Perfect-Balance Planar Clock Routing with Minimal Path Length

Qing Zhu Wayne W.M. Dai

UCSC-CRL-93-17
supercedes UCSC-CRL-92-12
March 26, 1993

Board of Studies in Computer Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064

ABSTRACT

The design of high speed digital VLSI circuits prefers that the clock net is routed on the metal layer with the smallest RC delay. This strategy not only avoids the difficulties of having different electrical parameters on different layers, but also eliminates the delay and attenuation of the clock signal through vias. The clock phase-delay is also decreased. In this paper, we present a novel algorithm, based on hierarchical max-min optimization, to construct a planar clock tree which can be embedded on a single metal layer. The clock tree achieves equal path length—the length of the path from the clock source to each clock terminal is exactly the same. In addition, the path length from the source to clock terminals is minimized. Some examples including industrial benchmarks have been tested and the results are promising. We further optimize the geometry of the clock tree to minimize both the skew and path delay of the clock signal while maintaining the planarity of the clock network. Some premilinary results are promising which achieve near zero skew by using SPICE simulation.

Keywords: clock routing, planar routing, equal path length, max-min optimization, Steiner tree

Contents

List of Figures

List of Tables

1 Introduction

There is a general trend in digital VLSI circuits towards more registers and higher clock frequencies. There are two major concerns in high-performance clock routing: minimizing clock skew and routability. For optimal system performance, the clock signal must reach each register at exactly (or almost exactly) the same time. The routability of the clock net also becomes an urgent concern in dense VLSI chips with thousands of clocked elements which may be unevenly distributed across a major area of the chip. The design of high speed digital VLSI circuits prefers that the clock net is routed on the metal layer with the smallest RC delay. This strategy not only avoids the difficulties of having different electrical parameters on different layers, but also eliminates the delay and attenuation of the clock signal through vias. The path delay of the clock signal from the source to the clocked elements is also decreased. For double metal layers in $1\mu m$ technology, typical resistance and capacitance values are 25 m$\Omega/\Box$ and 0.015 fF$/\mu m$ on metal two, and 57 m$\Omega/\Box$ and 0.029 fF$/\mu m$ on metal one. A typical resistance for a via is 410 m$\Omega/\Box$. Recently, Digital Equipment Corporation has succeeded in implementing a new microprocessor operating up to 200 MHz [8]. Its clock network is laid almost entirely on one metal layer, and is driven by a huge buffer near the clock source.

The "H" clock tree is widely used in the IC industry [2]. However, it is only applicable for symmetric arrays of logic elements. There have been two generalized H-tree algorithms proposed that try to distribute the clock to elements with arbitrary positions [13, 14]. In [13], a generalized H-tree is constructed in a top-down fashion by the method of means and medians (MMM). Although the difference in path lengths from clock source to clock terminals is bounded by $O(\frac{1}{\sqrt{n}})$ on the average case, it may be as large as half the diameter of the chip [14]. Another algorithm constructs a clock tree based on recursive geometric matching (RGM) in a bottom-up sequence [14]. This algorithm always yields clock trees with perfectly balanced path lengths for trees of two, three, or four terminals. But no bound is given for the general case. An improved algorithm [22] considers Elmore delay balance instead of geometric length balance but adopts a bottom-up process similar to that of [14]. This algorithm achieves delay balance by enlongating wires that have smaller delays. Another algorithm in [6] improves the Elmore delay matching method by considering the minimization of the total wire length. But this algorithm yields minimal wirelength only for a given connection topology and linear delay model. The major problem with these algorithms is that they create many overlaps in the clock networks. Multiple routing layers must be used to implement such a non-planar network. The performance and routability of the clock network are seriously sacrificed. So, the application of these algorithms to the real design of clock distribution for VLSI systems is limited.

In this paper, we present a novel algorithm to construct a *planar clock tree* which can be embedded on a single metal layer. The clock tree achieves *equal path length*—the length of the path from the clock source to each clock terminal is *exactly the same*. This is important since the wire length still dominates the timing delay of a wire [5, 1]. Such a planar clock tree with equal path length provides a good initial clock topology for designers to adjust for minimum skew. We can further optimize the geometry of the clock tree to minimize both the skew and path delay of the clock signal while maintaining the planarity of the clock network. Achieving exact zero skew of a clock network is still an engineering effort which finally relies on capacitance calculations and detailed interconnect simulation [15].

The key features of our algorithm are described as follows.

- The algorithm always constructs a planar clock tree.
- The lengths of the paths from the the clock source to each of the clock terminals are exactly the same.
- The path length from the source to clock terminals is the minimum.
- The clock source can be at arbitrary location in the layout.

The first three features of the algorithm are proved in Section 4. The last feature allows the flexible position of the clock source. In VLSI systems, the clock source may be a pad at the side of the chip/module frame, or may be a clock generator inside the chip/module. Our algorithm roots the clock tree directly at the source unlike the previous methods [13, 14, 22, 2] which must route from the clock source to the root of the tree after the tree is completed.

The organization of this paper is as follows. Section 2 defines the clock tree as a planar equal path length Steiner tree. In Section 3, we present our algorithm based on max-min optimization. We also describe the recursive divide-and-conquer paradigm of the algorithm. The time complexity of this algorithm is given in section 4. We prove in Section 5 that the algorithm always yields a planar clock tree with perfectly balanced path lengths, regardless the source position and the distribution of terminals. In addition, the length of the path from the clock source to the terminals is minimized. In section 6, we show an asymptotic bound of the total wire length of the clock tree as the number of clock terminals increases. We describe in section 7 an algorithm of sizing the planar equal path length clock tree to achieve zero skew based on Elmore delay. Experimental results are given in Section 8 we compare our results with previous algorithms, also including some preliminary results on zero skew sizing of the planar clock tree. Some concluding remarks are given in section 9, where we show that the algorithm can be used to construct a planar equal path length Steiner tree not only in Manhanttan space but also in Euclidean space.

A short version of this paper has been presented at ICCAD-92 [24].

2 Planar Equal Path length Steiner Tree

A clock *source* and a set of clock terminals (*sinks*) form a clock net in a VLSI chip or a multi-chip module. A clock tree, T, of the clock net is a Steiner tree which connects all sinks with the source. The length of a branch connecting two vertices in the tree is taken to be the Manhattan distance. The length of the path from source o to any sink t_j, is merely the sum of the lengths of all branches on the unique path from o to t_j. The cost of the clock tree is the sum of the lengths of its branches. The clock tree considered here is a special Steiner tree called *planar equal path length Steiner tree*. We give the definition as follows.

Planar Equal Path Length Steiner Tree Problem: 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 sink points are exactly the same. Note that, in general, this problem is NP-complete.

3 New Algorithm

3.1 Overview and Basic Concepts

We propose a new algorithm to construct a planar equal path length Steiner tree. The algorithm is a single-tree growth method since sinks are added to the tree one at a time.

The algorithm produces a series of partial trees $\{T_1, T_2, T_3, \ldots, T_n\}$, where T_i is the partial tree in which the first i sinks are connected. T_n is the final Steiner tree in which all n sinks are connected. The basic ideas of this algorithm are as follows.

