University of California
Santa Cruz

Chip and Package Co-Design of Clock Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

by

Qing Zhu

June 1995

The dissertation of Qing Zhu is approved:

Wayne Wei-Ming Dai

David Helmbold

Martine Schlag

Dean of Graduate Studies and Research

Contents

List of Figures

List of Tables

Chip and Package Co-Design of Clock Networks

Qing Zhu

ABSTRACT

This dissertation presents the new concept of chip/package co-design for clock networks. Constructing a large flat size clock network that achieves zero skew is very challenging in VLSI design. Two new concepts are presented in this dissertation: (1) Since the package layer has far smaller RC interconnect parameters than a chip layer, placing the global clock tree on the package layer can decrease the skew, delay and wire capacitance significantly. (2) Using tolerable skew constraints between sequentially adjacent registers in synchronous circuits can simplify the clock distribution and reduce the total wire length of the clock tree significantly.

We proposed a new clock distribution scheme which partitions the clock network into two levels. First, the clock terminals are partitioned into a set of clusters. For each cluster, a local on-chip clock tree is used to distribute the clock signal from a locally inserted buffer to terminals inside this cluster. The clock signal is then distributed from the main clock driver to each of local buffers by means of a global clock tree, which is routed on the package layer. Flip chip area I/O connections are used to make the connections between the global clock tree and the on-chip local clock trees.

Algorithms are proposed to construct a planar equal path length tree for the global clock tree. Since it is planar, the global clock tree can be embedded on a single package layer, reducing the delay and attenuation through vias as well as sensitivity to process variations. The global clock tree is further improved iteratively to reduce the total wire length while maintaining the tolerable skew constraints. The skew constrained cut-and-link tree plays a key role in the iterative refinement.

A clock sizing technique is used to decrease the skew of the global clock tree if the tolerable skew constraints are not satisfied. The optimal clock sizing problem is solved by a least square minimization method.

In addition, this dissertation investigates the delay bounded minimum Steiner tree problem. Delay bounded minimum Steiner tree can be used for the interconnect tree construction when the interconnect delay becomes significant. A new iterative algorithm is proposed to construct a delay bounded minimum Steiner tree.

Keywords: clock network, clock distribution, clock routing, net clustering, interconnect tree, package and chip co-design, flip chip, area I/O, multichip module

Acknowledgement

I extend the special appreciation to Professor Wayne Wei-Ming Dai for his advice and strong support throughout my Ph.D. study at UC Santa Cruz. People in the SURF group, especially David Staepelaere, provided a lot of help on the integration of the clock routing program. I would like to thank David Staepelaere for his careful reading on this dissertation. I would like to thank Joe G. Xi for providing suggestions and discussions throughout the entire research work. I enjoy the co-opeartion with Mehrdad Parsa on the delay bounded tree algorithm. I appreciate Professor David Helmbold and Professor Martine Schlag for serving in my Ph.D. defense committee and spending time reading the dissertation.

This dissertation is finished by the love and support from my wife Huiling Song and my parents in China.

CHAPTER 1. Introduction

Clocking is a major topic in high-speed digital system design. The clock frequency determines the CPU data processing rate[1], and the bus data transmission rate for I/O circuits and memories.[2] Digital system designers usually strive to maximize the clock frequency in order to achieve high system performance. Figure 1.1(a) shows the expected clock frequencies used in CPUs in the near future. Besides using a fast circuit family for logic and storage elements, a good clock distribution network is critical to attain a high clock frequency, which is the major research topic of this dissertation.

1.1 Clock Distribution

Clock distribution is one of the limiting factors for digital circuits operating above 80MHZ. Device technology improvement, such as deep-submicron with faster transistors, can only marginally solve the clock distribution problem because interconnect delay is gradually becoming the dominant factor in clock cycle time. Figure 1.1 and Figure 1.2 show the IC technology progress in the near future. Technological achievements such as smaller

---

[1] The CPU data processing rate is the ratio of the clock frequency to the average number of cycles required to execute an instruction.

[2] The data transmission rate of I/O and memory buses is the product of the clock frequency and the bus width.

delay and low power based on a new concept called *chip and package co-design* of clock networks. This methodology and related algorithms have been implemented in ANSI C programs.

1.2 Flip Chip and Area I/O Technology

In deep submicron (feature sizes of $0.5\mu m$ or less) processes, long interconnect delays dominate gate delays and limit the chip performance. With increasing I/O counts, chips become too small to accommodate all the I/Os using peripheral pads. Further decreasing of the bond pad pitch (distance between centers of two adjacent pads) causes reliability problems with the wire bonding technology. More and more chips will become pad limited. For low power systems with supply voltages of 3V or less, logic levels become sensitive to simultaneous switching noise due to large wire bond inductances. Even with increasing on-chip circuit speeds due to reduced process feature sizes, the off-chip delays from the chip I/Os to the package leads are becoming serious which limits the performance of an entire system.

In flip chip technology, the die is attached to the package via solder bumpers with pads arranged over the chip surface. The area I/O bond pads on the bare chip are solder bumped. The chips are placed face down directly on the package substrate and then the solder bumps are reflowed at high temperature. Figure 1.5 shows the flip chip assembly in a ball grid array (BGA) package. Flip chip technology [Bak87, Fry93] is becoming popular for high-performance and high-density VLSIs. It provides a number of advantages including:

1. Very short lead length, which reduces the lead inductance and hence the noise level.
2. Large numbers of area pads in addition to peripheral pads. The area I/O connections of flip chip, rather than peripheral I/O connections in wire bonding and TAB technologies, provide the ability to use package level layers for global connections that

lower chip-to-chip delays. In high speed multichip modules, several flip dice are mounted on a flip chip attachment provides the smaller delay from chip to package and the MCM package provides the smaller delay from chip to chip. Substrates operating at 400MHZ have been reported [GFK$^+$93]. These substrates use redesigned I/O buffers to reduce power consumption by $6.0X$, and increase performance by $2.5X$ when compared to conventional CMOS buffers.

Flip chip based VLSI systems and MCMs usually work at very high clock frequencies and have to meet the challenging goals of small skew and small delay. However, when MCM technology is used for lap-top computers and communication network circuits. Another goal is low power dissipation. MCM and flip chip technologies improve the performance of the clock distribution by significantly decreasing the lead inductance of off-chip I/O bondings and decreasing the chip-to-chip interconnect lengths. However, the most significant advantage provided by the area I/Os in the flip chips is that the on-chip global clock tree can be re-assigned to the package layer which has far smaller interconnect RC parasitics. The clock skew, clock delay and clock wire capacitance of the global clock tree will be dramatically reduced.

1.3 Advantages of Assigning Global Clock Tree to Package Layer

Because of the increased number of I/O connections avaliable with flip chip technology, it is feasible to route the global clock tree down to the package layers. On-chip interconnect resistance increases as feature sizes are scaled down, especially as we move to deep submicron. The package layers provide much wider and thicker interconnects than on chip with usually $10-100$ times larger interconnect scale. The RC parameters of interconnects can be far smaller on the package layer. This can be seen from the interconnect scaling properties. For multilayer embedded microstripline, the unit length resistance and unit length

	Wire width	Wire thickness	Resistance	Capacitance
On-chip M3 and M2 layer	1.2 μm	0.6 μm	22 $m\Omega/\mu m$	0.124 $fF/\mu m$
On-package layer	50 μm	50 μm	0.0134 $m\Omega/\mu m$	0.0127 $fF/\mu m$
Solder bump	-	-	2 $m\Omega$	1 fF

Table 1.1: RC Parameters of Flip Chip and Plastic Package Technology

capacitance are:

$$R \approx \rho/wt_m, \qquad C \approx \epsilon w/t_i \tag{1.1}$$

where w is the line width, t_m the metal thickness, t_i the dielectric thickness, ρ the metal resistivity, and ϵ the dielectric permittivity. Therefore, if we uniformly scale the interconnect cross-section of a line as well as the dielectric thickness, the per unit length capacitance will remain the same. However, the per unit length resistance is inversely proportional to the area of the cross-section.

Table 1.1 shows a comparison of the unit length RC parameters for plastic package layers and 0.6μm CMOS chip layers. This data is provided by LSI-Logic Corporation[Zhu94]. For the same length wire, the wire resistance on a package layer is 1690 times (three orders) smaller than the wire resistance on the M3 and M2 layers of a chip. Meanwhile, the wire capacitance on a package layer is 10 times smaller than the wire capacitance on the M3 and M2 layers of a chip. Note that the RC parasitics of a solder bump are negligible when compared with the wire RC parasitics.

A case study has been done to evaluate the performance improvement when the global clock tree is assigned to the package layer [Zhu94]. We use the clock network of a chip from LSI Logic corporation with 13440 clock loads. The RC interconnect parameters on chip and on package are shown in Table 1.1. We redesign the clock tree by putting the first level clock tree on a dedicated package layer. HSPICE simulation results are shown in Figure 1.6(c),

- An additional benefit results from moving the global clock distribution to the package level: it reduces the density of on chip interconnects by separating the global clock tree. This should allow routers to work at their best for remaining nets on the chip.

1.4 Dissertation Organization

This dissertation is divided into seven chapters.

Chapter 2 describes the chip and package co-design scheme for clock networks. Tolerable skew constraints between registers are used to guide the clock tree synthesis. A skew constrained net clustering method is presented. Industrial chips and benchmarks are tested.

Chapter 3 presents two algorithms for constructing a planar equal path clock tree. The correctness and time complexity are discussed.

Chapter 4 describes the skew constrained clock routing approach. An skew constrained iterative cut-and-link tree technique is presented.

Chapter 5 describes the clock network sizing technique used to decrease the skew of the clock tree in order to meet the tolerable skew constraints. The clock sizing optimization problem is solved by Gauss-Marquardt's least square minimization method.

Chapter 6 describes research on the delay bounded interconnect tree construction. An iterative approach is proposed to construct a delay bounded minimum Steiner tree.

Chapter 7 provides future directions to extend the current research and the summary of this dissertation.

CHAPTER 2. Clock Synthesis Scheme and Skew Constrained Net Clustering

2.1 Overall Scheme

A two-level clock distribution scheme is shown in Figure 2.1. The clock netlist is partitioned into a set of clusters of clock terminals, and local buffers are inserted at every cluster. The first level tree or global clock tree connects the clock driver (source) to local buffers, and the second level or local clock trees connect clock terminals within every cluster. The global clock tree is routed on the package level and solder bumps are used to connect to on-chip local buffers. A physical implementation of the two-level clock tree on the chip and package levels is shown in Figure 2.1(b). The local clock trees and the active clock driver and local buffers are implemented on chip.

The two-level clock tree is synthesized using the following principle stages. Figure 2.2 outlines these stages which are applicable to single chip packages and multi-chip modules.

- Net clustering: partitioning the clock net into a set of clusters of neighboring clock terminals.
- Buffer assignment and local clock routing: inserting the local buffer and constructing the local clock tree for every cluster.
- Global clock routing: constructing the global clock tree from the clock driver to the

[Fis90], we have the *negative skew constraint* when $t_j \leq t_i$:

$$t_i - t_j \leq P - d_{max}(i,j) - SETUP \tag{2.3}$$

Tolerable skew exists in a synchronous circuit:

1. Registers (flip flops) that are sequentially adjacent have tolerable skew constraint expressed by (2.2) and (2.3).
2. Registers (flip flops) that are not sequentially adjacent have no direct skew constraints.

2.3 Intra-Cluster and Inter-Cluster Skew Constraints

The intra-cluster skew constraints are imposed by the sequentially adjacent registers located in the same cluster which are expressed in (2.2) and (2.3). The delays of the local clock tree needs to satisfy these intra-cluster skew constraints. The sequentially adjacent registers located in different clusters impose the inter-cluster skew constraints that the delays of the global clock tree need to satisfy.

Given a pair of sequentially adjacent registers $R_i \rightarrow R_j$ located in two clusters C_1 and C_2, R_i and R_j satisfy the positive skew constraint and negative skew constraint expressed in (2.2) and (2.3). Let t_i be the path delay from the global clock driver (clock source) to R_i, which is composed of three items: the delay t_1^g in the global clock tree from the driver to the local buffer of cluster C_1, the intrinsic buffer delay t_1^b, and the delay t_i^l in the local tree from the local buffer to R_i, as illustrated in Figure 2.6.

$$t_i = t_1^g + t_1^b + t_i^l \tag{2.4}$$

Similarly, the delay from the global driver to R_j is

$$(t^g{}_1+t^b{}_1)-(t^g{}_2+t^b_2) \leq MIN(t^l_j-t^l_i+P-d_{max}(i,j)-SETUP) \quad (R_i \in C_1, R_j \in C_2) \tag{2.7}$$

Based on (2.6) and (2.7), the inter-cluster skew constraints are usually tighter than intra-skew constraints, because the inter-cluster skew constraints are derived to satisfy the smallest skew constraint of sequential registers in two clusters as well as compensating the delay differences of local trees. So, in the net clustering procedure, we combine sequential registers with tighter skew constraints in the same cluster in order to give more tolerable skew budget to inter-clusters.

2.4 Skew Constrained Net Clustering Method

Net clustering procedure partitions the clock terminals into a set of *isochronous clusters.*

Isochronous Cluster: within this cluster the intra-cluster skew constraints are satisfied by the local clock tree.

Two objectives are achieved in the net clustering.

(a) Neighbor registers with tighter skew constraints and closer locations are combined in the same cluster in order to give more tolerable skew budget between clusters. The inter-cluster skew constraints are usually tighter than intra-skew constraints.

(b) The number of clusters is minimized. Minimizing the cluster number is to reduce the area pads needed for the clock distribution. In addition, less clusters needed to be connected by the global clock tree result in less total wire length.

Given a set of clock terminals distributed in a plane, we construct a Delaunay triangulation graph G(V, E) to represent the neighboring in locations of clock terminals. Each vertex in V represents a clocked terminal. Each edge in E connects two neighbor clock terminals.

For every pair of neighbor clock terminals R_1 and R_2 with an edge e connected in the Delaunay triangulation graph G(V, E), we define $D(e)$ as the distance between R_1 and R_2. If R_1 and R_2 are also sequentially adjacent registers in the logic circuit (signal path is from R_1 to R_2 as shown in Figure 2.5), we define s_p as the positive skew constraint from R_1 to R_2, and s_n as the negative skew constraint.

