Dobberpuhl, R. Witek, R. Allmon, R. Anglin, D. Bertucci, S. Britton, L. Chao, R. Conrad, D. Dever, B. Gieseke, et al. A 200-MHz 64-b dual-issue CMOS microprocessor. *IEEE Journal of Solid-State Circuits*, 27(11):1555–1567, 1992.

[24] A. Dunlop and B. Kernighan. A procedure for placement of standard cell VLSI circuits. *IEEE Transactions on Computer-Aided Design*, 4(1):92–98, 1985.

[25] M. Edahiro. Minimum skew and minimum path length routing in VLSI layout design. *NEC Research and Development*, 32(4):569–575, 1991.

[26] A. Fisher and H. Kung. Synchronizing large systolic arrays. Technical report, Carnegie-Mellon Univ., Pittsburgh, PA (USA). Dept. of Computer Science, 1982.

[27] M. Franklin and D. Wann. Asynchronous and clocked control structures for VLSI based interconnection networks. In *ACM SIGARCH Computer Architecture News*, volume 10, pages 50–59. IEEE Computer Society Press, 1982.

[28] R. Gupta, B. Tutuianu, and L. Pileggi. The Elmore delay as a bound for RC trees with generalized input signals. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(1):95–104, Jan 1997.

[29] R. Heald, K. Aingaran, C. Amir, M. Ang, M. Boland, A. Das, P. Dixit, G. Goulds-berry, J. Hart, T. Horel, et al. Implementation of a 3rd-generation SPARC V9 64 b microprocessor. In *IEEE International Solid-State Circuits Conference*, pages 412–413. IEEE, 2000.

[30] D. Huang and A. Kahng. Partitioning-based standard-cell global placement with an exact objective. In *Proceedings of the 1997 International Symposium on Physical Design*, pages 18–25. ACM, 1997.

[31] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 37–46. Society for Industrial and Applied Mathematics, 2003.

[32] Jackson, Srinivasan, and Kuh. Clock routing for high-performance ICs. *Proceedings of the 27th Design Automation Conference*, 0:573–579, 1990.

[33] A. Kahng, J. Cong, and G. Robins. High-performance clock routing based on recursive geometric matching. In *Proceedings of the 28th Design Automation Conference*, pages 322–327, 1991.

[34] A. Kahng and G. Robins. *On optimal interconnections for VLSI*. Kluwer Academic Pub, 1995.

[35] A. Kahng, C. Tsao, and D. Huang. On the bounded-skew clock and steiner routing problems. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 508–513. ACM Press, 1995.

[36] A. Kahng and C.-W. A. Tsao. Planar-DME: a single-layer zero-skew clock tree router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(1):8 –19, Jan 1996.

[37] C. Kashyap, C. Alpert, F. Liu, and A. Devgan. Closed-form expressions for extending step delay and slew metrics to ramp inputs for RC trees. *IEEE Transaction-*

s on Computer-Aided Design of Integrated Circuits and Systems, 23(4):509–516, April 2004.

[38] B. K.D. and K. A.B. Zero-skew clock routing trees with minimum wirelength. In *Proceedings of Fifth Annual IEEE International ASIC Conference and Exhibit*, pages 17–21, Sep. 1992.

[39] N. Kurd, J. Barkarullah, R. Dizon, T. Fletcher, and P. Madland. A multigigahertz clocking scheme for the Pentium(R) 4 microprocessor. *IEEE Journal of Solid-State Circuits*, 36(11):1647–1653, Nov. 2001.

[40] W. Lam, J. Jam, C. Koh, V. Balakrishnan, and Y. Chen. Statistical based link insertion for robust clock network design. In *Proceedings of the 2005 IEEE/ACM International Conference on Computer-aided Design*, pages 588–591. IEEE Computer Society, 2005.

[41] W. Lam and C. Koh. Process variation robust clock tree routing. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 606–611. ACM, 2005.

[42] S. Lin and C. Wong. Process-variation-tolerant clock skew minimization. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided Design*, pages 284–288. IEEE Computer Society Press, 1994.