First, we choose the sink that has the maximal Manhattan distance to the source, and connect it to the source. This forms the first partial tree, T_1, as shown in Figure 3.1(a)). As the partial tree grows, all sinks are classified into two types. A sink is called a *free sink* if it has not yet been connected to the partial tree; a sink is called *connected sink* if it has been added to the tree. At any partial tree T_i ($1 \leq i < n$), we select a free sink and connect it to a branch of the tree maintaining equal path length from the source to the sink. This sink then becomes a connected sink. This may split the branch in two since a Steiner point is inserted on it. An example is shown in Figure 3.1(b), where sink t_2 is chosen and connected to branch b_1 in the tree T_1. The new Steiner point s is said to be a *balance point* since it has equal Manhattan distance to sink t_1 and sink t_2. A balance point s_{jk} for a free sink t_j exists on a branch b_k, if the Manhattan distance from s_{jk} to t_j is equal to the path length from s_{jk} to a connected sink in T_i. The balance point s_{jk} is called a *feasible balance point* of t_j, if a straight line connecting t_j and s_{jk} crosses no other branches. Connecting t_j to a feasible balance point on T_i forms a new tree T_{i+1} in which both planarity and path length balance are maintained.

Figure 3.1: Partial Steiner Trees

Usually a free sink t_j has several feasible balance points on branches in T_i. The feasible balance point with the minimum Manhattan distance to t_j is called *minimal balance point* of t_j in T_i. The Manhattan distance between t_j and its minimal balance point is called *minimal balance distance* of t_j in T_i. In Figure 3.2(a), sink t_3 has two feasible balance points: s_1 on branch b_1 and s_2 on branch b_2. The point s_2 is the minimal balance point since its distance to t_3 is shorter than s_1's. The distance between t_3 and s_2 is the minimal balance distance of t_3.

Another interesting phenomenon is the routing order of free sinks. For example in Figure 3.1(c), if the sink t_4 is connected to b_1 before sink t_2, the sink t_2 cannot find any balance point on the new branch b_4. Sink t_2 has a balance point s_2 on branch b_1 but no feasible balance point. If t_2 is connected to s_2 on b_1, a crossing over b_4 happens which results in a non-planar tree. In addition, the sink t_4 has a longer balance distance to b_1 than to b_2 (see Figure 3.1(b)). Therefore, the tree cost also increases. So, sink t_2 must be connected before t_4.

Two rules play key roles in our algorithm.

- **Min-rule**: always connect a free sink to its minimal balance point.
- **Max-rule**: at each stage, always select the free sink whose minimal balance distance is maximized among all remaining free sinks.

The Max-rule guides the routing order for free sinks. This rule ensures the planarity of the equal path length Steiner tree without iteration. The Min-rule reduces the tree cost efficiently.

Our algorithm grows the partial tree by iteratively applying the Max-Min rules. At a partial tree T_i, a free sink with the maximal minimal-balance-distance among all free sinks is connected to its minimal balance point. This free sink then becomes a connected sink and T_i is expanded to T_{i+1}. For example in Figure 3.1(b) and Figure 3.1(c), t_2, t_3, and t_4 all have the minimal balance points on branch b_1 but t_2 has the maximal minimal-balance-distance, therefore t_2 is connected before t_3 or t_4.

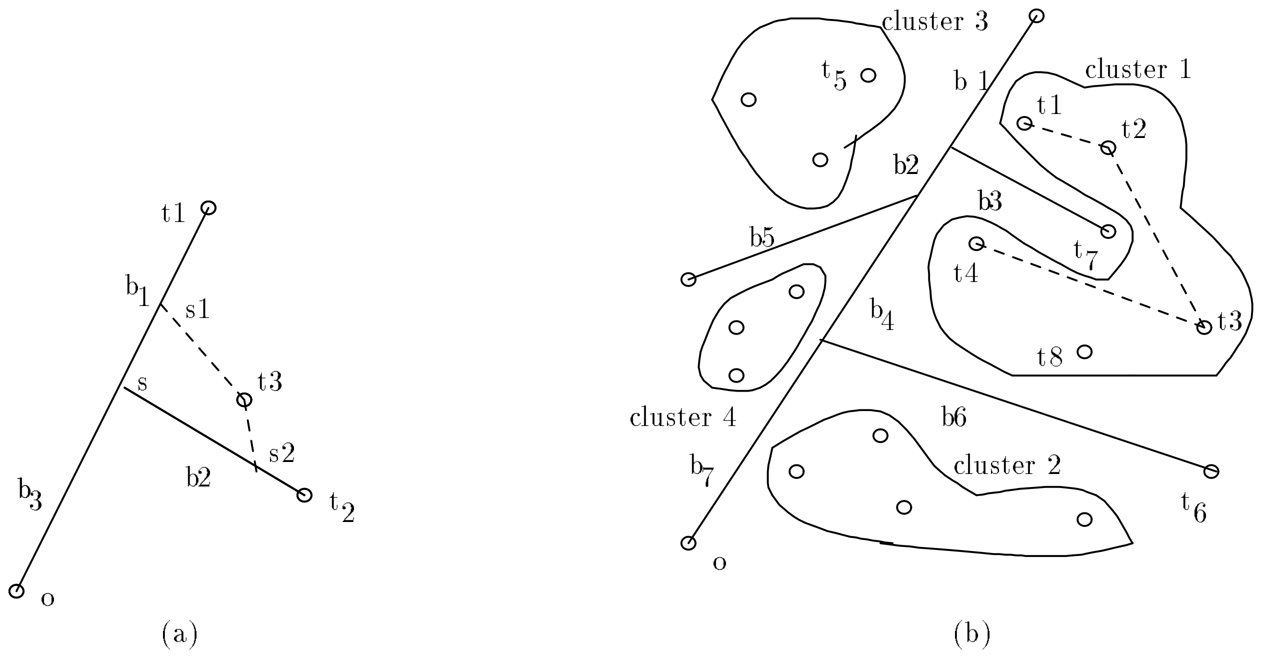

Figure 3.2: (a) Free sink t_3 has two feasible balance points on two bounding branches b_1 and b_2. (b) Partial tree T_4 separates free sinks into four independent clusters; t_1 and t_4 are reachable ; however, t_1 and t_5 are not reachable.

3.2 Hierarchical Max-Min Optimization

In order to explain our algorithm, it is useful to define a binary relation, called *reachable*, on the set of free sinks S. The relation is based on the concept of visibility. Two free sinks $a, b \in S$ are *visible* ($a \Leftrightarrow b$) if a straight line between them crosses no tree branches. Two free sinks $a, b \in S$ are defined to be *reachable* if either:

- $a \Leftrightarrow b$, or
- there exists a sequence of free sinks $(p_1, p_2, \ldots, p_k) \in S$ such that $a \Leftrightarrow p_1, p_k \Leftrightarrow b$, and $p_i \Leftrightarrow p_{i+1}$ for $1 \leq i < k$.

For example in Figure 3.2(b), t_1 is not visible to t_4 since the branch b_3 lies between them. But t_1 can be reachable to t_4, by way of the path transferring the visibility to other free sinks: $t_1 \Leftrightarrow t_2 \Leftrightarrow t_3 \Leftrightarrow t_4$. However in Figure 3.2(b), t_1 and t_5 are unreachable. Reachable is an equivalence relation and partitions S into a set of equivalence classes which we will refer to as *clusters.*

The partial tree T_i partitions the free sinks into clusters. As described above, free sinks in one cluster are reachable to each other, and sinks in different clusters are not. Each cluster is bounded by a set of tree branches. Only on these branches, can a free sink in the cluster possibly find feasible balance points. These branches are called the *bounding branches* of the free sinks in this cluster. In Figure 3.2(b), the free sinks are partitioned into four clusters by the tree T_4. For the sinks in the cluster 1, the bounding branches are $\{b_1, b_2, b_3, b_4, b_6\}$. The bounding branches are $\{b_6, b_7\}$ for cluster 2, $\{b_1, b_2, b_5\}$ for cluster 3, and $\{b_4, b_5, b_7\}$ for cluster 4. It is impossible for a free sink to find a feasible balance point on any branch not in its set of bounding branches because it would require a crossing a bounding branch.