Weight $S(e)$ is associated with each edge $e(R_1, R_2) \in E$ which is defined as follows:

$$S(e) = \begin{cases} D(e) - \frac{\alpha}{MIN(s_p, s_n)} & e \text{ intervenes two sequentially adjacent registers } R_1 \rightarrow R_2. \\ & \alpha > 0 \text{ is an adjustable parameter.} \\ D(e) & e \text{ intervenes two non-sequentially adjacent registers.} \end{cases} \tag{2.8}$$

Weight $S(e)$ is the combination of the closeness and the skew constraint tightness between two neighbor registers. The smaller edge weight in the Delaunay triangulation graph represents that two neighbor registers have tighter skew constraints or closer locations. A edge may have negative weight for two sequentially adjacent neighbor registers with tight skew constraints. The weighted Delaunay triangulation graph is shown in Figure 2.7.

With each subset V_i of V, we define $I(V_i)$ as the sum of $S(e)$ of all edges connecting both vertices belonging to V_i:

$$I(V_i) = \sum_{e(R_j \in V_i, R_k \in V_i)} S(e) \tag{2.9}$$

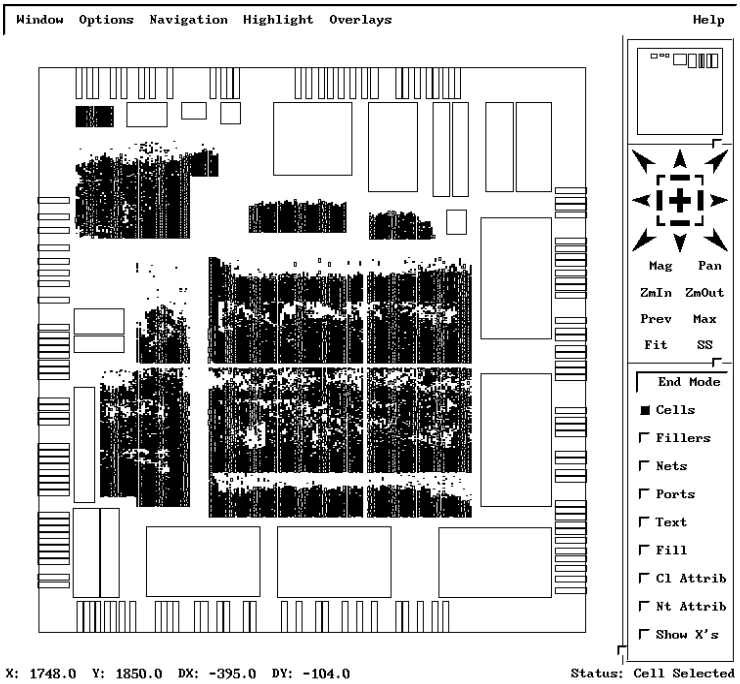
Skew Constrained Clock Net Clustering Problem: *Given a Delaunay triangulation* $G = (V, E)$ *of clock terminals (registers), with a weight function S(e) for every edge* $e \in E$ *defined in (2.8), find a family of* n *disjoint subsets of* V *with* $\bigcup_{i=1}^{n} V_i = V$*, such that* $\sum_{i=1}^{n} I(V_i)$ *is minimized for the smallest* n *admitting every* V_i *to be an isochronous cluster.*

2.5 Experimental Results

We have implemented the chip and package clock co-design methodology in ANSI C with a MOTIF/X-window user interface. The program has been integrated into UC Santa Cruz SURF system [SJDD93] for MCM and package routing. Clock net clustering is accomplished based on the Delaunay triangulation [Lu91].

We tested a large industrial mixed standard cell and macrocell chip from LSI Logic Corporation, which has 3879 registers as shown in Figure 2.8. It uses the $0.6\mu m$ CMOS flip chip technology and plastic package, and the electrical parameters are shown in Table 1.1. The local clock tree is routed on M2 and M3 layers with $1.2\mu m$ interconnect width, and the global clock tree is on package layer with $50\mu m$ interconnect width. The distribution of clocked registers and the Delaunay triangulation is shown in Figure 2.9(a). The clock net clustering is done as shown in Figure 2.9(b). Uniform size $0.6\mu m$ CMOS local buffers are added at the cluster center, and the skew of local buffer delay mismatch is reduced by using the buffers with the same size and the same process. The clock skew is caused mainly by the difference of interconnect delays in the two-level clock tree. Local clock trees within every cluster are routed with Manhattan minimum spanning trees.

We tested benchmarks r1-r5 [Tsa91b] and primary1-primary2 [JSK90] to evaluate the performance of the proposed clock synthesis methodology. Registers (clock terminals) are assumed to have the unique skew bound $0.05ns$ (smaller one of positive and negative skew bounds) for testing the benchmarks. All local clock buffers take the uniform 8X $0.6um$ CMOS inverter with the electrical data derived from [Bak87] with the output resistance 396Ω, the input capacitance $80fF$ and the intrinsic delay $0.035ns$. The global driver is $16X$ inverter size with the output resistance 198Ω. The chip parameters are provided in benchmarks by [Tsa91b, JSK90], the package and solder bump parameters are shown in



- ☑ Cells
- ☐ Fillers
- ☐ Nets
- ☐ Ports
- ☐ Text
- ☐ Fill
- ☐ Cl Attrib
- ☐ Nt Attrib
- ☐ Show X's

X: 1748.0 Y: 1850.0 DX: -395.0 DY: -104.0

Status: Cell Selected

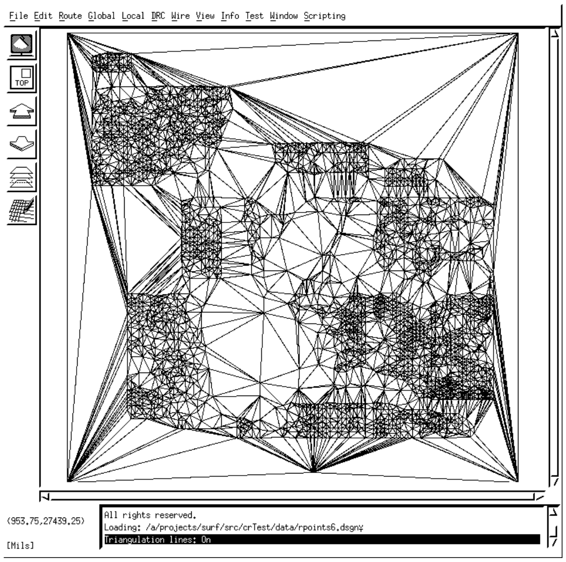
(a)

Figure 2.8: Mixed standard-cell/macrocell layout of testing chip with 3879 clocked registers.

Table 1.1. Table 2.5 shows the results. The RC delay from the global driver to the clock terminal is obtained by summing up three parts through the clock tree: d_g through the global clock tree plus the delay through the solder bump, d_b the intrinsic delay of the local buffer and d_l the delay through the local clock tree:

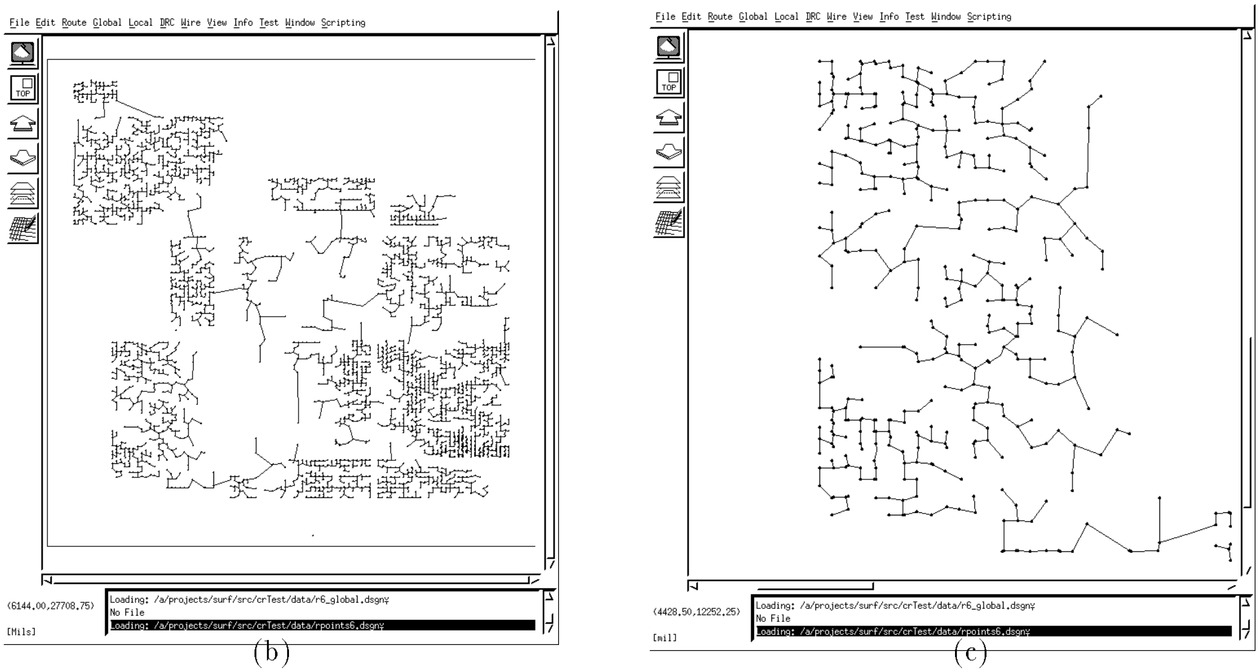
$$d = d_g + d_b + d_l$$

The solder bump adds the load capacitance of the global clock tree in the delay calculation. The skew is reported by the maximum difference of delays from the global driver to clock terminals through two-level clock trees. The dynamic power $P = CV_{dd}{}^2 f$ is linear to the total capacitance C of the clock tree, where $V_d d$ is the supply voltage and f the clock frequency. Table 2.5 shows the total capacitance which consists of four parts:



```
(953.75,27439.25)
[Mils]
All rights reserved.
Loading: /a/projects/surf/src/crTest/data/rpoints6.dsgn¥
Triangulation lines: On
```

(a)Delaunay triangulation and clock terminal distribution.



```
(6144.00,27708.75)
[Mils]
Loading: /a/projects/surf/src/crTest/data/r6_global.dsgn¥
No File
Loading: /a/projects/surf/src/crTest/data/rpoints6.dsgn¥
```

```
(4428.50,12252.25)
[mil]
Loading: /a/projects/surf/src/crTest/data/r6_global.dsgn¥
No File
Loading: /a/projects/surf/src/crTest/data/rpoints6.dsgn¥
```

(b) (c)

Figure 2.9: Testing chip: clock net clustering result. (a) Delaunay triangulation and clock terminal distribution. (b) Clustering of clock terminals; every cluster is connected by minimum Spanning tree. (c) Part of clustering result (left-bottom corner of (b))

$$C = C_w + C_t + C_b + C_s$$

where C_w is the total wire capacitance, C_t the total gate capacitance of clock terminals, C_b the total gata capacitance of local buffers, and C_s the capacitance of solder bumps.

Benchmark (Terminal Num)	Cluster Num	Skew (ps)	Longest Delay (ns)	Total Capacitance (pF)
r1 (267)	12	49.9	2.2	35.1
r2 (598)	25	49.0	3.5	72.4
r3 (862)	30	46.6	3.7	95.8
r4 (1903)	64	49.9	6.7	195.4
r5 (3101)	96	49.9	8.4	303.8
primary1 (269)	7	49.5	1.59	16.9
primary2 (603)	12	48.6	1.93	36.8

Table 2.1: Benchmark Results

We examine the performance advantages when laying the global clock tree on the package layers for benchmarks. Table 2.5 shows the skew and path delay of the global clock tree either on the package layer or on the chip layer. The global clock tree with equal path lengths is compared. On the package layer, the planar equal path length clock tree is very close to the zero skew clock tree (less than 4ps). On the chip layer, the planar equal path length clock tree probably produces a untolerable skew such that RC Elmore delay balance clock routing techniques [Tsa91b, CHJ$^+$92] are necessary. Meanwhile, the delay and power are also decreased for the clock tree on the package layer due to the far smaller interconnect RC parasitics.

2.6 Related Work

The two-level clock distribution approach is suitable for large size systems in VLSI, MCM and WSI. Anceau [Anc82], Friedman and Powell [FP84] and Embabi[BE94] proposed the modular design methodology of clock distribution. They assume the system can be partitioned into modules, and the size of each module has to be small enough so that the

	Skew (ps)			Propagation Delay (ns)			Wire Capacitance (pF)		
Benchmark	Chip	Package	Reduction (average)	Chip	Package	Reduction (average)	Chip	Package	Reduction (average)
r1	548.1	0.3	99.9%	9.9	1.1	77.2 %	8.5	4.0	54.1%
r2	3979.4	1.9		30.3	2.5		19.2	9.1	
r3	1014	0.5		34.4	2.9		22.4	10.5	
r4	1134.5	3.2		73.9	6.1		45.1	14.7	
r5	6326	2.9		112.1	8.7		62.4	29.3	
primary1	8.7	0.006		0.5	0.3		0.6	0.3	
primary2	43.5	0.03		0.9	0.5		1.2	0.6	

Table 2.2: Skew, delay and total wire capacitance results when the global clock tree is either on the package layer or on the chip layer.

conventional clock distribution schemes would yield a acceptable local skew within a worst case bound (e.g. five percent of one clock cycle time). The different clock path delays of every module are compensated for by the inter-module clock distribution. Hence, the global distribution can be achieved without interfering with the internal module design. Another advantage of the modular clock distribution is that it allows for saving power consumption by powering-up and powering-down the clock to each module upon request. This is only feasible by having independent clock nets for each module.

Our work makes three significant contributions to the modular clock distribution methodology. (a) A clustering method to generate the net partition in a top-down fashion, instead of the manual module partition assumed in previous work. (b) Use of the skew constraints of registers to guide the net partition, instead of using a worst-case skew bound. When clock frequency is rising, the worst-case skew bound is too small to obtain feasible size modules of the system. (c) A sophisticated approach for building the clock network as well as the concept of the chip/package co-design.

CHAPTER 3. Planar Equal Path Length Clock Tree Construction

Once the clock terminals are partitioned into a set of local on-chip clusters and the cluster buffers are placed, the next step in the chip/package clock network co-design is to construct a global clock tree that connects the main clock driver to each of the cluster buffers.