[43] Y. Liu, S. Nassif, L. Pileggi, and A. Strojwas. Impact of interconnect variations on the clock skew of a gigahertz microprocessor. In *Proceedings of the 37th Annual Design Automation Conference*, pages 168–171. ACM, 2000.

[44] B. Lu, J. Hu, G. Ellis, and H. Su. Process variation aware clock tree routing. In *Proceedings of the 2003 International Symposium on Physical Design*, pages 174–181. ACM, 2003.

[45] T. Mittal and C. Koh. Cross link insertion for improving tolerance to variations in clock network synthesis. In *Proceedings of the 2011 International Symposium on Physical Design*, pages 29–36. ACM, 2011.

[46] M. Mori, H. Chen, B. Yao, and C. Cheng. A multiple level network approach for clock skew minimization with process variations. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, pages 263–268. IEEE Press, 2004.

[47] M. Mori, H. Chen, B. Yao, and C.-K. Cheng. A mulitple level network approach for clock skew minimization with process variations. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 263–268, 2004.

[48] U. Padmanabhan, J. Wang, and J. Hu. Statistical clock tree routing for robustness to process variations. In *Proceedings of the 2006 International Symposium on Physical Design*, pages 149–156. ACM, 2006.

[49] U. Padmanabhan, J. Wang, and J. Hu. Robust clock tree routing in the presence of process variations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1385–1397, 2008.

[50] D. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. Harvey, H. Hofstee, C. Johns, et al. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *IEEE Journal of Solid-State Circuits*, 41(1):179–196, Jan. 2006.

[51] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital integrated circuits*. Prentice Hall, 2002.

[52] A. Rajaram, J. Hu, and R. Mahapatra. Reducing clock skew variability via crosslinks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1176–1182, Jun. 2006.

[53] A. Rajaram and D. Pan. Fast incremental link insertion in clock networks for skew variability reduction. In *Proceedings of the 7th International Symposium on Quality Electronic Design*, pages 79–84. IEEE Computer Society, 2006.

[54] A. Rajaram and D. Pan. Variation tolerant buffered clock network synthesis with cross links. In *Proceedings of the 2006 International Symposium on Physical Design*, pages 157–164. ACM, 2006.

[55] A. Rajaram, D. Pan, and J. Hu. Improved algorithms for link-based non-tree clock networks for skew variability reduction. In *Proceedings of the 2005 International Symposium on Physical Design*, pages 55–62. ACM, 2005.

[56] A. Ranjan, K. Bazargan, S. Ogrenci, and M. Sarrafzadeh. Fast floorplanning for effective prediction and construction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(2):341–351, 2001.

[57] P. Restle, C. Carter, J. Eckhardt, B. Krauter, B. McCredie, K. Jenkins, A. Weger, and A. Mule. The clock distribution of the power4 microprocessor. In *IEEE International Solid-State Circuits Conference*, volume 1, pages 144–145. IEEE, 2002.

[58] P. Restle, T. McNamara, D. Webber, P. Camporese, K. Eng, K. Jenkins, D. Allen, M. Rohn, M. Quaranta, D. Boerstler, et al. A clock distribution network for microprocessors. *IEEE Journal of Solid-State Circuits*, 36(5):792–799, 2001.

[59] P. Restle, A. Ruehli, and S. Walker. Multi-GHz interconnect effects in microprocessors. In *Proceedings of the 2001 International Symposium on Physical Design*, pages 93–97, Apr. 2001.

[60] Restle P.J. and McNamara, T.G. and Webber, D.A. and Camporese, P.J. and Eng, K.F. and Jenkins, K.A. and Allen, D.H. and Rohn, M.J. and Quaranta, M.P. and Boerstler, D.W. and others. A clock distribution network for microprocessors. In *IEEE Journal of Solid-State Circuits*, pages 184–187, 2000.