The algorithm is suited for parallel routing by applying the max-min rules locally on each cluster of free sinks at T_i. Free sinks in different clusters cannot conflict with one another in the routing order, because they are separated by the tree. Therefore, each cluster selects one free sink to be connected at each iteration. This sink only needs to be compared with other free sinks in the same cluster (Max Rule). When looking for the minimal balance point, a free sink needs only to check feasible balance points on the bounding branches for its cluster (Min-Rule).

When a free sink is connected to the partial tree, the new branch may partition the free sinks of the original cluster into new clusters. Shown in Figure 3.2(b), when branch b_6 connects sink t_6 to the tree, it separates the free sinks of the original cluster into cluster 1 and cluster 2. However, connecting t_7 to form branch b_3 does not further partition cluster 1.

Initially, all sinks are in one cluster. As the partial tree expands, it separates the remaining free sinks into more and more clusters. As a result, the tree grows faster and faster by locally applying max-min rules on each cluster. This divide-and-conquer paradigm accelerates the algorithm significantly. This is verified in the experiments.

Growing the partial tree by using the Max-Min rules on each cluster is a hierarchical *max-min optimization* process. We formulate this process of applying Max-Min rules in a cluster C as

$$\max_j\{\min_i\{d(t_j, b_i), b_i \text{ is a bounding branch of } t_j\}, t_j \in C\}$$

The term $d(t_j, b_i)$ is the Manhattan distance of free sink t_j to a feasible balance point on bounding branch b_i. In Section 4, we prove that our algorithm, based on hierarchical max-min optimization, not only maintains a planar equal path length Steiner tree, but also yields the minimum path length from the source to the sinks.

An example is shown in Figure 3.3(a), where o is the source and the other points are the sinks. Proceeding from Figure 3.3(b) to Figure 3.3(f), the algorithm grows the tree by using the max-min optimization on free sink clusters. The final result shown in Figure 3.3(g) is a planar equal path length Steiner tree. The hierarchical cluster partitioning process of this example is revealed in Figure 3.4, which recursively separates the free sinks into smaller clusters. Each node in Figure 3.4 represents a cluster of free sinks. At each step, the max-min optimization is applied to each cluster. The notation [p, q] appearing inside each node, means the cluster initially consisted of p free sinks and was partitioned into subclusters after q sinks were connected.

The source in Figures 3.3(a-g) was placed at the side of the problem. The clock tree shown in Figure 3.3(h) results from choosing the source inside the set of sinks.

(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 3.3: Example of using the algorithm to grow a planar equal path length Steiner tree. (a) 18 sinks and point o is the source on the left-bottom side. (b) – (f) At a level, a free sink in each cluster is connected to the partial tree. (g) The final tree where the source is on the side shown in (a). (h) The tree where the source o is set inside the sinks.

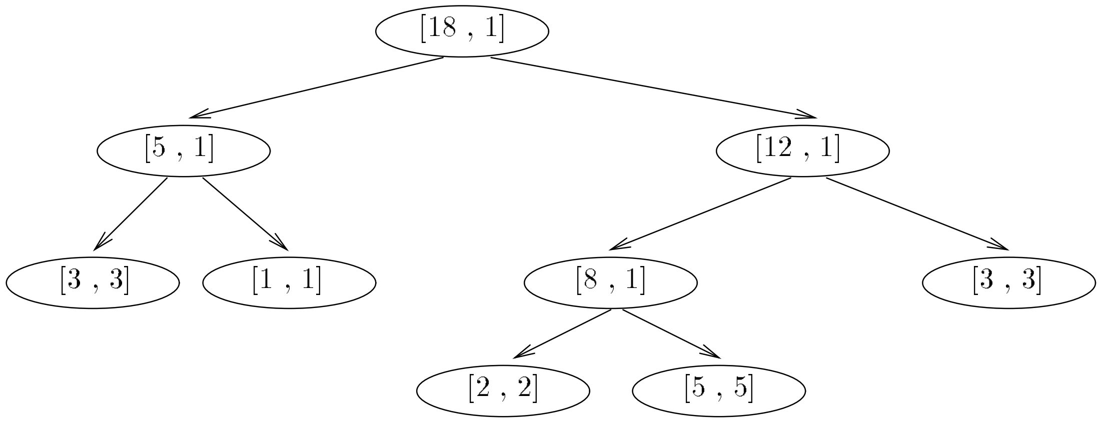

Figure 3.4: Top down free sink cluster partition

Our algorithm is summarized in the following pseudo-code.

Planar clock routing algorithm based on hierarchical max-min optimization

```
Input: a source $s$, and a set of sinks $D$;
Output: a planar equal path length Steiner tree $T$.
Procedure **PlanarClockRouter**($s, D, T$) {
    $C_0 = D$;
    $T = (\{s\}, \emptyset)$;
    CreateBranch($C_0, T$);
}

Procedure **CreateBranch**($C, T$) {
    Find $t^*, b^*$ such that
    $d(t^*, b^*) = \max_j\{\min_i\{d(t_j, b_i), b_i$ is a bounding branch of $t_j\}, t_j \in C\}$;
    Create a branch from $t^*$ to its minimal balance point on $b^*$ resulting in new $T$;
    if $(C - \{t^*\} \neq \emptyset)${
        Partition $C - \{t^*\}$ into subclusters $C_1, C_2, \ldots, C_k$ using the reachable relation;
        for each $i \in [1, k]$
            CreateBranch($C_i, T$);
    }
}
```

4 Time Complexity

We define the tree growing by one level as follows: for each cluster C_i, select a free sink $t_i^* \in C_i$ to be connected next according to the max-min rule, create a branch connecting t_i^* to the tree, and update the minimal balance points for the remaining free sinks in C_i. Recall that each cluster can be processed independently. Let n_i be the number of free sinks in C_i. For each cluster C_i, the branch creation can be done in $O(1)$ time, and the selection of t_i^* and the updating of minimal balance points for remaining free sinks can be done in

$O(n_i)$ time. So, the total time required at each level is $\sum_i O(n_i) = O(n)$, where n is the total number of sinks. So overall the time complexity is $O(l \cdot n)$, where l is the number of levels of the tree growing.

In the worst case, at each level, branches are connected to the tree in such a way that no further partition of the clusters occurs. This implies $l = n$. So, the worst-case running time of the algorithm is $O(n^2)$. However, l is usually much smaller than n for large examples. For example, $l = 21$ and $n = 269$ for Primary1 benchmark example, and $l = 28$ and $n = 603$ for Primary2.

5 Correctness

We mention in the introduction section that the algorithm based on max-min optimization guarantees to construct a planar equal path length Steiner tree, regardless of the source position and the number of sinks. The following three lemmas establish the proof of the theorem about the correctness of the algorithm. To make the proofs more straight forward, at each step, we assume that the algorithm is used in a flat form where every time a free sink with the maximal minimal-balance-distance (among all free sinks) is connected to its minimal balance point. However, we can easily extend the following lemmas to hold for the hierarchical (cluster) form of the algorithm.

We define a *leaf bounding branch* as a bounding branch which connects to a sink.

Lemma 1: *A free sink can find the minimal balance point only on a leaf bounding branches.*

Proof: Assume a free sink t has a minimal balance point, s, on a non-leaf bounding branch $\overline{s_0 s'}$. That is, s lies in between s_0 and s' (see Figure 5.1). Assume, without the loss of generality, point s_0 is nearer to the source o than s' in the partial tree. Since $\overline{s_0 s'}$ is a non-leaf bounding branch, there are at least two branches other than $\overline{s_0 s'}$ connecting to s'. Without the loss of generality, let t' be one of the sinks connected to s' through a sequence of Steiner points s_1, s_2, ..., $s_k\}$. Because s is a balance point:

$$\|(s,t')\| = \|(s,s')\| + \|(s',s_1)\| \ldots + \|(s_k,t')\|$$

where $\|(x,y)\|$ represents the Manhattan distance between two points x and y. It follows that $\|(s,t)\| > \|(s',t')\|$. According to the max rule t should be connected before t', which leads to contradiction. $\square$

Lemma 2: *For a free sink t_j and a leaf bounding branch b_k, $b_k = (s_k, t_k)$ where b_k connects a sink t_k, t_j has a feasible balance point on b_k if and only if*

$$\|(s_k,t_j)\| \leq \|(s_k,t_k)\| \tag{5.1}$$

where $\|(x,y)\|$ represents the Manhattan distance between two points x and y.