The global clock tree must satisfy the inter-cluster skew constraints. These constraints are derived from the initial register-register constraints and the cluster partitioning. In general, skew control is a more challenging problem for the global clock tree because it spans a larger area than the local clock trees. However, placing the global clock tree on the package mitigates this problem due to the reduced RC parasitics of package interconnects. In the experimental results we have obtained for benchmarks, when the global clock trees are routed using planar equal path length trees on the package layer, they achieve skews less than 4ps using the Elmore delay model.

Our approach places the entire global clock tree on a dedicated layer of the package. Current single-chip package and MCM technologies already use dedicated layers for power and ground networks. When the global clock tree is placed on a single package layer, delay and attenuation through vias are reduced, as well as the sensitivity to process violations between layers.

The following three steps are used to produce a single layer global clock tree which meets the cluster-cluster skew constraints:

(1) Topology generation: construct an initial planar clock tree that has equal path lengths from the source to each cluster buffer.

(2) Geometric embedding: transform the initial tree into a topologically equivalent rectilinear embedding which maintains the planarity and equal path length.

(3) Refinement: iteratively refine the clock tree in order to decrease the total wire length while maintaining the inter-cluster skew within the constraints.

The rest of this chapter deals with the first two steps: creating the initial clock tree and transforming it to a Manhattan routing. The refinement phase is addressed in the following chapter.

3.1 Related Work

H-clock trees are widely used for *symmetric* distribution of clock terminals with uniform loading capacitances, such as in systolic arrays of processing elements [DF83, FK85]. Some algorithms have been proposed to construct zero-skew clock trees for *arbitrary (non-symmetric)* distribution of clock terminals. Jackson and Kuh [JSK90] proposed the Method of Means and Medians (**MMM**), constructing a generalized H-tree in a top-down fashion. MMM recursively partitions a mass of clock terminals into two subclusters, and then connects the center of the mass to the centers of two subclusters. As analyzed [KCR91], although the difference in path lengths from clock source to clock terminals is bounded by $O(\frac{1}{\sqrt{n}})$ on the average case, MMM may produce a clock tree with path length difference as large as half the diameter of the chip. Kahng, Cong and Robins [KCR91, CKR90] proposed an efficient recursive geometric matching (**RGM**) method, constructing the clock tree in a bottom-up fashion. RGM out-performs MMM on benchmarks in the reduction of the path length skew and the total wire length. On the average, the total wire length of the clock tree produced by RGM is within a constant factor of an optimal Steiner tree, and in the worst case, bounded by $O(\sqrt{l_1 l_1} \cdot \sqrt{n})$ for n terminals arbitrarily distributed in a $l_1 \times l_2$ grid. RGM always yields a clock tree with equal path lengths for two, three, and four terminals. But no theoretical bound is given for the path length skew in general cases with more than four terminals. Instead of balancing path length in [KCR91], Tsay [Tsa91b, Tsa93] achieves zero-skew based on Elmore delay model. He proposed a linear time algorithm to compute the Elmore delay in a bottom-up order starting from terminals, and the Elmore delay evaluation is simultaneously used to guide the bottom up path matching process. This algorithm can also handle buffered clock trees with variable loading capacitances. This algorithm may need to elongate faster paths via wire "snaking" as necessary. Other algorithms [CHJ$^+$92, BK92, Eda93a, CC93, Eda94] further reduced the total wire length of the clock tree, based on the Elmore delay model. Chao, Hsu and Ho [CHJ$^+$92] proposed a deferred-merge embedding (DME) method, achieving 10% wire reduction over [Tsa91b]. Boese and Kahng [BK92] independently developed an algorithm with the identical principle of the DME, achieving 12% wire reduction. Edahiro [Eda93a, Eda94] proposed a greedy DME algorithm based on the neighbor clustering, achieving 17% wire reduction, which is the best result of current zero skew clock routing algorithms. Similar idea can also be found in [LM92]. A simulated annealing version of Elmore delay matching algorithm, was proposed by Chou and Cheng [CC93].

We proposed the first planar equal path length clock tree construction algorithm based on the motivation that a single-layer clock tree has the advantage of reducing the delay and attenuation through vias as well as the sensitivity to process variation [ZD92]. A planar equal path-length global clock tree can be routed on the top thicker layer of the chip or on a package layer with far smaller interconnect RC parasitics. Following our work,

It is feasible to route the global equal path length clock tree on a single layer, since we connect the global clock tree to sets of clusters instead of every terminal. The number of destinations of the global clock tree is significantly reduced, and the destinations are distributed in a wide area. Local area clock terminals are connected in local clock trees. Hence the planar routability of the global clock tree can be readily assured according to layout design rules.

3.3 The Max-Min Algorithm

This algorithm for constructing a planar equal path length clock tree, is a single-tree growth method. The algorithm starts from the source, and sinks are added to the tree gradually. 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 the final Steiner tree with n sinks connected). The algorithm operates as follows.

We first connect the sink, which has the maximal Manhattan distance to the source. This forms the first partial tree, T_1 (Figure 3.2(a)). As the tree grows, all sinks are classified into two types. A sink is called a *free sink*, if it has not yet been connected in the tree; otherwise called a *connected sink*. At any partial tree T_i ($1 \leq i < n$), we select a free sink and connect it to a branch of the tree, while maintaining the planarity and equal path lengths of the tree. This sink then becomes a connected sink. This may split the branch in two, since a Steiner point is inserted on it. Proceeding from Figure 3.2(b) to Figure 3.2(c), sink t_2 is chosen and connected to branch b_1. A new Steiner point s_{21}, inserted on b_1 for t_2, is said to be a *balance point*, since s_{21} has equal Manhattan distance to sink t_1 and sink t_2. Generally, a balance point s_{jk} exists for a free sink t_j 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 the connected sinks in its subtree. Further, balance point s_{jk} is called a *feasible* balance point of t_j, if a straight line

first splitting line is obtained by extending the new leaf branch b' from the sink t to the side of P_C. (ii) The second splitting line is the line that bisects the angle formed between b and b'. The dividing rule is illustrated in Figure 3.6(b).

In the conquering rule, the new leaf branch b' which connects sink t in P_C and its balance point in the leaf bounding branch b, is entirely in the convex bounding polygon P_C. After tri-partitioning P_C using the dividing rule, the new leaf branch b' becomes a leaf bounding branch for two of three new polygons.

The tree growing process preserves the next three properties.

Property 3.1: *Every bounding polygon is a convex polygon.*

Proof: The initial polygon is the rectangular bounding box, and hence is convex. The algorithm recursively uses straight lines to divide the convex bounding polygons. So, every bounding polygon is still a convex polygon. □

Property 3.2: *The tree branches are contained in the sides of the bounding polygons.*

Proof: When a new leaf branch is added, it becomes one of splitting lines that further partitions the bounding polygon (based on the dividing rule). So, this leaf branch remains on the sides of bounding polygons. □

Property 3.3: *Every bounding polygon has exactly one leaf bounding branch.*

Proof: The proof is by induction. Initially, the first leaf branch from the source to the farthest sink, is the one and only one leaf bounding branch of two convex polygons. Suppose this property holds for a bounding polygon P_C which has one and only one leaf bounding branch b. A free sink in P_C is connected to b using a new leaf branch b'. P_C is divided into three convex polygons P_{C1}, P_{C2} and P_{C3}. By the construction of the splitting lines in the dividing rule, as shown in Fig. 3.6(b), each of P_{C1}, P_{C2} and P_{C3} is bounded by either b or b', but not both. □

balance point of t_k on b. Every other free sink, t, in C has its balance point s on $\overline{s_k t_1}$, because t has a smaller balance distance than t_k. Since $\|(s_k, s)\| + \|(s, t)\| = \|(s_k, t_1)\|$, and $\|(s_k, t)\| \leq \|(s_k, s)\| + \|(s, t)\|$ (triangle inequality), we have

$$\|(s_k, t)\| \leq \|(s_k, t_1)\|. \tag{3.2}$$

According to the conquering rule, a new leaf branch $\overline{s_k t_k}$ is added which connects t_k to the balance point s_k on b. Since $\|(s_k, t_1)\| = \|(s_k, t_k)\|$, based on (3.2), we have

$$\|(s_k, t)\| \leq \|(s_k, t_k)\|. \tag{3.3}$$

The bounding polygon P_C is divided into three new bounding polygons based on the dividing rule, after branch $\overline{s_k t_k}$ is inserted. Each of these new polygons will have either $\overline{s_k t_k}$ or $\overline{s_k t_1}$ as its leaf bounding branch. So, by Property 3.4 and either (3.2) or (3.3), every remaining free sink in the new polygons will be able to find a feasible balance point on its leaf bounding branch. □

Theorem 3.1: *The clock tree produced by the divide-and-conquer algorithm is planar and has equal path length.*

Theorem 3.2: *The planar equal path length clock tree produced by the algorithm has minimal source-to-sink 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. □

An example with 18 sinks in which a planar equal path length clock tree is grown is shown in Figure 3.9. The final clock tree is shown in Figure 3.9(e). The divide-and-conquer algorithm always constructs a planar equal path length clock tree, no matter of the location of the clock source.[1] The clock tree shown in Figure 3.9(f) results from choosing the clock source inside the set of sinks.

The divide-and-conquer algorithm of constructing a planar equal path length clock tree, is summarized in the following pseudo-code.

Planar equal path length tree construction based on divide-and-conquer algorithm

```
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 $t^*$ such that
    $d(t^*, b^*) = \max_j \{d(t_j, b^*), b^*$ is the leaf bounding branch of C, $t_j \in C$ };
    Create a branch from $t^*$ to its balance point on $b^*$ resulting in new $T$;
    if $(C - \{t^*\} \neq \emptyset)${
```

---

[1] The clock source may be located at the inside of the layout. Clock signal comes from a logic element (inverter) in a VLSI chip. For pin-grid array (PGA) multichip modules, the clock source is usually set at the middle pin of the module to reduce clock skew.

$t_i^* \in C_i$ to be connected, 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.

Overall the time complexity of the tree growing 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 clusters occurs. This implies $l = n$. So, the worst-case running time of the algorithm is $O(n^2)$.

3.6 Geometrical Embedding of Planar Equal Path Length Tree

Geometrical embedding transfers each branch of a planar equal path length clock tree to a set of rectilinear wires. The tree is embedded level by level in a bottom up order, starting from leaf branches (connected to the sinks). Without the loss of generality, assume the trees are binary trees. Two branches connecting the same parent node are called *sibling branches*. For each branch $e = (v_i, v_j)$, the *bounding box* of e is the smallest rectangle which encloses the end points v_i and v_j as shown in Figure 3.11.

Ho, Vijayan and Wong [HVW89, JHVW90] proposed a linear time algorithm, that derives an *optimal cost* rectilinear Steiner tree from a given minimum spanning tree (MST), by using L-shape embedding of branches. The difference here is that the embedding needs to maintain the planarity and equal path lengths of the clock tree.

Two sibling branches are embedded together at each level. When the bounding boxes of two sibling branches intersect, the intersection defines the base line of the two branches

wires and branches except for the sibling branch, such that the planarity of the clock tree is kept.

The basic idea of Ohtsuki's line-search algorithm is as follows. Construct a set of horizontal lines by extending each horizontal edge of the obstacles until they hit another obstacle or the boundary of the region. A similar construction is done for vertical line segments. In addition, we generate two escape lines incident at target T: one horizontal (h_T) and the other vertical (v_T). Similarly, we generate horizontal and vertical line segments h_S and v_S incident at the source S, and put them into a queue. The algorithm works as follows (Figure 3.13). Each time a line segment is picked from the queue, all the line segments that intersect it are enumerated. If one of them is either h_T or v_T, then the search can be terminated and a path backtraced. Otherwise, these new line segments are put into the queue and the above process is repeated. This algorithm is guaranteed to find a path if one exists, and it is a *minimum-bend* path if multiple paths exist.

We modify Ohtsuki's algorithm for finding the rectilinear embedding of one branch in the tree, with the child node as the source S and the parent node as the target T. The routing region is the bounding box of the branch. The resultant path overlaps the base line as much as possible to reduce the wire length (Figure 3.12(a)). The modified line search algorithm is as follows.

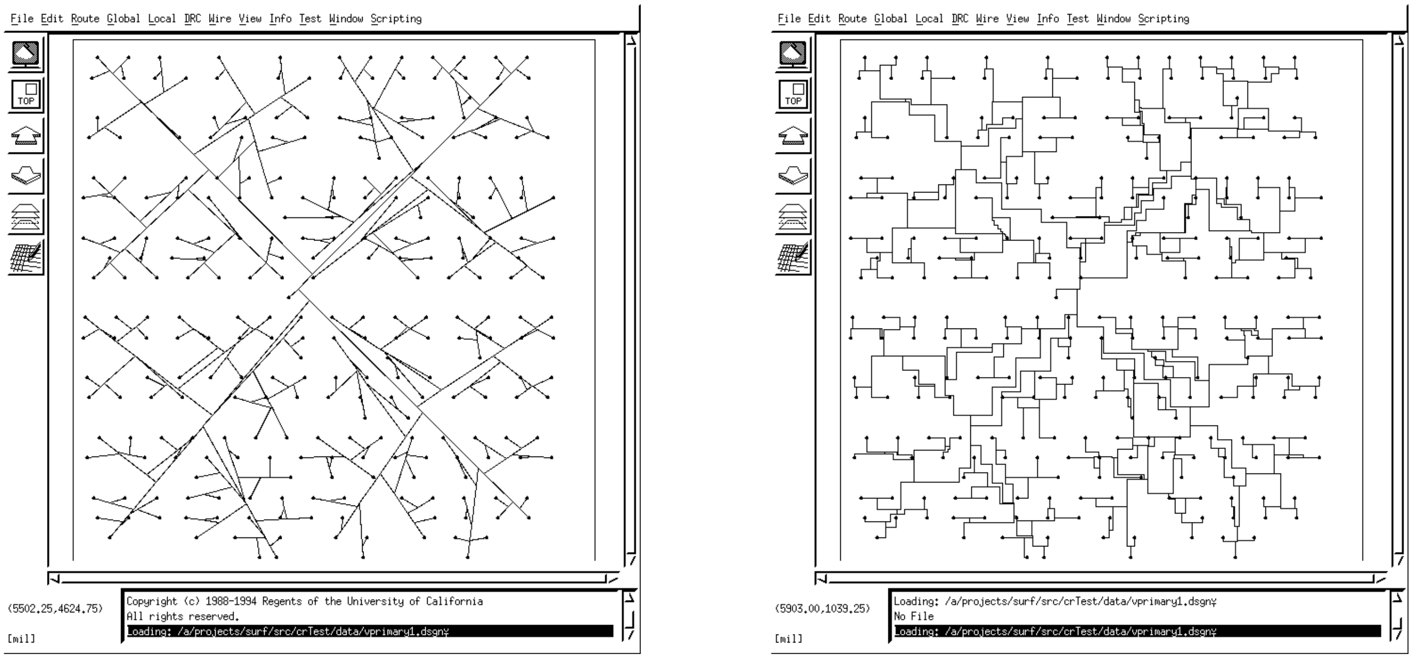
(1) Construct a set of horizontal and vertical escape line segments, with the existing of arbitrary-angle branches, by extending one horizontal and one vertical line segments from each corner of the obstacles until they hit another obstacle or the boundary of the region. If a line segment hits an arbitrary-angle branch, a new line segment is generated from the hit point in the perpendicular direction (see Figure 3.14(a)).