[61] A. Ruehli. *Circuit analysis, simulation, and design: general aspects of circuit analysis and design*. North-Holland Publishing Co. Amsterdam, The Netherlands, 1986.

[62] R. Senthinathan, S. Fischer, H. Rangchi, and H. Yazdanmehr. A 650-MHz, IA-32 microprocessor with enhanced data streaming for graphics and video. *IEEE Journal of Solid-State Circuits*, 34(11):1454–1465, 1999.

[63] G. Shamanna, N. Kurd, J. Douglas, and M. Morrise. Scalable, sub-1W, sub-10ps clock skew, global clock distribution architecture for Intel Core i7/i5/i3 microprocessors. In *IEEE Symposium on VLSI Circuits (VLSIC)*, pages 83–84, June 2010.

[64] R. Shelar. An algorithm for routing with capacitance/distance constraints for clock distribution in microprocessors. In *Proceedings of the 2009 International Symposium on Physical Design*, pages 141–148, 2009.

[65] R. Shelar. Routing with constraints for post-grid clock distribution in microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(2):245–249, Feb. 2010.

[66] N. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, 3rd edition, 1998.

[67] H. Su and S. Sapatnekar. Hybrid structured clock network construction. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, pages 333–336. IEEE Press, 2001.

[68] P. Suaris and G. Kedem. An algorithm for quadrisection and its application to standard cell placement. *IEEE Transactions on Circuits and Systems*, 35(3):294–303, 1988.

[69] H. Tian, W. Tang, E. F. Young, and C. Sze. Grid-to-ports clock routing for high performance microprocessor designs. In *Proceedings of the 2011 international symposium on physical design*, pages 21–28. ACM, 2011.

[70] C. Tsao and C. Koh. UST/DME: a clock tree router for general skew constraints. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 7(3):359–379, 2002.

[71] R.-S. Tsay. An exact zero-skew clock routing algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(2):242–249, Feb. 1993.

[72] G. Venkataraman, Z. Feng, J. Hu, and P. Li. Combinatorial algorithms for fast clock mesh optimization. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-aided Design*, pages 563–567. ACM, 2006.

[73] G. Venkataraman, Z. Feng, J. Hu, and P. Li. Combinatorial algorithms for fast clock mesh optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(1):131–141, Jan. 2010.

[74] G. Venkataraman, N. Jayakumar, J. Hu, P. Li, S. Khatri, A. Rajaram, P. McGuinness, and C. Alpert. Practical techniques to reduce skew and its variations in buffered clock networks. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, pages 592–596. IEEE, 2005.

[75] G. Venkataraman, C. Sze, and J. Hu. Skew scheduling and clock routing for improved tolerance to process variations. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 594–599. ACM, 2005.

[76] G. Wilke and R. Murgai. Design and analysis of tree+ local meshes clock architecture. In *International Symposium on Quality Electronic Design*, pages 165–170. IEEE, 2007.

[77] L. Xiao, Z. Xiao, Z. Qian, Y. Jiang, T. Huang, H. Tian, and E. Young. Local clock skew minimization using blockage-aware mixed tree-mesh clock network. In *2010 IEEE/ACM International Conference on Computer-Aided Design*, pages 458–462, Nov. 2010.

[78] T. Xue and E. S. Kuh. Post routing performance optimization via multi-link insertion and non-uniform wiresizing. In *In IEEE-ACM International Conference on Computer-Aided Design*, pages 575–580, 1995.

[79] J.-S. Yang, A. Rajaram, N. Shi, J. Chen, and D. Pan. Sensitivity based link insertion for variation tolerant clock network synthesis. In *International Symposium on Quality Electronic Design*, pages 398–403, Mar. 2007.

[80] X. Ye, P. Li, M. Zhao, R. Panda, and J. Hu. Analysis of large clock meshes via harmonic-weighted model order reduction and port sliding. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design*, pages 627–631. IEEE Press, 2007.

[81] Q. Zhu and W.-M. Dai. Perfect-balance planar clock routing with minimal path-length. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 473–476, Nov, 1992.