Proof: If a feasible balance point say s_j of free sink t_j exists on a leaf branch b_k, then $\|(s_j,t_j)\| = \|(s_j,t_k)\|$ (see Figure 5.2). If s_j is s_k, by definition $\|(s_k,t_j)\| = \|(s_k,t_k)\|$ and (5.1) is true. Otherwise, $\Delta s_k s_j t_j$ forms a triangle. For a triangle $\Delta s_k s_j t_j$ in **Manhattan distance**, the following constraint holds on side lengths.

$$\|(s_k,t_j)\| \leq \|(s_k,s_j)\| + \|(s_j,t_j)\| \tag{5.2}$$

Since $\|(s_j,t_j)\| = \|(s_j,t_k)\|$, (5.2) can be rewritten as

$$\|(s_k,t_j)\| \leq \|(s_k,s_j)\| + \|(s_j,t_k)\| = \|(s_k,t_k)\| \tag{5.3}$$

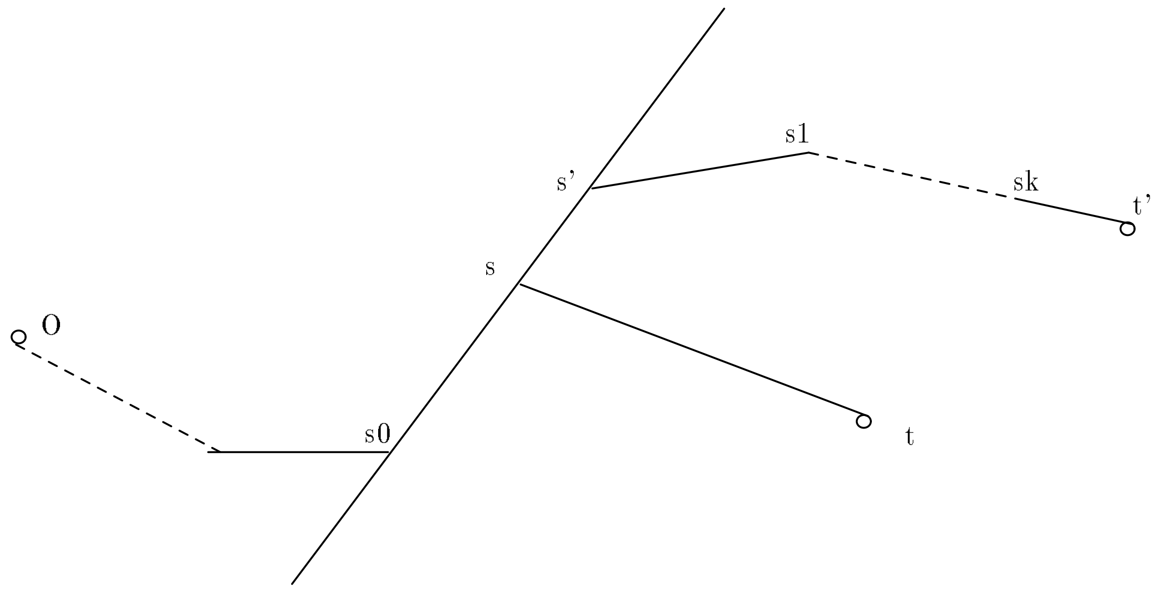

Figure 5.1: Sink t has the minimal balance point s on a non-leaf bounding branch $\overline{s_0 s'}$.

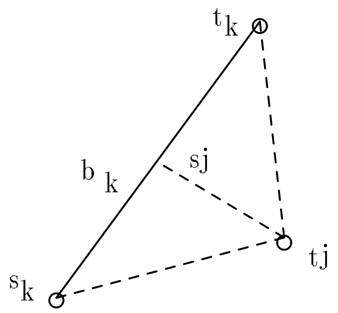

Figure 5.2: Free sink t_j has a feasible balance point s_j on a bounding leaf branch b_k.

On the other side, if $\|(s_k, t_j)\| \leq \|(s_k, t_k)\|$, we prove that a feasible balance point of t_j exists on the leaf bounding branch b_k. Assume a point s_j moves from s_k to t_k along b_k (see Figure 5.2), and we define $\delta(s_j) = \|(s_j, t_j)\| - \|(s_j, t_k)\|$. When s_j starts at s_k, we have: $\delta(s_k) = \|(s_k, t_j)\| - \|(s_k, t_k)\|$. Because (5.1) is satisfied, $\delta(s_k) \leq 0$. When s_j arrives at t_k, we have: $\delta(t_k) = \|(t_k, t_j)\| - \|(t_k, t_k)\|$. Because $\|(t_k, t_k)\|$ is zero, $\delta(t_k) > 0$. Since $\delta(s_j)$ changes continuously from negative to positive when s_j moves from s_k to t_k along a straight line b_k, there should be a point s_j^* on b_k such that $\delta(s_j^*) = 0$ or $\|(s_j^*, t_j)\| = \|(s_j^*, t_k)\|$. Point s_j^* is a feasible balance point of t_j on b_k. □

Lemma 3: *Every free sink can find at least one feasible balance point at any partial tree T_i $(1 \leq i < n)$.*

Proof: We use the mathematical induction according to the series of partial trees $\{T_1, T_2, \ldots, T_n\}$, where T_i means i sinks have been connected to the partial tree.

The basis, T_1 is shown in Figure 5.3(a). T_1 consists of a single branch b_1 which connects the source o and the farthest sink t_1. For any free sink t_j, $\|(o, t_j)\| \leq \|(o, t_1)\|$. Obviously, b_1 is a leaf *bounding* branch for any t_j. By lemma 2, each t_j has a feasible balance point s_j on b_1.

(a) (b)

Figure 5.3: (a) Free sink t_j has a feasible balance point s_j on branch b_1 in partial tree T_1 which connects the source o with the farthest sink t_1. (b) t_k has no feasible balance point in partial tree T_{i+1}.

The inductive step, we intend to prove that every free sink has at least one feasible balance point on $T_i + 1$ $(1 \leq i < n-1)$, if it is true on $T_1, \ldots, T_{i-1}, T_i$. The proof is achieved by the contradiction. Assume that a free sink t_k can not find any feasible balance point on T_{i+1}. For t_k, there are a set of leaf bounding branches $\{\ \overline{t_{k_1}s_{k_1}}, \overline{t_{k_2}s_{k_2}}, \ldots, \overline{t_{k_m}s_{k_m}}\ \}$, as shown in Figure 5.3(b). A leaf branch $b_{k_j} = \overline{t_{k_j}s_{k_j}}(1 \leq j \leq m)$, where t_{k_j} is a sink. Since t_k has no feasible balance point on $b_{k_1}, b_{k_2}, \ldots, b_{k_m}$, based on lemma 2, we have:

$$\|(t_k, s_{k_j})\| > \|(t_{k_j}, s_{k_j})\| \tag{5.4}$$

Let t_{k_m} be the latest sink added to the tree, among $t_{k_1}, t_{k_2}, \ldots, t_{k_m}$. There exists a partial tree T_d $(1 \leq d \leq i)$ to which t_{k_m} is going to be connected next. Note that t_{k_1}, $t_{k_2}, \ldots, t_{k_{m-1}}$ have been connected to T_d. According to the inductive hypothesis, t_k has a feasible balance point s_k on some leaf branch in T_l. By (5.4), the only possible candidate for such a leaf branch is the branch which t_{k_m} is also going to connect to. Recall t_k has no feasible balance point on leaf branch $b_{k_m}(\overline{t_{k_m}s_{k_m}})$ at T_i, such that $\|t_k, s_k\| > \|t_{k_m}, s_{k_m}\|$. According to Max rule, t_k should be connected to the branch before t_{k_m}. This leads to contradiction.