(2) We generate only one (not two) target line incident at T. This specific target line is in the direction of the base line. For example, if the base line is horizontal, we generate the

the planarity and equal path length well maintained. The wire length of the rectilinear clock tree, is reduced from 167.9 to 152.1, as a result of the wire sharing strategy in the geometrical embedding.

	Primary1 (269 terminals)			Primary2 (603 terminals)		
	MMM	RGM	PCR	MMM	RGM	PCR
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	167.9	406.3	376.7	422.5
Time (sec)	2.6	54.9	38.5	20.2	397.1	144.2

Table 3.1: Statistics of three clock routing algorithms on two benchmarks. CPU time is measured on SUN (Sparc 1+) Workstation.



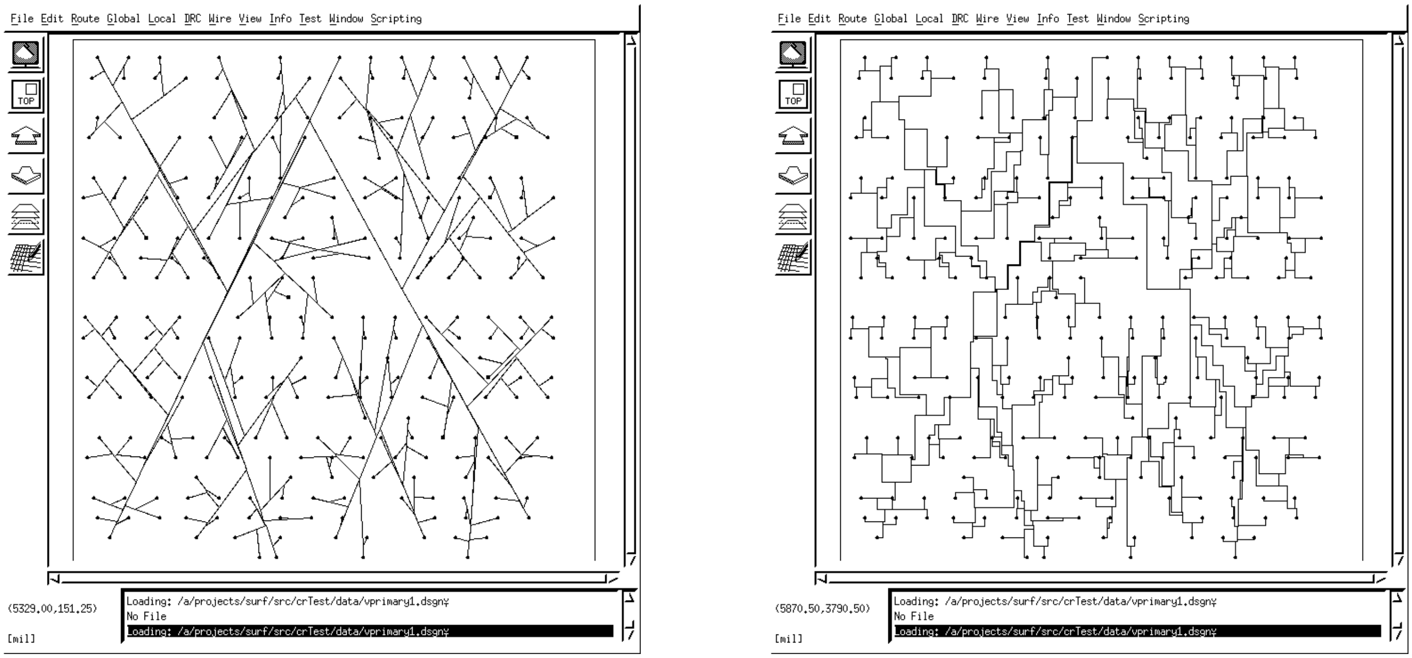
Copyright (c) 1988-1994 Regents of the University of California
All rights reserved.
Loading: /a/projects/surf/src/crTest/data/vprimary1.dsgn¥

Loading: /a/projects/surf/src/crTest/data/vprimary1.dsgn¥
No File
Loading: /a/projects/surf/src/crTest/data/vprimary1.dsgn¥

(a) (b)

Figure 3.17: Primary1: (a) Planar clock tree with equal path lengths on Primary1. (b) Geometrical embedding.

Clock source location can be flexibly selected in the tree construction. Figure 3.17 shows the Primary1 result when the clock source is located at the center of the instance; and Figure 3.18 is the result of alternatively selecting the clock source at the boundary of the instance.



```
Loading: /a/projects/surf/src/crTest/data/vprimary1.dsgn¥
No File
Loading: /a/projects/surf/src/crTest/data/vprimary1.dsgn¥
```

(a)

```
Loading: /a/projects/surf/src/crTest/data/vprimary1.dsgn¥
No File
Loading: /a/projects/surf/src/crTest/data/vprimary1.dsgn¥
```

(b)

Figure 3.18: (a) Clock source is selected on the top side of instances. (b) Geometrical embedding.

CHAPTER 4. Skew Constrained Clock Routing

In order to minimize skew, the global clock tree generated by the topological construction and geometrical embedding phases described in the previous chapter is designed to have equal length paths from the source to each local cluster buffer. However, minimizing the skew is not required — it is only necessary to maintain the skew within the cluster-cluster skew constraints. In general, an equal path length tree may have much longer total wire length than a minimum Steiner tree. This chapter describes an iterative refinement method for reducing the total wire length of the global clock tree while still satisfying the inter-cluster skew constraints.

4.1 Problem Formulation

Planar skew constrained minimum clock tree problem: *Given a clock source and a set of sinks, and a set of skew constraints for every pair of sinks, find a planar clock tree with the minimum total edge length, such that the set of skew constraints are satisfied.*

The planar skew constrained minimum clock tree construction is a NP hard problem since it becomes a minimum Steiner tree problem when we set the skew constraints to be ∞. A heuristic is proposed as follows. Starting from a planar equal path length clock tree, we iteratively refine the tree to decrease the total wire length monotonically. The tree planarity and skew constraints are always maintained in the refinement phase.

4.2 Iterative Skew Constrained Cut-and-Link

We iteratively refine the clock tree, changing its topology to form $T_0, T_1, \ldots, T_n$, where T_0 is a planar equal path length tree. The refinement is terminated when the total wire length of the clock tree can no longer be further decreased. At each refinement from T_k to T_{k+1} (initially, $k = 0$), the following conditions are satisfied:

- T_{k+1} is a planar clock tree satisfying the skew constraints.
- To refine the tree in the right search direction, we make a maximal reduction (steepest-gradient) of the total wire length at each refinement.

Cut-and-link is the key operation in the tree refinement: the tree T_k is cut to two subtrees T^a and T^b by deleting one path in the tree, then a new path previously not in the tree are added to link the two subtrees T^a and T^b again. Once a round of cut-and-link is done, T_k is refined into T_{k+1}.

We use a skew constrained shortest path to link the two subtrees again, and the cut-and-link conceptualization is shown in Figure 4.1(a). A skew-constrained shortest path p between T^a and T^b is defined as the path with the smallest cost, subject to the constraint that the new tree $T_{k+1} = T^a \cup T^b \cup p$ is a skew constrained clock tree.

A Hanan grid G is constructed based on the initial tree T_0 that is a planar equal path length rectilinear tree. The tree refinement process is done based on G. Let s be the clock source, $S_1 = \{\text{sinks}\}$, and $S_2 = \{\text{Steiner nodes in } T_0\}$. $G(V, E)$ is the Hanan grid of point set $S_1 \cup \{s\} \cup S_2$, as shown in Figure 4.2. A vertex in V is an intersection point of horizontal and vertical lines extended from all points in the set $S_1 \cup \{s\} \cup S_2$. The edge cost is the length of the edge. Note that the initial planar equal path length clock tree is fully embedded in $G(V, E)$.

algorithm is proposed to find the *subtree-to-subtree shortest path* instead of the vertex-to-vertex shortest path.

Dijkstra's algorithm solves the shortest path problem from a source s to a destination t on a weighted graph $G(V, E)$ for the case in which all edge costs are nonnegative. Every vertex v in V has the *distance estimate* which is the distance from s to v using the shortest path which has been found. Initially, s has the zero distance estimate; and all other vertices have the infinite distance estimate. The Dijkstra's algorithm starts from the source s and repeatedly explores the adjacent vertices in the graph until the destination t is explored. The algorithm maintains a priority queue Q which stores all vertices in the *search wave frontier* that are discovered but have not been explored. Initially, $Q = \{s\}$. The algorithm repeatedly selects the vertex in Q with the minimum distance estimate, and makes the relaxation of the shortest-path distance estimate for the neighbor vertices. Those neighbor vertices, which have not been in Q before, are inserted into Q and keyed by their distance estimates. The details of the Dijkstra's algorithm can be found in the textbook[CLR90].

Two subtrees T^a and T^b are disjoint in the graph. In order to find the shortest path from T^a (source subtree) to T^b (destination subtree), the Dijkstra's algorithm is modified as follows:

1. All vertices in the source subtree T^a are initialized to have the zero distance estimates, and other vertices in the graph have the infinite distance estimates. The priority queue Q is initialized to include all vertices in T^a.
2. Then, Dijkstra's algorithm is called that repeatedly selects the vertex in Q with the minimum distance estimate, makes the relaxation of the shortest-path distance estimates to the neighbor vertices, and inserts into Q the neighbor vertices which have not been in Q before[CLR90].

3. The algorithm terminates when any vertex in the destination subtree T^b is selected from Q.

The subtree-to-subtree shortest path is found in one pass by calling the above version of the Dijkstra's algorithm. A new tree is thus constructed by linking T^a and T^b using the found shortest path. The subtree-to-subtree shortest path algorithm guarantees the new tree has no self-loops. The shortest path between T^a and T^b will not cross T^a or T^b more than one time. Because vertices in T^a are initialized to have zero distance estimates, any of the vertices in T^a may only be the starting vertex (not an intermediate vertex) of the shortest path.[1] Because the algorithm terminates when one vertex in the destination subtree T^b is connected, the shortest path will not cross T^b at two vertices.

A *pseudo source* s consists of all vertices in the source subtree, and a *pseudo destination* t consists of all vertices in the destination subtree. The subtree-to-subtree shortest path is thus equivalent to the vertex-to-vertex shortest path from s to t. Self-loops at s or t are not allowed in the found path because the path is the shortest one. This observation means the subtree-to-subtree shortest path can not cross the each of two subtrees T^a and T^b more than one time.

The skew constrained shortest path between T^a and T^b has no more cost than the previously deleted path. The deleted path is also one of the skew constrained paths between T^a and T^b, but it may not be the shortest one.

The following property is the summary of the above discussion.

Property: *After a round of cut-and-link refinement, the new tree is planar, skew constrained and has smaller cost.*

---

[1] The relaxization procedure in the Dijkstra's algorithm[CLR90] will never change the distance estimates of the vertices in T^a, because vertices in T^a are initialized to have the zero distance estimate. Note that edge costs are non-negative in the graph.

A chain node in the clock tree T_k is defined as a Steiner node (not the sink or the source) which is adjacent to exactly two tree edges. As shown in 4.1(b), node A is a chain node, but node B is not.

A *collapsed tree* T'_k is derived as a collapsed version of T_k; each edge in T'_k represents a path in T_k. The algorithm gets T'_k based on T_k. First, T'_k is the copy of T_k. Then, the algorithm iterates all nodes in T'_k by the breadth-first search which starts at the source. For each chain node, the algorithm removes this node from T'_k and merges two adjacent tree edges as one tree edge in T'_k. At the end, remaining edges in T'_k represent the paths in T_k which can be used for the cut-and-link refinement.

We define the *gain* as the total wire length reduction after a round of cuting and linking the tree. The cost of a path is the sum of edge lengths of this path. Let p be a path in tree T_k with cost c, and p^* the corresponding skew constrained shortest path to be added into tree T_{k+1} with cost c^*, the *gain* g of this cut-and-link is computed as:

$$g = c - c^* \tag{4.2}$$

We compute gains of cut-and-link trees for all paths in T_k, and select the one with the *maximum gain.* Cut-and-link is performed on each path in T_k (a edge in the collapsed tree T'_k) tentatively. This is done independently for each path in the tree; thus, the current tree stays the same for each tentative cut-and-link. The selection for the definite cut-and-link refinement to transform the current tree is made after evaluating all paths in the current tree T_k.

Iterative refinement of the clock tree terminates when the maximal gain equals to zero, that is no further cost reduction. The convergency is guaranteed by the monotonical cost reduction of the tree refinements. A high-level pseudo code of the clock tree refinement phase is shown in Figure 4.5.

```
INPUT:
    $G(V,E)$ = graph, $s$ = source,
    $S$ = set of sinks,
    $C$ = set of skew constraints for sinks,
OUTPUT:
    A skew constrained clock tree spanning $S \cup \{s\}$.
PROCEDURE ClockTreeRefinement($G(V,E)$, $s$, $S$, $DB$) {
    $k = 0$;
    $T_0$ = planar equal path length clock tree;
        (using algorithm described in Chapter 4);
    do {
        $BestGain = -\infty$;
        Obtain the collapsed tree $T'_k$ from $T_k$;
        for (Each path $p$ in $T_k$ (each edge in $T'_k$)) {
            $q$ = skew constrained shortest path corresponding to $p$;
            $g$ = gain if $p$ is switched to $q$;
            if ($BestGain < g$)
                $BestGain = g$;
        }
        if ($BestGain > 0$) {
            $(p,q)$ = pair of cut-and-link paths with the gain equal to $BestGain$;
            Remove $p$ from tree $T_k$, making $T_k - p$ equal to two subtrees $T^a$ and $T^b$;
            $T_{k+1} = q + T^a + T^b$;
            $k = k + 1$;
        }
    } while ($BestGain > 0$);
}
```

Figure 4.5: Procedure of the clock tree refinement

4.3 Stochastic Analysis of Time Complexity

The convergency of the iterative tree refinement is guaranteed by the monotonical cost reduction. Based on Cayley's Theorem [Cay89], there are n^{n-2} possible spanning trees on n nodes, where n is the number of nodes in the Hanan grid. The nodes of the clock tree is a subset of total n nodes in the grid. So, the total number of possible clock trees is also $O(n^{n-2})$.

We use a probabilistic method to estimate the average number of iterations needed in the iterative tree refinement. Theorem 4.1 shows the average bound of the iteration times in a graph with n nodes. The Hanan grid, used for the skew constrained clock tree refinement, has the degree at most 4. So, Corollary 4.1 shows the average bound of the iteration times in a *degree-bounded graph* such as the Hanan grid. In the Hanan grid, the average iteration times of the skew constrained clock tree refinement is linear to the number of nodes.

To analyze the average number of iterations in the tree refinement, examine the behavior of an idealized algorithm, call it IA, which can find the optimal solution of the minimum Steiner tree. IA constructs an initial Steiner tree and iteratively refines the Steiner tree to get to the global minimum. By Cayley's Theorem [Cay89], there are n^{n-2} possible spanning trees on n nodes; Thus the number of Steiner trees spanning n_s nodes ($n_s \leq n$) is then bounded by n^{n-2}. Construct a Markov chain of n^{n-2} states, where each state corresponds to a spanning tree. Sort these states in a decreasing order from left to right with respect to the cost of the Steiner tree (not the spanning tree) breaking ties arbitrarily. Transition edges are only from a state S_i go only to a state to the right of S_i. Assume that each of the possible transitions is equally likely[2]. The transition probability from S_i to S_j is assumed to be

$$P_{ij} = \frac{1}{i-1} \text{ for } 1 \leq j < i \quad \text{and } P_{11} = 1. \tag{4.3}$$

Theorem 4.1: *The expected number of iterations executed by the tree refinement is* $O(nlog(n))$ *for* n *nodes, assuming the transition probability is assigned by (4.3).*

Proof: To analyze the maximal number of iterations in the tree refinement, examine the behavior of the idealized algorithm IA which can find the minimum Steiner tree. Based on

---

[2] This transition probability is assumed for the analytic purpose, but it is hard to realize.

the Markov chain with the transition probability in (4.3), let $\mathcal{T}_i$ be the number of transitions needed to go from state i to state 1. The expected value can be found by conditioning on the first transition. Let Y be the random variable of the next state of the first transition.

$$\begin{aligned} E[\mathcal{T}_i] &= E[E[\mathcal{T}_i|Y]] = \sum_y E[\mathcal{T}_i|Y = y]P\{Y = y\} \\ &= \sum_y E[\mathcal{T}_i|Y = y]\frac{1}{i-1} \\ &= \frac{1}{i-1}\sum_{y=1}^{i-1}(1 + E[\mathcal{T}_y]) \\ &= 1 + \frac{1}{i-1}\sum_{y=1}^{i-1} E[\mathcal{T}_y] \end{aligned}$$

Using induction with $E[\mathcal{T}_1] = 0$, it can be shown that $E[\mathcal{T}_i] = 1 + \frac{1}{i-1}\sum_{y=1}^{i-1} E[\mathcal{T}_y] = \sum_{y=1}^{i-1} 1/y \approx \log(i)$. Therefore, if IA starts in the most expensive state, i.e., $i = n^{n-2}$, then the expected number of transitions, i.e., iterations, is $O(\log(n^{n-2})) = O(n\log(n))$. This is for an ideal algorithm that finds the global minimum. The skew constrained clock tree refinement is likely to be terminated earlier in a local minimum at a middle state of the Markov chain. Thus, the maximum expected number in the skew constrained clock tree refinement is $O(n\log(n))$. □

Corollary 4.1: *The expected number of iterations executed by the tree refinement is* $O(n)$ *for* n *nodes in a degree-bounded graph (e.g. the Hanan grid), assuming the transition probability is assigned by (4.3).*

Proof: The edge number is bounded by $dn/2$ for a degree-bounded graph with the maximum degree d and n nodes. Therefore, the maximum number of possible graphs is $2^{dn/2}$, which gives an upper bound on the number of spanning trees. Following the same line of proof in Theorem 4.1, we get the expected number of the refinement iterations to be

$O(\log(2^{dn/2})) = O(n)$. $\Box$

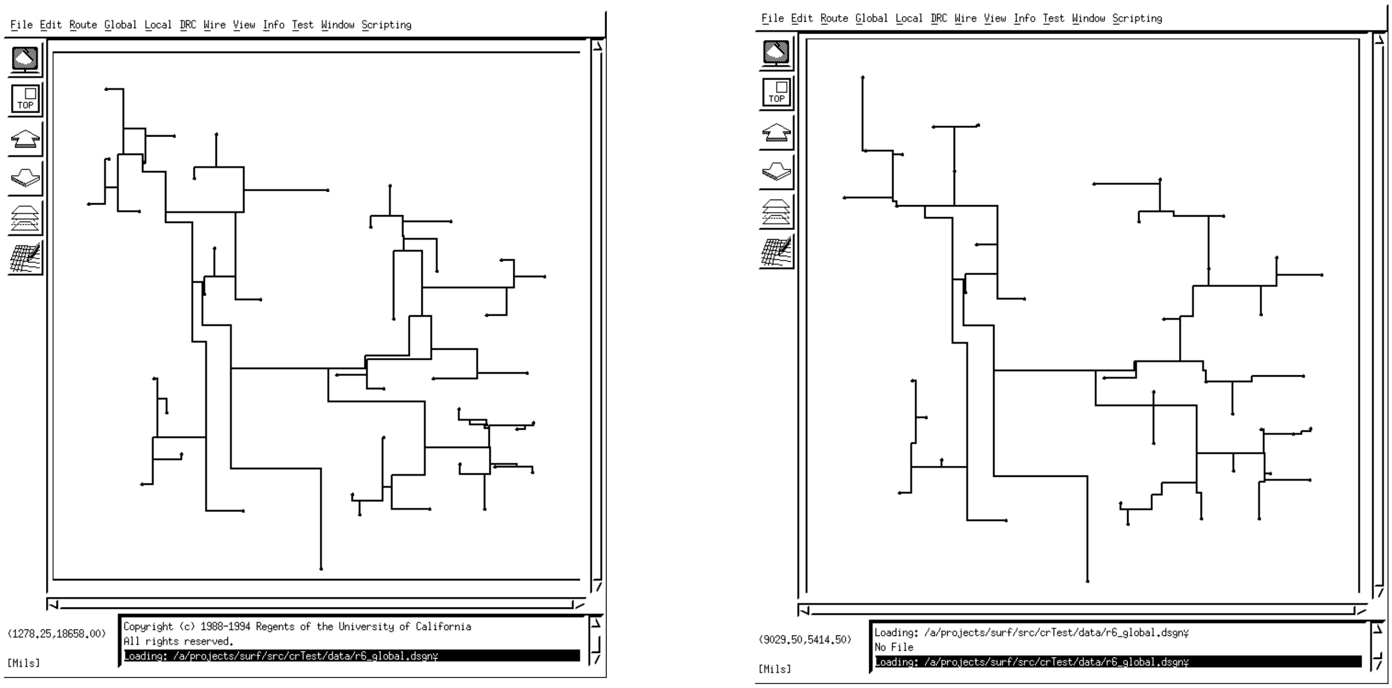
At each iteration in the refinement of the global clock tree, the running time is dominated by finding the skew constrained shortest path on the Hanan grid. The incremental construction of the skew constrained shortest path can be done [3] in $O(ks(n))$ for n nodes in Hanan grid, where $s(n)$ is the complexity of Dijkstra's shortest-path algorithm which is $s(n) = O(n^2)$. k is the number of shortest paths needs to be evaluated for the skew constrained shortest path construction. Note that k is not known a priori, and is determined only after the skew constrained shortest path is found. Each iteration of the refinement is done in $O(k|T|n^2)$ for $|T|$ edges in the tree, since we compute gains of all pairs of cut-and-link paths for selecting the one with the largest gain. The overall time complexity of the skew constrained clock tree refinement in the Hanan grid can then be obtained based on Corollary 4.1.

4.4 Experimental Results

Figure 4.6 shows the global clock tree for the testing chip from LSI Logic Corporation with 3879 registers. The chip is shown in Figure 2.8. Figure 4.6(a) is the planar equal path length clock tree, and Figure 4.6(b) the refined planar clock tree by controlling the inter-cluster skew under $0.25ps$. The total wire length is decreased by 18.9% after refinement. The maximal RC Elmore delay of the global clock tree is $1.64ns$, and the time-of-flight delay is $0.22ns$ where the longest path length of the global clock tree is $36596\mu m$ and the dielectric constant ϵ is 3.2 of the plastic package. RC delay analysis is still valid for the global clock tree on the package layer since the RC Elmore delay is much more than two times of the time-of-flight delay.

---

[3] Our implementation of the k-shortest path uses the algorithm in [Law76] which takes $O(kn^3)$.



```
(1278.25,18658.00)
[Mils]
Copyright (c) 1988-1994 Regents of the University of California
All rights reserved.
Loading: /a/projects/surf/src/crTest/data/r6_global.dsgn¥
```

```
(9029.50,5414.50)
[Mils]
Loading: /a/projects/surf/src/crTest/data/r6_global.dsgn¥
No File
Loading: /a/projects/surf/src/crTest/data/r6_global.dsgn¥
```

(a) (b)

Figure 4.6: (a) Initial global clock tree with equal path lengths. (b) Refined global clock tree.

Table 4.4 shows the total wire length and total wire capacitance of the initial global clock tree (planar equal path length clock tree) and the refined global clock tree for benchmarks r1-r5 and primary1-primary2. The total wire length reduction of global clock trees is ranged in $7 - 30\%$ under the tolerable skew bound $0.05ns$. Results are obtained when the global clock tree is laid on the package layer using RC parameters in Table 1.1.

Table 4.4 shows the total wire length comparison of the proposed two-level clock distribution scheme under tolerable skew constraints and the result obtained by a zero skew clock routing algorithm [Eda93a] which was reported to achieve the smallest total wire length among existing zero skew clock routing algorithms [Tsa91b, CHJ$^+$92, CHJ91, CC93]. The total wire length in the two-level clock distribution scheme includes the refined global clock tree and local clock trees. Making use of tolerable skew concept as in the proposed scheme

	Total Wire Length (μ m			Total Wire Capacitance (pF)	
	Initial	Refined	Reduction	Initial	Refined
r1	315696	291501	7.6%	4.0	3.7
r2	711922	497330	30.1 %	9.0	6.3
r3	830132	715882	13.8%	10.5	9.1
r4	1669150	1157671	30.6%	21.2	14.7
r5	2311002	2117701	8.4%	29.3	26.9
primary1	21159	19627	7.2%	0.27	0.25
primary2	45195	36550	19.1%	0.57	0.46

Table 4.1: Benchmarks: total wire length and wire capacitance of the global clock trees. Initial global clock tree is the planar equal path length tree, and the refined clock tree is obtained under the tolerable skew bound $0.05ns$.

can significantly reduce the total wire length of the clock trees.

	[Eda93a] (μm)	Proposed scheme (μm)	Reduction (on average)
r1	1253347	1107112	20%
r2	2483754	1994799	
r3	3193801	2628026	
r4	6499660	5031093	
r5	9723726	7782269	
primary1	129185	99943	
primary2	303994	212923	

Table 4.2: Total wire length compared with a zero skew clock routing algorithm[Eda93a]. The proposed scheme has the tolerable skew $0.05ns$.

CHAPTER 5. Clock Network Sizing

When the global clock tree is routed on the single-chip package layer, the planar equal path length clock tree is nearly a zero skew tree due to the small package RC parameters. However, when the interconnect resistance is significant such as inside the chip or on the substrate of a large scale multichip module, because of the unbalanced loading capacitances in the tree, the skew of a equal path length clock tree may be intolerable. This chapter describes a clock sizing optimization method that assigns variable wire widths for a given clock network (tree or mesh) to reduce the clock skew.

5.1 Clock Sizing Technique

Reducing the clock skew by optimizing the interconnect has centered on two techniques: one adjusting the interconnect length and the another adjusting the interconnect width.[1] The *length adjustment technique* moves the balance point or elongates the interconnect length to achieve the skew reduction, and this idea can be found in previous clock routing algorithms [Tsa91b, Tsa93, CHJ$^+$92]. The *sizing technique* achieves the skew reduction by assigning variable widths for wires. An example is shown in Figure 5.1, where the lengths from s to t_1 and t_2 are equal, but t_2 has a larger path delay than t_1 since t_2 has a larger

---

[1] Of course, there are other techniques like the buffer insertion to reduce the clock skew which is out of the scope of this dissertation.

sizing methods are based on RC delay model (not considering the inductance L), and suitable for a tree topology (not a general clock network).

In this chapter, a new optimization approach is described to perform the automatic sizing of clock wires for a given clock network. The sizing optimization considers the upper and lower bounds of wire widths which are constrained by the routing resource and the manufacturing technology. Experimental results show our method can achieve 10× skew reduction and 14% path delay reduction after the sizing. This sizing optimization method can be widely applied, due to the following two features:

- The clock distribution topology can be either a tree or a general network with loops (such as a clock mesh)
- Interconnect model and delay evaluation, incorporated in the sizing optimization, considers the inductance effect in high speed applications.