Based on the above three Lemmas we have:

Theorem 1: *The Steiner tree produced by the algorithm is planar and has equal path length.*

Theorem 2: *The equal path length Steiner tree produced by the algorithm has minimal path length.*

Proof: The proof is trivial. The lower bound on path length is obviously the distance from the source to the furthest terminal. The first step of the algorithm is to connect the source to this furthest terminal. Since the remaining terminals are connected in such a way as to match the length of the first path, the minimal path length is achieved. □

6 Asymptotic Bound of Total Wire Length

The total wire length of the clock tree heavily depends on the distribution of sinks. The total wire length may be as worse as $O(n)$, where n is the number of sinks. As shown in Figure 6.1, the worst case happens when the clock sinks are distributed around the boundaries of a diamond region and the source is set at the center. Every sink is connected to the source, with the equal balance Manhattan distance r which is the half length of the diagonal of the diamond, such that the total wire length is $n * r$.

Figure 6.1: Worst case total wire length

However, in practice clock sinks are randomly distributed over a rectangle routing area. It is more interesting to know the asymptotic bound of the total wire length of the clock tree with a large number of sinks which are independent and uniformly, in probability, distributed on a rectangle region.

Let L be the total wire length of the clock tree with n sinks. Given a clock source at the center of layout, L is a function $L(t_1, t_2, \ldots, t_n)$ of the locations of the n sinks, $\{t_1, t_2, \ldots, t_n\} \subset R^2$, where R^2 represents a geometric plane. A function L is called a *nontrivial function*, if $L(\phi) = 0$ for the empty set ϕ.

Lemma 4: *L is a nontrivial function.*

Proof: This lemma is true since the total wire length is zero if there is no clock terminal to be connected.

A function L is called a *Euclidean function*, if it satisfies two properties: *(a) $L(\alpha t_1, \alpha t_2, \ldots, \alpha t_n) = \alpha L(t_1, t_2, \ldots, t_n)$, for all real $\alpha > 0$; (b) $L(t_1 + t_0, t_2 + t_0, \ldots, t_n + t_0) = L(t_1, t_2, \ldots, t_n)$, for all $t_0 \in R^2$.*

Lemma 5: *$L(t_1, t_2, \ldots, t_n)$ is a Euclidean function, for any finite set $\{t_1, t_2, \ldots, t_n\} \subset R^2$.*

Proof: $L(t_1, t_2, \ldots, t_n)$ equals to the sum of all branch lengths of the clock tree. For each branch $\overline{s_i s_j}$, the length is the Manhattan distance $\|s_i, s_j\|$. It is trivial to check that $\|s_i, s_j\|$ satisfies the above two properties. If all coordinates are scaled by a positive real α, L will also be scaled by α. Also, if all coordinates are shifted by the same distance specified by t_0 ($t_0 \in R^2$), L remains the same. So, L is a Euclidean function. $\square$

In the following, we define a special kind of functions called *subadditive function.* Let $\{Q_i : 1 \leq i \leq m^2\}$ be a partition of a unit region $[0,1]^2$ into smaller subregions with boundaries parallel to the coordinate axle, such that the boundary of a smaller region has the length of $1/m$. A function L is called $subadditivefunction$ if it satisfies the following *subadditivity property.* Given a positive integer m, there is a $C > 0$, such that

$$L(\{t_1, t_2, \ldots, t_n\} \cap [0,1]^2) \leq \sum_{i=1}^{m^2} L(\{t_1, t_2, \ldots, t_n\} \cap Q_i) + Cm \qquad (6.1)$$

In (6.1), the item $\{t_1, t_2, \ldots, t_n\} \cap [0,1]^2$ represents the subset of sinks of $\{t_1, t_2, \ldots, t_n\}$ which are located inside region $[0,1]^2$, and the item $\{t_1, t_2, \ldots, t_n\} \cap Q_i$ denotes the subset of sinks of $\{t_1, t_2, \ldots, t_n\}$ which are located in a subregion Q_i $(1 \leq i \leq m^2)$.

Lemma 6: *$L(t_1, t_2, \ldots, t_n)$ is a subadditive function, if $\{t_1, t_2, \ldots, t_n\}$ are independent and uniformly distributed in $[0,1]^2$.*

Figure 6.2: Clock tree Traversing procedure. Clock sinks are distributed in the region $[0,1]^2$, and o is the clock source located at the center of the region.

Proof: Suppose that sinks are independent and uniformly distributed in region $[0,1]^2$. Assume the clock source is located at the center of $[0,1]^2$ (See Figure 6.2(a)). We describe a hierarchical clock tree traversing procedure to sum up the total wire length of the clock tree. Note that the clock tree is actually obtained by the clock routing algorithm based on the max-min rules, assuming that numerous clock sinks are distributed independently and uniformly in region $[0,1]^2$. In the first level, $[0,1]^2$ is partitioned into four subregions with boundary length $1/2$, and the clock source is extended to centers $c_{1,1}, c_{1,2}, c_{1,3}$ and $c_{1,4}$ of the four subregions (see Figure 6.2(b)). The total wire length of these four branches from clock source to $c_{1,1}, c_{1,2}, c_{1,3}$ and $c_{1,4}$ is $4\frac{1}{2}$. Shown in Figure 6.2(c), the tree traversing continues by extending the center of the current region to centers of its subregions, and every subregion can be further partitioned into four smaller regions. Note that the tree is extended from $c_{1,1}$ to $c_{2,3}$ and $c_{2,4}$ with two branches, for example, because of the planar requirement. In the second level of tree traverse as shown in Figure 6.2(c), the total length of new branches, except for branches shown in Figure 6.2(b), is $4 \cdot 5 \cdot \frac{1}{2^2}$. In the third level of tree traverse as shown in Figure 6.2(d), the total length of new branches, except for branches shown in Figure 6.2(c), is less than $4^2 \cdot 6 \cdot \frac{1}{2^3}$. Note that six branches are extended from $c_{2,3}$ and $c_{2,4}$ to centers of four smaller subregions as shown in Figure 6.2(d). Suppose that $[0,1]^2$ is partitioned into m^2 regions. Each region Q_i $(1 \leq i \leq m^2)$ has the boundary length of $1/m$. Let s be the total levels of region partition in the above procedure. We have $s = \log m$. The total wire length of tree branches which are extended from the clock source to centers of m^2 small subregions is thus less than $4\frac{1}{2} + 4 \cdot 5 \cdot \frac{1}{2^2} + 4^2 \cdot 6 \cdot \frac{1}{2^3} + 4^3 \cdot 6 \cdot \frac{1}{2^4} + \cdots < \sum_{k=0}^{s-1} \frac{64^k}{2^{k+1}} = \sum_{k=0}^{s-1} 3 * 2^k$. Let L_i be the wire length of the clock tree inside region Q_i. Let $\sum_{i=1}^{m^2} L_i$ be the total wire length of tree branches inside those small subregions. We obtain the upper bound of L:

$$L \leq \sum_{i=1}^{m^2} L_i + \sum_{k=0}^{s-1} 3 * 2^k = \sum_{i=1}^{m^2} L_i + 3(m-1) \tag{6.2}$$

Based on (6.2), we have

$$L \leq \sum_{i=1}^{m^2} L_i + 3m \tag{6.3}$$

which satisfies the subadditivity property as stated in (6.1). □

The next Lemma 7 is presented in [17] which improved the theory in [19]. Lemma 7 is based on a weakening of subadditivity property [17]. For $m = 2$ or $m = 3$, there exists a constant C_1, such that

$$L(\{t_1, t_2, \ldots, t_n\} \cap [0,1]^2) \leq \sum_{i=1}^{m^2} L(\{t_1, t_2, \ldots, t_n\} \cap Q_i) + C_1 \tag{6.4}$$

Lemma 7: *Assume L is a nontrivial, subadditive, Euclidean functional on space R^d (d is the dimension). Set $a_2 = C_1/(2^{d-1} - 1)$. Set $a_1 = a + d2^{d-1}a_2$, where $a = L(t)$ for any variable t. Then for each non-empty finite subset F of $[0,1]^d$ with n elements, we have*

$$L(F) \leq a_1 n^{(d-1)/d} \tag{6.5}$$

In our case of (6.3), for $m = 2$, we have $C_1 = 6$. $a = L(t)$ is the length of clock tree which has only one terminal t in $[0,1]^2$. $a \leq 1$ since we connect this clock terminal directly with the source. The clock routing is performed on a plane R^2, that is $d = 2$. So, (6.5) can be rewritten as

$$L(F) \leq a_1\sqrt{n} \tag{6.6}$$

where $a_1 = a + 4a_2 = a + 4C_1 = a + 24$ $(0 \leq a \leq 1)$, is a positive constant.

Based on the above lemmas $4 - 7$ and (6.6), we have

Theorem 3: *The asymptotic bound is $O(\sqrt{n})$ for the total wire length of the clock tree with n sinks which are independent and uniformly distributed, in probability, in* $[0,1]^2$.

7 Sizing Widths of Clock Tree

The final objective of routing the clock net is to achieve zero skew and minimal phase delay of the clock waveforms at clock terminals. The planar clock tree with the equal and minimal path length is a good initial clock topology. This topology can be further improved with respect to clock skew by optimizing the branch widths. To achieve delay balance, instead of making the faster path slower by enlonging branches as in [22], we make slower paths faster by sizing. During the sizing, the clock topology is well maintained.

In this paper, we assume that the clock tree behaves as a RC tree with a source at the root. The planar equal path length clock tree is implemented on the same metal layer, such that all the RC parameters are the same and vias are eliminated. The source is driven by a set of transistors. Each clock branch is a RC line. For branch b_i, we have: $r_i = r_s l_i / w_i$, $c_i = c_s l_i w_i$, where l_i is the length and w_i is the width of the branch b_i. r_s is the sheet resistance with the unit $\Omega/\Box$, and c_s is the capacitance of the unit area with the unit $pf/\mu m^2$. We use the π circuit to model a RC line (see Figure 7.1(b)). Each clock pin t_k functions as a load capacitance c_k(see 7.1(c)). Note that clock pins are allowed to have different load capacitances.

Figure 7.1: (a) a RC line. (b) π-equivalent circuit of a RC line. (c) load capacitance of a terminal.

We size the branches of the clock tree in a bottom up order of starting from clock terminals. The similar algorithm is proposed in [22]. However, instead of the branch lengths [22], we take the branch widths as variables. First, we state the sizing algorithm for a binary clock tree. At the end of this section, we will show that the algorithm can be applied on a general clock tree with arbitrary node degree.

One recursive step is performed on two sibling branches with the same parent node in the clock tree. As shown in Figure 7.2(a), n_0 is the parent node with two children n_1 and n_2. Node n_1 is the root of subtree $ST(n_1)$, and n_2 is the root of subtree $ST(n_2)$. $\overline{n_0 n_1}$ and $\overline{n_0 n_2}$ are two sibling branches. The equivalent RC circuit of these two sibling branches is shown in Figure 7.2(b). r_1 and c_1 are the resistance and capacitance of the branch $\overline{n_0 n_1}$, and r_2 and c_2 the resistance and capacitance of branch $\overline{n_0 n_2}$. C_1 is the sum of total capacitance of all branches and terminals in $ST(n_1)$. D_1 is the time delay from n_1 to leaf nodes in $ST(n_1)$. If n_1 is a leaf node (terminal), D_1 is zero and C_1 equals to the load capacitance of the terminal. C_2 and D_2 are defined similarly for $ST(n_2)$.

Figure 7.2: (a) $\overline{n_0 n_1}$ and $\overline{n_0 n_2}$ are two sibling branches of the clock tree. Node n_0 is the parent node of n_1 and n_2. $ST(n_1)$ denotes the subtree rooted at node n_1, and $ST(n_2)$ denotes the subtree rooted at node n_2. (b) The π-RC circuit of two sibling branches $\overline{n_0 n_1}$ and $\overline{n_0 n_2}$.

Assume that $ST(n_1)$ and $ST(n_2)$ have achieved equal path delays by sizing the widths. Based on Elmore delay, to achieve the equal path delay from n_0 to leaf nodes in both subtrees in Figure 7.2(b), we have:

$$r_1(\frac{c_1}{2} + C_1) + D_1 = r_2(\frac{c_2}{2} + C_2) + D_2 \tag{7.1}$$

We specify that $r_1 = r_s l_1 / w_1$, $c_1 = c_s l_1 w_1$, and $r_2 = r_s l_2 / w_2$, $c_2 = c_s l_2 w_2$, where l_1, w_1 are the length and width of branch $\overline{n_0 n_1}$, and l_2, w_2 are the length and width of branch $\overline{n_0 n_2}$. Note that w_1 and w_2 are variables. Substituting the above r, c values into (7.1), we get:

$$r_s l_1 C_1 \frac{1}{w_1} + \frac{r_s c_s}{2} {l_1}^2 + D_1 = r_s l_2 C_2 \frac{1}{w_2} + \frac{r_s c_s}{2} {l_2}^2 + D_2 \tag{7.2}$$

Let $x_1 = \frac{1}{w_1}$, $x_2 = \frac{1}{w_2}$, and $\theta = \frac{r_s c_s}{2}({l_1}^2 - {l_2}^2) + (D_1 - D_2)$. (7.2) is rewritten as:

$$l_2 C_2 x_2 = l_1 C_1 x_1 + \frac{\theta}{r_s} \tag{7.3}$$

We thus obtain:

$$x_2 = \frac{1}{l_2C_2}(l_1C_1x_1 + \frac{\theta}{r_s}) \tag{7.4}$$

or

$$x_1 = \frac{1}{l_1C_1}(l_2C_2x_2 - \frac{\theta}{r_s}) \tag{7.5}$$

We first specify w_1 as the normal width. then obtain w_2 based on (7.4), where x_1, x_2 are inverses of w_1, w_2. If w_2 is less than the normal width, we have to assign w_2 as the normal width, and get w_1 based on (7.5). Let C_0 denote the sum of total capacitances in the subtree rooted at n_0, and D_0 denotes the path delay from n_0 to leaf nodes in the subtree. Note that the subtree rooted at n_0 has zero skew after we adjust the widths of $\overline{n_0n_1}$ and $\overline{n_0n_2}$. We have:

$$C_0 = C_1 + C_2 + c_s(l_1w_1 + l_2w_2) \tag{7.6}$$

$$D_0 = D_1 + \frac{r_sl_1}{w_1}(\frac{c_sl_1w_1}{2} + C_1) \tag{7.7}$$

We apply the above sizing step on every pair of sibling branches, by starting from leaf nodes (terminals) and then recursively going through the clock tree. The algorithm of recursive clock tree sizing is described in the following pseudo-code.