5.2 Problem Formulation

A clock network distributes the clock signal from a driver, called *clock source*, to a set of clocked elements, called *terminals*. Note that a clock network is not restricted to be a tree; loops are allowed (see Figure 5.2). The clock network consists of a set of nodes and a set of branches. Each node is one of the clock source, the clock terminals and the branch junctions in the clock network. Each branch is a segment connecting two nodes.

The sizing of a clock network is to assign feasible widths for branches to minimize the clock skew and path delay from the clock source to terminals. A *feasible width* of a branch is that bounded by the maximum and minimum allowable widths. The *maximum* allowable width of a branch is decided based on the routing resource in the layout, and the *minimum* allowable width is due to the fabrication technology. The set of possible feasible widths for

d_s: largest propagation delay from the the output of the clock source driver to the clock terminals;

The clock skew is $d_s - d_f = f(w_1, w_2, \ldots, w_n)$, and we formulate the optimal sizing of a clock network as an n-dimensional constrained optimization problem:

Objectives

$$Min\ f(w_1, w_2, \ldots, w_n) \tag{5.1}$$

Constraints

$$\begin{array}{ll} {w_1}^b \le w_1 \le {w_1}^t, & w_1 \in \{{w_1}^b, {w_1}^b + \Delta_1, {w_1}^b + 2\Delta_1, \ldots, {w_1}^t\} \\ {w_2}^b \le w_2 \le {w_2}^t, & w_2 \in \{{w_2}^b, {w_2}^b + \Delta_2, {w_2}^b + 2\Delta_2, \ldots, w_2^t\} \\ & \vdots \\ {w_n}^b \le w_n \le {w_n}^t, & w_n \in \{{w_n}^b, {w_n}^b + \Delta_n, {w_n}^b + 2\Delta_n, \ldots, {w_n}^t\} \end{array} \tag{5.2}$$

Inequality (5.2) indicates that the branch widths should be bounded and discrete according to some increments, based on the routing resource constraint and the fabrication technology. Usually, for a IC chip in a $1\mu m$ technology, branch widths are allowable between $1\mu m \sim 10\mu m$ with the increment $0.5\mu m$. For a substrate of a thin-film multi-chip module, branch widths are bounded between $10\mu m \sim 50\mu m$ with the increment $1\mu m$.

An *exhaustive enumeration method* will take $O(k^n)$ times, for n branches with k feasible widths, to iterate all possible combinations of feasible widths to obtain the global optimization. This method is not acceptable since it has exponential time complexity. We propose an efficient optimization method to solve this problem.

5.3 Optimization Method

The optimal sizing of a general clock network is an n-dimensional optimization problem. The constraints in (5.2) constitute a *feasible set* in which the solution is located. We turn the clock skew minimization into a *least-squares estimation* problem.

The clock signal is sent from the source to each clock terminal t_i $(1 \leq i \leq m)$ with a delay d_i. Define a *delay error* $g_i = d_i - d_f$ for a terminal t_i (d_f is the least delay from the source to terminals, $d_f = min(d_1, d_2, \ldots, d_m)$). For m clock terminals, the *delay error vector* is obtained as follows

$$G = (g_1, g_2, \ldots, g_m)^T = (d_1 - d_f, d_2 - d_f, \ldots, d_m - d_f)^T \tag{5.3}$$

Minimizing G means approaching d_i to d_f such that both skew and path delays are decreased.[3] We sum up the squares of these error delays:

$$\Phi(w_1, w_2, \cdots, w_n) = G^T G = \sum_{i=1}^{m} {g_i}^2 = \sum_{i=1}^{m} (d_i - d_f)^2. \tag{5.4}$$

and get the *root-mean-square (rms) error*

$$\delta = \sqrt{\Phi/m} = \sqrt{\sum_{i=1}^{m} {g_i}^2/m} \tag{5.5}$$

Theorem 5.1 shows the consistency of minimizing the skew and minimizing the *rms* error for the optimization. Note that the clock skew is equal to $d_s - d_f$ where $d_s = max(d_1, d_2, \ldots, d_m)$.

Theorem 5.1: *Given a clock network, the root-mean-square error defined in (5.5) and the*

---

[3] Minizing the skew is the major objective, however as a by-product of using our sizing method, the path delays are also decreased.

clock skew are linearly bounded by each other.

Proof: Since $d_i \leq d_s$ for all $1 \leq i \leq m$, where m is the number of clock terminals and is a constant for a given clock network, based on (5.4) and (5.5), we have $\delta = \sqrt{\sum_{i=1}^{m} (d_i - d_f)^2/m} \leq \sqrt{\sum_{i=1}^{m} (d_s - d_f)^2/m} = d_s - d_f$. On the other hand, we have $d_s - d_f \leq \sqrt{m \sum_{i=1}^{m} (d_i - d_f)^2/m} \leq \sqrt{m}\delta$. □

Most algorithms for the least-squares estimation of nonlinear parameters have centered about either of two methods [Mar63]. In one method, the objective model is expanded as a *Taylor series* and corrections to parameters which are calculated at each iteration based on the assumption of local linearity. The other method is based on the modifications of the *steepest-descent method.* Both of these two methods have their disadvantages depending on applications. While the methods based on Taylor series suffer from the possible divergence of the successive iterates, the steepest-descent based methods may have a very slow convergence of the optimum solution after the first few rapid iterations.

The Gauss-Marquardt's method [Mar63] tries to perform an optimum interpolation between the Taylor series method and the gradient based method. The optimum search is based upon the maximum neighborhood in which the truncated Taylor series gives an adequate representation of the nonlinear objective model. With the proof [Mar63], Gauss-Marquardt's method combines the best features of the previous two methods but hopefully avoids their limitations, which shares with the gradient methods the ability to converge from an initial guess quickly, and the ability of Taylor series method to reach the converged values rapidly after the vicinity of the converged values has been reached.

We use the Gauss-Marquardt's method to solve the sizing optimization problem, since Theorem 5.1 shows the consistency of minimizing the skew and the *rms* error. Set a width vector $W = \{w_1, w_2, \cdots, w_n\}^T$ for n branches. Starting with an initial solution $W^{(0)}$, W is optimized according to the following iteration formula:

$$W^{(k+1)} = W^{(k)} - (J^T J + \lambda\Lambda)^{-1} J^T G^{(k)} \tag{5.6}$$

Here, the superscript k is the iteration count ($k = 0$ initially). $G^{(k)}$ is the delay error vector shown in (5.3). J is a $m \times n$ sensitivity matrix, with the (i, j)th element $J(i, j) = \partial g_i / \partial w_j$.

To obtain J, one needs to get the sensitivity of the delay error g_i with respect to branch widths. These sensitivities are computed by the numerical differentiation method [RR78], which is applicable to a general RLC network. Λ is a diagonal matrix which takes the diagonal elements of the matrix $J^T J$, i.e.

$$\Lambda_{n\times n} = \begin{pmatrix} \sum_{i=1}^{n}(\partial g_i/\partial w_1)^2 & 0 & 0 & \cdots & 0 \\ 0 & \sum_{i=1}^{n}(\partial g_i/\partial w_2)^2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \sum_{i=1}^{n}(\partial g_i/\partial w_n)^2 \end{pmatrix}. \tag{5.7}$$

The λ in (5.6) is updated at each iteration such that the objective function is always decreased:

$$f(W^{(k+1)}) \leq f(W^{(k)}) \tag{5.8}$$

Also, the resultant widths are always bounded in the feasible range ($W^b \leq W^{(k+1)} \leq W^t$) ($W^b$ is the vector of lower bounds of branch widths, and W^t the vector of upper bounds). The strategy of selecting λ is given in Figure 5.3 [Mar63]. We take $\lambda^{(0)} = 0.02$ initially.

The above iterative optimization routine is repeated, and the clock skews are monoton-

```
$W(\lambda^{(k)})$: branch width vector at the $k$th iteration, obtained
            by the Lagrange multiplier $\lambda^{(k)}$;
$W(\lambda^{(k+1)})$: branch width vector at the $(k+1)$th iteration,
              obtained by the new Lagrange multiplier $\lambda^{(k+1)}$;
$W^b, W^t$: vectors of lower and upper bounds of branch widths, specified by designs.
$f(\lambda^{(k)})$: skew at the $k$th iteration;
$f(\lambda^{(k+1)})$: skew at the $(k+1)$th iteration;
**Step 1.** Initialize scaling parameters $\alpha$ and $\beta$; /* We take $\alpha = 4$ and $\beta = 2$ */
**Step 2.** $\lambda^{(k+1)} = \lambda^{(k)}/\alpha$;
          If $(f(\lambda^{(k+1)}) \leq f(\lambda^{(k)})$ and $W^b \leq W(\lambda^{(k+1)}) \leq W^t)$
            Return($\lambda^{(k+1)}$);
**Step 3.** $\lambda^{(k+1)} = \lambda^{(k)}$;
          If $(f(\lambda^{(k+1)}) \leq f(\lambda^{(k)})$ and $W^b \leq W(\lambda^{(k+1)}) \leq W^t)$
            Return($\lambda^{(k+1)}$);
**Step 4.** $\lambda^{(k+1)} = \lambda^{(k)} * \beta$;
          Loop {
                If $(f(\lambda^{(k+1)}) \leq f(\lambda^{(k)})$ and $W^b \leq W(\lambda^{(k+1)}) \leq W^t)$
                  Return($\lambda^{(k+1)}$);
                Else $\lambda^{(k+1)} = \lambda^{(k+1)} * \beta$;
                }
```

Figure 5.3: Strategy of selecting λ to achieve the skew reduction (or zero skew reduction) at the $(k+1)$ iteration.

ically reduced, until the skew constraints between terminals are met. In the end, we fit the width of a branch into the nearest discrete slot which is apart an increment Δ from adjacent ones, such that the constraints shown in (5.2) are well satisfied.

The iterative sizing optimization starts from an initial $W^{(0)}$. In the following, we propose an initial sizing algorithm to determine $W^{(0)}$ for a clock tree.[4] In some cases, $W^{(0)}$ already makes the clock tree to achieve a tolerable skew, such that the iterative optimization routine is not needed. Shown in Figure 5.7(a), after the initial sizing, tree A has the skew decreased from $0.27ns$ to $0.06ns$, which is tolerable for most designs.

The clock tree initial sizing algorithm starts, when all branches have the minimum width. The algorithm assigns one branch at a time with the *best* feasible width. The best feasible

---

[4] For a clock network with loops, $W^{(0)}$ is obtained by assigning the branches with the minimum width.

width makes the clock tree to achieve the smallest skew at current stage. Therefore, we propose the first rule for the clock tree initial sizing:

Rule 1: *At each stage, always assign a branch with the feasible width that results in the smallest skew of the clock tree.*

A reasonable order for sizing the branches is important to achieve a good result. Shown in Figure 5.4, node v_0 in the clock tree is supposed to have three children v_1, v_2, v_3, and one parent v_4. Branch $\overline{v_4 v_0}$ is the *parent branch* of the three branches $\overline{v_0 v_1}$, $\overline{v_0 v_2}$ and $\overline{v_0 v_3}$, and these three branches are *children branches* of $\overline{v_4 v_0}$. The clock tree is partitioned into four subtrees as shown in Figure 5.4. A set of terminals is connected in each of Subtree 1, Subtree 2 and Subtree 3. Assume that at current stage the slowest terminal with the largest path delay d_s is terminal ${t_1}^1$ which is in Subtree 1. If we enlarge the width of branch $\overline{v_0 v_1}$, the resistance of $\overline{v_0 v_1}$ is decreased but the capacitance of $\overline{v_0 v_1}$ is increased. Due to the decreased resistance of branch $\overline{v_0 v_1}$, the delays are reduced from v_0 to all terminals in Subtree 1 including the slowest terminal ${t_1}^1$. Meanwhile, due to the increased capacitance of $\overline{v_0 v_1}$, the delay is increased from the source to v_0, such that the delays are increased from the source to terminals in other subtrees (Subtree 2 and Subtree 3). So, the clock skew is reduced, and at the same time the largest path delay to ${t_1}^1$ is also reduced. The above observation is well verified in experiments.

Now we try to enlarge the width of branch $\overline{v_0 v_2}$ (or $\overline{v_0 v_3}$) shown in Figure 5.4, where the slowest terminal t_1^1 is in Subtree 1. Due to the increased capacitance of $\overline{v_0 v_2}$, the delays from the source to terminals in Subtree 1 are increased. On the other hand, due to the decreased resistance of $\overline{v_0 v_2}$, the delays are decreased from v_0 to terminals in Subtree 2. Note that the slowest terminal ${t_1}^1$ is in Subtree 1. The sizing of branch $\overline{v_0 v_2}$ has almost no help to decrease the clock skew, meanwhile the largest path delay from the source to ${t_1}^1$ may be increased. This observation is also well verified in experiments. We thus propose

Clock tree initial sizing algorithm

```
Input: a clock tree $T$, a source, a set of terminals;
Output: $T$ with assigned branch widths;

Procedure ClockTreeInitialSizing($T$) {
        $Q$ = a set of branches connected to the source;
        $t_s$ = the slowest terminal of the clock tree;
        $b$ = the ancestor branch of $t_s$ extracted from $Q$;
        $s$ = the skew of the clock tree;
        while ($b \neq NULL$) {
            Assign $b$ using the feasible width which results in the smallest $s$;
            for (each children branch $b_i$ of $b$)
                Insert $b_i$ into $Q$;
            Update $t_s$;
            $b$ = the ancestor branch of $t_s$ extracted from $Q$;
            }
        }
```

Figure 5.5: Clock tree initial sizing algorithm

wave frontier which are discovered but have not been explored. Based on Rule 2, the branch in the queue which is the ancestor branch of the slowest terminal, is always explored next. This branch is assigned by the best feasible width according to Rule 1. The search wave frontier is then expanded to the children branches, by inserting children branches into the queue. The slowest terminal may be changed. The above process is repeated, until the leaf branch, which connects to the slowest terminal, has been explored. The further sizing of remaining branches may not help the reduction for the clock skew (see the explanation of Rule 2).

The high-level pseudo code of the clock tree initial sizing algorithm is summarized in Figure 5.5.

An example for the initial sizing of a clock tree is shown in Figure 5.6.

use the wire sizing to further reduce the clock skew.

Examples	# of terminals	Maximum Path Length		Longest Branch Length		Shortest Branch Length	
		Chip (mm)	MCM (mm)	Chip (mm)	MCM (mm)	Chip (mm)	MCM (mm)
mesh	11	4.5	45.0	1.67	16.7	0.17	1.7
tree A	19	1.87	18.70	0.53	5.30	0.04	0.4
tree B	14	2.0	20.00	0.90	9.00	0.33	3.3
tree C	54	5.15	5.15	2.36	23.6	0.05	0.5

Table 5.1: Tested examples are implemented in two cases: (1) a $1\mu m$ IC chip based on the distributed RC line model; (2) a thin film substrate of a multi-chip module based on the lossy RLC line model.

Each example is tested in two cases: (1) a IC chip based on the distributed RC interconnect model, and (2) a thin-film multi-chip module based on the lossy RLC interconnect model. The clock networks (mesh and trees) are laid on one metal layer, and Table 5.4 shows the electrical parameters of a $1\mu m$ process CMOS chip and a thin-film MCM for testing. Note that clock terminals have variable load capacitances. Data of unit length resistance, capacitance and inductance are obtained based on the branch width $1\mu m$ for chips and $10\mu m$ for MCMs. The unit length resistance, capacitance and inductance, for an arbitrary branch width, can be derived based on the RLC formulae in (8.3), (8.1) and (8.2). The constraints on branch widths are set as follows. The branch widths are bounded between $1\mu m \sim 10\mu m$ for the $1\mu m$ CMOS technology, and between $10\mu m \sim 50\mu m$ for the thin-film MCM substrate.

	$R_b(m\Omega/\mu m)$	$C_b(fF/\mu m)$	$L_b(pH/\mu m)$	$R_d(\Omega)$	$C_t(pF)$	$W^*(\mu m)$
Chip	25	0.015	0	390	$0.8, 1.0$	1
MCM	8	0.06	0.38	25	1.0	10

Table 5.2: Electrical parameters of a $1\mu m$ CMOS chip and a advanced MCM design. R_b, C_b and L_b are resistance, capacitance and inductance per unit length, obtained based on the branch width $1\mu m$ for chips, and $10\mu m$ for MCMs. R_d is the driver resistance, and C_t the terminal load capacitance. W^* is the normal (minimum) width of branches.

Results of testing examples are listed in Table 5.4. The average skew reduction is high up to 78%, compared to using the minimum width of branches. With a bit increase of the path delay for the clock mesh, path delays of all clock trees are decreased after the sizing. The average (the longest) path delay reduction is 14% for clock trees.

	Clock Network with Normal Widths							
	mesh		tree A		tree B		tree C	
	Chip	MCM	Chip	MCM	Chip	MCM	(Chip)	(MCM)
Clock Skew (ns)	0.25	6.35	0.10	1.47	0.27	2.82	1.60	8.44
Largest Path Delay (ns)	2.31	11.26	1.94	7.98	5.25	15.41	19.15	57.68
Net Area (mm^2)	0.05	4.75	0.02	1.73	0.02	2.17	0.12	11.74
	Clock Network with Optimized Widths							
	mesh		tree A		tree B		tree C	
	Chip	MCM	Chip	MCM	Chip	MCM	Chip	MCM
Clock Skew (ns)	0.05	3.58	0.004	0.002	0.03	0.04	0.20	0.40
Path Delay (ns)	2.56	11.12	1.87	5.66	5.20	10.37	18.70	50.13
Net Area (mm^2)	0.15	8.15	0.05	2.95	0.07	3.57	0.18	16.05

Table 5.3: Results of testing examples, with the comparison to using the minimum width of branches.

Figure 5.7 shows the skew reduction when the iteration proceeds (the clock tree is tree A, with similar results for other clock tree examples). Clock skew is monotonically decreased during the optimization, and converges to the stable value quickly after 6 – 7 iterations. The skew reduction of the clock mesh is less than the clock trees, since the clock mesh has stronger interaction among clock terminals [5]. For clock trees, the resultant tiny skews are tolerable in today's high-speed designs. In Figure 5.7, the initial skew (the first point) is obtained based on branches assigned with the minimum width, and the second point for the clock tree is obtained by using the initial sizing algorithm which significantly dropped down the clock skew. For the clock mesh, we do not perform the initial sizing, and simply mark the second point with the same value as the first point.

We also studied the increased area due to the sizing. Figure 5.8 shows the distribution of the branch number with specified widths for testing examples. It is indicated in Figure 5.8 that the branches with smaller widths still dominate, which results in a small increase

---

[5] This result suggests us that a balanced clock tree solution is preferred than a clock mesh for the skew reduction.

30, 40, 50 and 60 times the minimum width for the MCM case. The corresponding clock source resistance R_d is shown in Table 5.4. Other parameters are the same as in Table 5.4. We simulated the skew and the largest path delay of each tested clock network with the different R_d caused by the variable driving transistor width.

	Driving Transistor Width (times of the minimum width)							
	IC				MCM			
	1	4	8	10	30	40	50	60
$R_d(\Omega)$	1560	390	195	156	52	39	31	26

Table 5.4: A set of driving transistor widths for the clock source on the technologies of a $1\mu m$ IC chip and a thin-film MCM.

Figure 5.9 is the plot of the clock skew and the largest path delay for the clock mesh and the clock tree (tree A), obtained by sizing the width of the clock driver. The path delay is obviously reduced by enlarging the transistor width, as indicated in Figure 5.9(b). However, the clock skew is slowly *increased* for the larger driver size (Figure 5.9(a)). This result provides an important clue, that a large-size clock driver have no positive effect for reducing the clock skew, even though the path delay or rise time is decreased! So other techniques such as the interconnect sizing are necessary to reduce the clock skew.

CHAPTER 6. Delay Bounded Routing

The main topic of this dissertation is the design of clock distribution networks. This chapter shows our research work on other timing critical nets. As feature sizes shrink to 0.5 micron and less, we enter the era of deep-submicron VLSI designs. According to the interconnect scaling theory[Bak87], the interconnect resistance is in proportion to the square of the scaling factor, such that the interconnect resistance becomes comparable to gate on-resistance. Meanwhile, as the gate delays and gate sizes are scaled down, the interconnect delays become more dominant. The wiring configurations of critical nets and clock nets have to be carefully designed for correct timing. For the timing issues of deep-submicron designs, interconnect optimization is becoming just as important as device optimization.

Delay constrained interconnect tree construction is a critical task in timing driven physical layout. This chapter defines the delay bounded minimum Steiner tree (DBMST) for delay constrained low cost interconnect tree construction. The iterative approach used in the refinement of the global clock tree described in Chapter 4 is applied to construct a near-optimal DBMST. The new methodology has two major impacts on the related research area: (1) using the delay bounds determined by the logic circuit static timing analysis to guide the physical interconnect tree construction, and (2) obtaining a low cost interconnect tree by satisfying the given delay bounds instead of using a minimum delay Steiner tree. A new timing driven layout framework based on DBMST construction is outlined in this chapter.

6.1 Survey of Related Research Work

The minimum Steiner tree has the smallest total cost, however, the source-to-sink path delay may be too large to satisfy the required bound. Therefore, research has been done in recent years on the interconnect tree construction based on the source-to-sink path delay requirement. People construct interconnect trees that trade-off the tree cost and the radius (the longest source-to-sink path length). One early related work is done by Ho, Lee, Chang and Wong[HLCW89] which investigated the complexity and NP-hardness of finding a bounded-diameter spanning tree or a bounded-diameter Steiner tree in a graph. Cohoon and Randall [CR91] proposed a heuristic which simultaneously considered the cost and the radius. Cong, Kahng, Robins, Sarrafzadeh and Wong [CKR+92] proposed a generalized formulation using a parameter ϵ to trade-off the radius and cost. They [CKR+92] proposed a provably good algorithm BRBC, that can always generate a tree in which the path length is bounded by $(1+\epsilon)R$ (R is the source-to-sink radius) and the wiring cost is within a factor $2(1+(2/\epsilon))$ of the optimal Steiner tree. A similar approach is also proposed by Awerbuch, Baratz and Peleg [ABP90]. Lim, Cheng and Wu [LC92] proposed a modified version of Prim's minimum spanning tree algorithm to control the radius for individual source-sinks connections. The radius-cost trade-off tree can be viewed as the one constructed between the minimum spanning tree (MST) (or minimum Steiner tree) and the shortest-path tree (SPT); Alpert, Hu, Huang and Kahng proposed a algorithm [AHHK93] which makes this MST-SPT combination in the tree construction. Cong, Leung and Zhou [CLZ93] optimized special Steiner arborescences called A-trees to make the MST-SPT trade off.

An important problem needed to be addressed is how to turn the delay bounds to the path length bounds or radius, since the delay of an RC tree depends on the topology of the tree. In the deep submicron ICs, the delay of the interconnect tree has to be computed by the distributed RC delay model[Tsa91b]. The path length is no longer accurate for the estimation of the source-to-sink delay.

Recently, minimal RC delay interconnect trees have also been investigated, which has three interesting variants: (1) minimizing the delay from the source to an identified critical sink or a set of critical sinks; and (2) minimizing the maximal source-to-sink delay; (3) minimizing the maximal delay slack. Boese, Kahng, McCoy and Robins have proposed several methods [BKMR93, BKMR94, BKR93] to minimize the delays at identified critical sinks; i.e. the optimal tree minimizes the linear combination of sink delays $f = \sum_{i=1}^{k} \alpha_i \cdot d(s_i)$, where α_i ($\alpha_i \geq 0$) is the the criticality of the sink s_i. They revealed the good fidelity of using Elmore delay model[Elm48] to guide a minimal delay interconnect tree compared to being guided by SPICE simulation. They proposed two branch and bound algorithms BB-SORT-C and BB-SORT. BB-SORT-C constructs the optimal critical sink tree to minimize the linear combination of sink delays, and the optimality is proved based on the observation [BKMR94] that the Hanan grid contains all Steiner points of the optimal critical sink tree. BB-SORT constructs the near-optimal tree that minimizes the maximal Elmore delay of the interconnect tree. Although BB-SORT-C and BB-SORT have exponential time complexity, they provides the optimal or near-optimal solutions for the empirical analysis of other heuristics. Boese, Kahng and Robins also proposed two heuristics [BKMR93, BKR93, BKMR94] for the minimum Elmore delay Steiner trees: SERT and SERT-C. SERT is a modified Prim's algorithm when adding sinks to the tree it minimizes the maximal Elmore delay of the interconnect tree. SERT-C is a modification of SERT that minimizes the delay at a single critical sink. While Hong, Xue, Kuh, Cheng and Huang [HXK$^+$93] proposed a modified Dreyfus-Wagner Steiner tree algorithm for minimizing the maximal path delay, Prasitjutrakul and Kubitz [PK90] proposed an algorithm for the minimization of the *delay* *slacks* (i.e., differences between the real delays and the given delay bounds) at sinks.

The major difficulty in the critical sink delay minimization is how to choose the accurate criticality for each critical sink to satisfy the delay bounds of multiple critical paths. Designers have to depend on intuition and perform design iterations. In typical deep submicron designs, more than 60 percent of the paths in a timing-critical design are critical [TT94].

In the rest of this chapter, we present a new approach for the interconnect tree construction under the given delay bounds. The delay bounds are decided by the static timing analysis of the logic circuits. A low cost interconnect tree is obtained by satisfying the given delay bounds instead of finding the minimum delay Steiner tree.

6.2 Delay Constrained Interconnect Tree Problem

A *signal path* in logic circuits is from a primary input to a primary output going through a set of intermediate logic cells (Figure 6.1(a)). The delay of a signal path consists of two components: delays of logical cells and delays of interconnects.

$$d_{path} = \sum_{c_i \in path} d_{c_i} + \sum_{w_i \in path} d_{w_i} \tag{6.1}$$

where d_{c_i} is the delay of logic cell c_i, and d_{w_i} the delay of interconnect w_i.

A signal path is partitioned into a set of *interconnect segments* between logic cells. The longest (critical) path delay is bounded by $\mathcal{D}$ ($d_{path} \leq \mathcal{D}$); $\mathcal{D}$ is decided by the required clock frequency and critical path throughput speeds [NBHY89]. Delay bound for each interconnect segment is determined such that the delay bound $\mathcal{D}$ of the entire signal path is guaranteed. Hauge, Nair and Yoffa [HNY87] proposed the zero-slack algorithm that computes the maximal allowable delays for individual interconnect segments, based on the static timing analysis of logic paths. This algorithm begins by computing delay slacks based on a tentative set of interconnect delays chosen so that they meet the timing requirement. The algorithm increases the delays in this set until they are maximized in the sense they still satisfy the requirements, but a further delay increase on any connection would produce a violation. Therefore, the delay bounds are obtained for the interconnect tree. Methods for the delay bound generation can also be found in [Luk91, YLS92, Fra92].

The delay bound assignment process specifies the delay bound for individual interconnect segments in order to guide the timing driven physical layout stage. A *delay constrained netlist* is defined as a set of locations of source cell and sink cells in the layout, plus the delay bounds for each source-to-sink connection (See Figure 6.1(b)). Based on the delay constrained netlist, a interconnect tree is constructed to minimize the tree cost while satisfying the source-to-sink delay bounds.

The minimum Steiner trees minimize the total wire capacitance but not the wire resistance. In the following situations, the high interconnect resistance has the significant impact on the path delay, such that the net topology has to be optimized to meet the delay constraints.

- **Technology migration**: the interconnect resistance is increased due to the decreased feature size and the increased chip dimension.
- **Mismatch between the logic design and the physical layout**: the delay bounds of the interconnect tree is decided in the logic design stage, but the interconnect tree is constructed in the physical layout stage.
- **Global nets**: the most timing critical nets are those spanning a wide layout area such that the interconnect delays are dominant. The cases of global nets are from function block to function block in chips or from chip to chip in MCMs.

In the following, a delay bounded minimum Steiner tree (DBMST) is used for the

T_0 is thus a max-delay-slack tree based on the second condition. Obviously, a max-delay-slack tree helps T_0 to satisfy the first condition: sinks have non-negative slacks. If a max-delay-slack tree still cannot have the non-negative slacks, the locations of the source and sinks or the logic circuits have to be updated.

Max-delay-slack tree construction has been proved to be a NP-hard problem [ZPD94, BKMR94]. Prasitjutrakul and Kubitz[PK90] proposed a heuristic that constructs a max-delay-slack tree by connecting sinks to the tree in the order of the closeness to the tree. This enforced order based on neighborness will definitely lose the delay optimization [BKR93].