Recursive clock tree sizing algorithm based on Elmore delay

```
Input: a clock tree $T$ with a source $s$ and a set of terminals (sinks);
Output: a clock tree with variable branch widths.

Procedure **ClockTreeSizing**($T$, technology file){
    $n_0$ = the child of source $s$, $b_0$ = the branch connecting $n_0$ and $s$;
    $D_0$ = the delay from $n_0$ to leaf nodes, $C_0$ = total capacitances of the subtree rooted at $n_0$;
    CalculateNodeDelay($n_0, C_0, D_0$);
    Assign $b_0$ a large width specified by users.
}

Procedure **CalculateNodeDelay**($n_0, C_0, D_0$) {
    if ($n_0$ is a terminal $t_k$)
        $D_0 = 0$, $C_0 = c_k$;
    else {
        $n_1, n_2$ = two children of $n_0$;
        CalculateNodeDelay($n_1, C_1, D_1$);
        CalculateNodeDelay($n_2, C_2, D_2$);
        $\theta = \frac{r_sc_s}{2}({l_1}^2 - {l_2}^2) + (D_1 - D_2)$;
        $w_1$ = normal width, $x_1 = \frac{1}{w_1}$;
        $x_2 = \frac{1}{l_2C_2}(l_1C_1x_1 + \frac{\theta}{r_s})$, $w_2 = \frac{1}{x_2}$;
        if ($w_2 <$ normal width) {
            $w_2 =$ normal width, $x_2 = \frac{1}{w_2}$;
            $x_1 = \frac{1}{l_1C_1}(l_2C_2x_2 - \frac{\theta}{r_s})$, $w_1 = \frac{1}{x_1}$;
            }
        $D_0 = D_1 + \frac{r_sl_1}{w_1}(\frac{c_sl_1w_1}{2} + C_1)$, $C_0 = C_1 + C_2 + c_s(l_1w_1 + l_2w_2)$;
    }
}
```

Source s in the algorithm is assumed to have only one child n_0. b_0 is the branch which is connected to s. The width of b_0 has no effect on the skew based on Elmore delay. But we still prefer to assign b_0 a large width to minimize the path delay. The width of b_0 is usually larger than other branches and is specified by users in the real design. If s has two children, we can take $n_0 = s$ and $b_0 = null$.

This sizing method can also be applied on the general clock tree with nodes which may have more than two children. As in Figure 7.2(a), we assume n_0 has a set of children nodes $\{n_1, n_2, \ldots, n_m\}(m \geq 2)$. r_i and c_i are the resistance and capacitance of the branch $\overline{n_0n_i}$ $(1 \leq i \leq m)$. Let $ST(n_i)$ be the subtree rooted at n_i. C_i is the sum of total capacitance of all branches and terminals in $ST(n_i)$. D_i is the time delay from n_i to leaf nodes in $ST(n_i)$. Assume that $ST(n_1)$, $ST(n_2)$, ..., $ST(n_m)$ have achieved equal path delays by sizing the widths. Similar to equation (7.1), to achieve the equal path delay from n_0 to leaf nodes in all subtrees, we have:

$$\begin{aligned} r_1(\frac{c_1}{2} + C_1) + D_1 &= r_2(\frac{c_2}{2} + C_2) + D_2 \\ r_1(\frac{c_1}{2} + C_1) + D_1 &= r_3(\frac{c_3}{2} + C_3) + D_3 \\ &\vdots \\ r_1(\frac{c_1}{2} + C_1) + D_1 &= r_m(\frac{c_m}{2} + C_m) + D_m \end{aligned} \tag{7.8}$$

where $r_i = r_sl_i/w_i$, and $c_i = c_sl_iw_i$. l_i, w_i are the length and width of branch $\overline{n_0n_i}$ $(1 \leq i \leq m)$. Let $x_1 = \frac{1}{w_1}$, $x_2 = \frac{1}{w_2}$, ..., and $x_m = \frac{1}{w_m}$. We define that $\theta_2 = \frac{r_sc_s}{2}({l_1}^2 - {l_2}^2) + (D_1 - D_2), \theta_3 = \frac{r_sc_s}{2}({l_1}^2 - {l_3}^2) + (D_1 - D_3), \ldots,$ and $\theta_m = \frac{r_sc_s}{2}({l_1}^2 - {l_m}^2) + (D_1 - D_m)$. Similar to (7.3), we can rewrite (7.8) as

$$\begin{aligned} l_2C_2x_2 &= l_1C_1x_1 + \frac{\theta_2}{r_s} \\ l_3C_3x_3 &= l_1C_1x_1 + \frac{\theta_3}{r_s} \\ &\vdots \\ l_mC_mx_m &= l_1C_1x_1 + \frac{\theta_m}{r_s} \end{aligned} \tag{7.9}$$

If we specify the width of one of branches $\{\overline{n_0n_1}, \overline{n_0n_2}, \ldots, \overline{n_0n_m}\}$, based on (7.9), we can obtain the widths of other branches, achieving the zero skew from n_0 to leaf nodes in its subtree.

8 Experimental Results

The planar clock routing algorithm has been implemented in ANSI C with a MOTIF/X-window user interface. Several examples, including industrial benchmarks, have been tested and the results are promising.

Table 1 highlights the comparison on two MCNC benchmarks among MMM [13], RGM [14] and our algorithm. The criterions compared are: planarity, path length skew, longest path length, total wire length, and running time. The MMM algorithm has not achieved an equal path length clock tree because of the non-zero skew of path lengths. Compared with the RGM algorithm, our algorithm decreases the path length by 24.5% on Primary1 and 19.6% on Primary2. The path length is a first order indication of phase-delay and attenuation from the clock source to terminals. Using our algorithm, the total length of the clock tree increases by 19.0% on Primary1 and 19.8% on Primary2. This extra length is a result of maintaining the planarity of the tree. The running time of our algorithm on Primary2, which has more than 600 terminals, is only 1/3 the time required by the RGM algorithm. This speed-up is due to the divide-and-conquer nature of our algorithm. Figure 8.1 shows two planar clock trees generated from Primary1. Figures 8.1(a) and 8.1(b) show, respectively, the clock trees resulting from choosing a source on the edge of the instance and at the center of the instance. Our algorithm also constructs a planar clock tree with equal path length for Primary2 (see Figure 8.2), while the RGM algorithm produces a non-planar tree with many overlaps (see Figure 8.3). The other previous algorithms obtain similar non-planar trees [13, 22, 6]. The nonplanar clock tree must employ two or more routing layers to finish physical embedding, such that worsening the clock skew and source-sink path delay because of electrical parameter variations of different layers and vias.

	Primary1			Primary2		
	MMM	RGM	Our Algorithm	MMM	RGM	Our Algorithm
Planarity	No	No	Yes	No	No	Yes
Path Length Skew	0.29	0.00	0.00	0.74	0.00	0.00
Longest Path Length	7.24	7.51	6.03	13.05	11.58	9.96
Total length	161.7	153.9	190.1	406.3	376.7	469.9
Time (sec)	2.6	54.9	38.5	20.2	397.1	144.2

Table 8.1: Statistics of different clock routing algorithms on two benchmarks. Primary1 has 269 clock terminals and Primary2 has 603 clock terminals. CPU time is measured on SUN (Sparc 1+) Workstation.

As shown in Figure 8.4(a), we apply the sizing algorithm in Section 7 on a clock tree of five terminals with equal path lengths from the source s. Initially, the clock tree has a uniform width 2 μm. By using the above sizing algorithm, branch widths of the clock tree are shown in 8.4(b). We assume the clock tree is implemented on metal two with 1 μm feature size and take $r_s = 27\ m\Omega/\Box$ and $c_s = 0.017 \times 10^{-3}\ pf$. Every branch b_i is assigned a 2-tuple $\{l_i, w_i\}$, where l_i is the length of b_i and w_i the width. Both are in the unit of μm. Starting from five leaf nodes (terminals) which have the same load capacitance 0.8 pf. we obtain the branch widths by applying the recursive sizing algorithm (see Figure 8.4(b)). Note that we assign a large width 20 μm on the trunk which is directly connected from the source s to reduce the path delay. We used SPICE to simulate the equivalent $\pi - RC$

(a)

(b)

Figure 8.1: Planar clock tree with equal path length on Primary1 benchmark. (a) clock source is inside the chip/module. (b) clock source is on the frame of the chip/module.

Figure 8.2: Planar clock tree with equal path length on Primary2 benchmark.

Figure 8.3: Clock tree of RGM algorithm on Primary2 benchmark. (figure copy of [2])

circuit of the planar equal path length clock tree shown in Figure 8.4. Simulation results are shown in Figure 8.5, where the horizontal axis indicates the time with the unit of 10^{-8} s and the vertical axis shows the voltages at clock terminals. Shown in Figure 8.5(a), the waveforms have a skew of 0.21 ns, where we assign identical width 2μm on the clock tree. After sizing the clock tree with varible widths shown in Figure 12(b), we reduce the clock skew to 0.09 ns. The final clock waveforms are shown in Figure 8.5(b). As the result, the maximum path delay from the source to terminals is also decreased from 0.34 ns to 0.14 ns.

Figure 8.4: (a) Planar clock tree with equal path length with five terminals. Each terminal has the same load capacitance 0.8 pf. (b) The sizing result of the clock tree. Each edge b_i has a 2-tuple $\{l_i, w_i\}$, where l_i is the length of b_i and w_i is the width assigned to b_i.

(a)

(b)

Figure 8.5: Waveforms at five terminals in a planar clock tree with equal path-length as shown in Figure 15. (a) We assign the identical width 2 μm on the clock tree. (b) We size the clock tree with variable widths as shown in Figure 15(b).

9 Concluding Remarks

This paper presents a new algorithm to construct a planar clock tree based on hierarchical max-min optimization. We have proved that the algorithm guarantees a planar equal path length clock tree rooted directly at the source, such that the path length from the source to terminals is minimized. A planar clock tree may be implemented on a single metal layer. Since it is easier to achieve uniform electrical parameters on a single layer than when switching layers, it is easier to adjust a planar clock tree for zero skew and minimal path delay. Planar clock routing becomes feasible when more layers are available.

The planar equal path length Steiner tree problem is a valuable problem in the field of computational geometry. In addition to routing clock nets, some other nets with special synchronization requirements, may benefit from such a tree.

The algorithm can also be applied to construct an equal path length Steiner tree on

Euclidean space. We only need to use Euclidean distance instead of Manhattan distance in the algorithm to measure the balance distance of a sink. Theorem 1 and Theorem 2 still hold in the case of using the algorithm on Euclidean space. Lemma 1 is true according to Max rule no matter of Euclidean distance or Manhattan distance. Lemma 2 follows that constraint (2) is well known for three side lengths of a triangle on Euclidean space. Lemma 3 is proved only based on Max-Min rules and Lemma 1, Lemma 2, no matter what Euclidean distance or Manhattan distance is used to measure the balance distance. So, Theorem 1 and Theorem 2 are naturally achieved because three preceding lemmas are correct on Euclidean space.

Equal path-length is the first order indication of balanced path delay from the source to every clock terminal. We further obtain zero skew on Elmore delay by sizing the planar equal path length clock tree while maintaining the planarity of the clock tree. Some preliminary results are promising which achieve near zero skew by using SPICE simulation.

10 Acknowledgement

This work was supported by the National Science Foundation Presidential Young Investigator Award under Grant MIP-9009945. The authors would like to thank Jason Cong and Andrew Kahng of UCLA for providing the source codes of their implementations [14, 13] and benchmarks. Andrew Kahng provided the references and comments on subadditive functionals. WanSoo Rhee provided her recent paper [17]. Joe Xi did the SPICE simulation and discussed with authors several times during the course of this research. David Staepelaere provided helpful comments on a early draft of this paper.

References

[1] H. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley Publishing Company, 1987.

[2] H. Bakoglu, J. T. Walker, and J. D. Meindl. A symmetric clock-distribution tree and optimized high-speed interconnections for reduced clock skew in ulsi and wsi circuits. In *Proc. IEEE Intl. Conf. on Computer Design*, pages 118–122, 1986.

[3] S. Boon, S. Butler, R. Byrne, B. Setering, M. Casalanda, and A. Scherf. High performance clock distribution for cmos asics. In *IEEE Custom Integrated Circuits Conference*, pages 15.4.1–15.4.5, 1989.

[4] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. A survey of optimization techniques for integrated circuits. *Proc. of the IEEE*, 69(10):1334–1362, 1991.

[5] D. L. Carter and D. F. Guise. Effects of interconnections on submicron chip performance. *VLSI Design*, pages 63–68, Jan. 1984.

[6] T.H. Chao, Y.C. Hsu, and J.M.Ho. Zero skew clock net routing. In *Proc. of 29th Design Automation Conf.*, pages 518–523, 1991.

[7] D.F.Wann and M.A. Franklin. Asynchronous and clocked control structures of vlsi-based interconnection networks. *IEEE Trans. Computers*, C-32(3):284–293, March 1983.

[8] D. Dobberpuhl and R. Witek. A 200mhz 64b dual-issue cmos microprocessor. In *Proc. IEEE Intl. Solid-State Circuits Conf.*, pages 106–107, 1992.

[9] M. Edahiro. A clock net reassignment algorithm using voronoi diagrams. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 420–423, 1990.

[10] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, 1990.

[11] E. G. Friedman and S. Powell. Design and analysis of a hierarchical clock distribution system for synchronous standard cell/macrocell vlsi. *IEEE Journal of Solid-State Circuits*, sc-21(2):240–246, 1984.

[12] J. Ho, D. T. Lee, C. H. Chang, and C. K. Wong. Bounded-diameter spanning tree and related problems. In *Proc. ACM Symp. on Computational Geometry*, pages 276–282, 1989.

[13] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh. Clock routing for high-performance ics. In *Proc. of 27th Design Automation Conf.*, pages 573–579, 1990.

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

[15] D. C. Keezer and V. K. Jain. Clock distribution strategies for wsi: A critical survey. In *Intl. Conf. on Wafer Scale Integration*, pages 277–283, 1991.

[16] P. Ramanathan and K. G. Shin. A clock distribution scheme for non-symmetric vlsi circuits. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 398–401, 1989.

[17] WanSoo T. Rhee. A matching problem and subadditive euclidean functionals. *to be appeared in The Annals of Probability*, 1993.

[18] J. Rubinstein, P. Penfield, and M. A. Horowitz. Signal delay in rc tree networks. *IEEE Trans. on Computer-Aided Design*, CAD-2(3):202–211, 1983.

[19] J. M. Steele. Subadditive euclidean functionals and nonlinear growth in geometric probability. *The Annals of Probability*, 9(3):365–376, 1981.

[20] J. M. Steele. Growth rates of euclidean minimal spanning trees with power weighted edges. *The Annals of Probability*, 16(4):1767–1787, 1988.

[21] J. M. Steele and T. L. Snyder. Worst-case growth rates of some classical problems of combinatorial optimization. *SIAM Journal on Computing*, 18(2):278–287, 1989.

[22] R.-S. Tsay. Exact zero skew. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 336–339, 1991.

[23] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

[24] Qing Zhu and Wayne W.M. Dai. Perfect-balance planar clock routing with minimal path-length. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 473–476, 1992.