Natural order of sink connections should be related to the *delay slacks*, instead of the physical closeness, since our goal is to maximize the minimum delay slack and sinks have variable delay bounds. We propose a new algorithm in order to construct the max-delay-slack tree driven by the global information of delay slacks at sinks. The tree is grown from the source, and at each stage, sinks are classified into two types: *connected sinks* and *free sinks*. The connected sinks have been connected to the tree, and free sinks have not. A weighted bipartite graph $G = (V_l, V_r, E)$ called *path slack graph* is defined to represent the path candidates in order to connect the free sinks to the tree. The shortest path is used to connect the free sink, because the shortest path adds the minimum additional capacitive load to the tree.

Vertices in left side (V_l) of G represent the nodes in the tree, and vertices in right side (V_r) represent the free sinks. Edges of G represent the shortest paths from every free sink to every node in the tree. Weights are assigned to edges in G to represent the minimum delay slack of the resulted routing tree, when we connect the free sink to the specific node on the tree using the corresponding shortest path. Let $t_1, t_2, \ldots, t_k$ be connected sinks, and t_{k+1} be a free sink. Let n_i be any node in the tree. The edge weight $W(n_i, t_{k+1})$ is defined as:

connected sinks. The *critical mapping* step determines which free sink is connected to the tree next. The critical mapping at one stage of tree growing is the mapping with the least weight. The free sink of the critical mapping is called *critical free sink*. In other word, the critical free sink is selected of all free sinks that will results in the smallest delay slack of the tree if this critical free sink is connected. The reasons are as follows. The objective of a max-delay-slack tree is to maximize the minimum delay slack of sinks. The optimality of the tree depends on the maximization of delay slacks for critical sinks. As we know, for a sequential router (everytime a destination is connected), early stage connections have more flexibility to maximize the delay slacks of critical sinks, and later connections have less freedom or optimization space due to the existence of connected sinks.

The algorithm starts from the source, and at each stage a critical free sink is connected to the tree using the critical mapping. The high-level pseudo code is given in Figure 6.8. T_0 shown in Figure 6.4(b) is constructed using the algorithm (In Figure 6.4(b), sinks are marked in the order of connections, i.e. t_1 is connected before t_2, etc.)

6.4.2 Dynamic Update of Path Slack Graph

Everytime a free sink is connected to the tree, we make a *incremental update* of the path slack graph G, instead of re-constructing G again, to reduce the running time. The incremental update of G is accomplished in three steps:

(a) Add on G new nodes in the tree, and add the new edges (shortest paths) on G from the new nodes to the free sinks;

(b) Update connection paths from nodes of the tree to free sinks;

(c) Update weights of edges in G.

Step (a) is done by listing new nodes on the left side of G; these new nodes are just added to T. On the right side of G, the free sink is deleted that has just been connected to

T, and p^* the new path added to T. For every path candidate p corresponding to an edge in the path slack graph G, a *correctness check* is made for p: we trace the path p, and it is a *correct path* if it has not encountered the new path p^*, elsewise it is a incorrect path. So, incorrect paths are removed from G and correct paths remain in G without recomputation.

Instead of re-computing all paths (shortest paths) from nodes on T to free sinks, we first make the correctness check to all existing paths in G. Only for incorrect paths and new nodes on T, we recompute shortest paths to free sinks. So, most of previous connection paths (correct paths) are kept in G without the re-computation.

6.5 Iterative Cut-and-Link for Refining DBMST

DBMST performs continuous delay-bounded refinement of the routing tree starting from T_0. The algorithm terminates when the tree cost can no longer be further decreased. For the feasible search optimization, at each refinement from T_k to T_{k+1}, the following conditions are satisfied.

1. To maintain the solution in the feasible region, we keep T_{k+1} delay-bounded.
2. To refine the tree in the right search direction, i.e. we make a maximal cost reduction (steepest-gradient) at each refinement.

Cut-and-link is the key operation in the tree refinement, that the tree T_k is cut to two subtrees T^a and T^b by deleting one path in the tree, then a new path previously not in the tree is added to link the two subtrees T^a and T^b again. Once a round of cut-and-link is done, T_k is refined into T_{k+1}. The cost of a path is the sum of costs of edges on this path. If the cost of the new path is smaller than the cost of the deleted one, the cost of the tree is decreased. The concept of cut-and-link tree has been explained in Chapter 4 for the refinement of the global clock tree. Instead of a skew constrained shortest path to relink two subtrees T^a and T^b, we find a delay bounded shortest path p between T^a and

T^b with the smallest cost, subject to the constraint that the new tree $T_{k+1} = T^a \cup T^b \cup p$ is a delay-bounded tree. The algorithm to find a delay bounded shortest path is the same as the skew constrained shortest path given in Chapter 4.

The high-level description of DBMST is shown in Figure 6.10. An example is illustrated in Figure 6.4.

```
INPUT:
    $G(V, E)$ = graph, $s$ = source,
    $S$ = set of sinks,
    $DB$ = set of delay bounds for sinks,
OUTPUT:
    A delay bounded Steiner tree spanning $S \cup \{s\}$.
PROCEDURE DBMST($G(V, E)$, $s$, $S$, $DB$) {
    $k = 0$;
    $T_0$ = a max-delay-slack tree;
    do {
        $BestGain = -\infty$;
        Derive the deducted tree $T'_k$ from $T_k$;
        for (Each path $p$ in $T_k$ (each edge in $T'_k$)) {
            $q$ = delay bounded shortest path corresponding to $p$;
            $g$ = gain if $p$ is switched to $q$;
            if ($BestGain < g$)
                $BestGain = g$;
        }
        if ($BestGain > 0$) {
            $(p, q)$ = pair of cut-and-link paths with the gain equal to $BestGain$;
            Remove $p$ from tree $T_k$, making $T_k - p$ equal to two subtrees $T^a$ and $T^b$;
            $T_{k+1} = q + T^a + T^b$;
            $k = k + 1$;
        }
    } while ($BestGain > 0$);
}
```

Figure 6.10: DBMST Description

Always limiting the search in the feasible region (R_b) definitely loses some opportunity for tracing the best search path to the global optimum solution. The opportunity of global optimum solution is increased if we increase the space of the feasible search region. Define a *pseudo feasible region* R'_b expanded from R_b. For a sink t_i, slack $s(i) = D(i) - d(i)$,

to the delay bound requirement, we continue to find a cut-and-link in R'_b with the maximal gain using the same algorithm as in Section 4.2. The cut-and-link in R'_b is accomplished such that the tree is allowed to have a relaxed negative minimum slack, i.e. $s^* \geq -\delta$ ($\delta \leq 0$), when we decide the delay bounded shortest path p between two subtrees T^a and T^b. p is the delay bounded shortest path which makes the tree $T = T^a + T^b + p$ with the smallest slack $s^* \geq -\delta$.

A transition cost $c_{k,k+1}$ from T_k to T_{k+1} is defined as follows, where C_{k+1} and C_k are costs of T_{k+1} and T_k.

$$c_{k,k+1} = \begin{cases} C_{k+1} - C_k & \text{if } T_{k+1} \text{ is a delay bounded tree} \in R_b; \\ +1 & \text{else } T_{k+1} \in R'_b. \end{cases} \tag{6.3}$$

If T_{k+1} is a delay bounded tree, we have $c_{k,k+1} \leq 0$. Based on (6.3), if T_{k+1} can not satisfy the delay bounds but still with $s^* \geq -\delta$ (in R'_b), we have $c_{k,k+1} = +1$ (positive). As shown in Fig. 6.11(b), the first step of cut-and-link has positive $c_{k,k+1}$ (the delay bounds are not satisfied), but we still proceed to the following $m-1$ cut-and-link trees, and select m^* cut-and-link tree refinements as the result of T_{k+1}; T_{k+1} has less cost than T_k since $c^*_{k,k+1} < 0$. If a sequence of m cut-and-link trees results in positive (+1) transition cost at every time, as shown in Fig. 6.11(c), then $T_{k+1} = T_k$, because no new delay bounded T_{k+1} can be constructed by these cut-and-link trees.

6.6 Time Complexity Analysis

We analyze the time complexities of two major stages in DBMST: constructing T_0 and the later refinement process. The time complexity of constructing T_0 is dominated by the construction of the bipartite path slack graph. The path slack graph can be constructed in $O(n^2|T|)$, where n is the number of nodes in the routing graph (not the bipartite path slack graph) and $|T|$ is the number of nodes in the tree. The dynamic update of the bipartite path slack graph can further reduce the computational time. Since m sinks are connected sequentially, the total time complexity of constructing T_0 is thus $O(m|T_0|n^2)$, where $|T_0|$ is the number of nodes in the output tree.

The average bound of iterations executed by DBMST can be obtained by the stochastic analysis as shown in Theorem 4.1 and Corollary 4.1. The expected iteration number of tree refinements executed by DBMST is $O(nlog(n))$ for n nodes in a routing graph. The expected number of tree refinements executed by DBMST is $O(n)$ for n nodes in a degree-bounded routing graph. At each iteration, the running time is dominated by finding the delay bounded shortest path which can be done in $O(kn^2)$ when using k-shortest path incremental construction [Law76], where k is the number of shortest paths evaluated when the delay bounded one is found. So, each iteration of the refinement is done in $O(k|T|n^2)$ for $|T|$ edges in the tree, since we compute gains of all pairs of cut-and-link paths for selecting the one with the largest gain.

6.7 Experimental Results

DBMST has been implemented in C++ with the delay constrained netlist as the input. We evaluate the performance of DBMST compared with existing approaches based on $0.35\mu m$ CMOS technology (Table 6.4). Experimental results are shown in Table 1 and Table 6.2. Since there are no known standard benchmarks, we take the benchmarks originally for the clock nets [Tsa91a, CHJ+92]. Note that only random examples are used in most of related papers [CKR+92, BKR93, BKMR94]. For the reason of efficiency, we partition every benchmark in multiple nets with on average 15 terminals. Table 1 and Table 6.2 shows the average result of multiple nets (r1: 18 nets, r2: 40 nets, r3: 58 nets, r4: 127 nets, r5: 207 nets, primary1: 18 nets, primary2: 40 nets).

Table 1 lists the result statistics of interconnect trees constructed by DBMST, 1-Steiner [KR92] and SERT [BKR93] based on the RC Elmore delay. Besides the delay satisfication for all testing examples, DBMST obtains the total wire length on average 0.9% higher than the total wire length obtained by 1-Steiner. On the other hand, the total wire length of DBMST is on average 38% less than that obtained by SERT. By making use of the delay bounds for the iterative tree refinement, instead of the delay minimization as in SERT, DBMST can significantly reduce the penalty of the total wire length. The running speed of DBMST is reasonably efficient since it takes 1/3 – 1/2 running time of 1-Steiner.

	Total Wire Length (μm)			Delay (ns)				CPU Time (sec)	
	1-Steiner (min-cost)	DBMST	SERT (min-delay)	1-Steiner	DBMST	Bound	SERT	1-Steiner	DBMST
r1	78943	79697	131976	8.4	5.2	6.0	4.5	74.2	25.5
r2	112341	113755	180052	15.7	9.8	10.0	7.7	77.1	25.3
r3	115092	115643	187462	17.0	9.7	10.0	8.2	78.4	25.8
r4	150834	152124	245554	26.3	15.4	16.0	13.4	77.6	24.3
r5	166356	167746	268783	32.6	19.5	20.0	15.1	80.2	25.7
Primary1	9458	9589	15612	0.18	0.14	0.15	0.11	24.5	11.5
Primary2	9563	9572	15637	0.17	0.15	0.15	0.12	20.9	10.7

Table 6.1: Experimental Comparison of DBMST, 1-Steiner and SERT. Running time is measured on a SUN Sparc-20 workstation.

Table 6.2 shows the results of DBMST and BRBC for the same testing benchmarks, based on the linear delay model, that is, the path length is used as the path delay estimation. BRBC constructs a radius (longest path length) bounded interconnect tree , and the bound of the radius is adjusted by $(1+\epsilon)d_{max}$, where d_{max} is the maximum Manhattan distance between the source to sinks and ϵ is the adjusting parameter to control the required bound of path lengths. Here, DBMST is executed by using the path length as the delay estimation for the purpose to compare BRBC.

DBMST is superior to BRBC in the total wire length as well as the longest path length. Due to the efficient iterative refinement process in DBMST, the total wire length of DBMST is 3.1 – 33.3% less than the constructive approach BRBC. When the path length bound is decreased (using the smaller ϵ), the reduction of the total wire length by DBMST is increased. For the shortest path trees ($\epsilon = 0$), the reduction of the total wire length is 33.3%. Table 6.2 also shows that the radius (longest path length) of interconnected tree by DBMST is always less than that by BRBC. We think it comes from the different methodologies used in DBMST and BRBC. DBMST starts from the shortest path length tree, and then refines the tree for small total wire length. BRBC starts from the minimum Steiner tree, and refines the tree until the satisfication of the bound on path lengths.

ϵ		Total Wire Length (μm)			Radius of Tree (μm)			
		BRBC	DBMST	Reduction (average)	Bound	BRBC	DBMST	Reduction (average)
2.0	r1	81910	79738	3.1%	73896	33626	28364	12.4%
	r2	116389	113082		105197	47587	42186	
	r3	119801	115478		110818	50257	44373	
	r4	156544	152128		144291	65134	57669	
	r5	173334	167547		152756	71042	61268	
	primary1	9764	9545		10277	4721	4310	
	primary2	9807	9558		14925	5402	5114	
1.0	r1	82432	79738	3.8%	49264	33102	28364	11.1 %
	r2	117748	113072		70131	47174	42104	
	r3	120864	115510		73878	49712	44051	
	r4	157949	152117		96194	64177	57603	
	r5	174131	167656		101837	68648	61025	
	primary1	9764	9550		6851	4721	4250	
	primary2	9807	9558		9950	5402	5114	
0.5	r1	87779	79943	8.6%	36948	30644	28146	7.2%
	r2	123325	113522		52598	43367	40617	
	r3	128428	115826		55409	46081	42932	
	r4	167047	152673		72145	58974	54771	
	r5	183053	167937		76378	64218	59202	
	primary1	10792	9611		5138	4391	4085	
	primary2	9807	9558		7462	5402	5114	
0.0	r1	128222	84933	33.3%	24632	24632	24632	0%
	r2	175547	120354		35065	35065	35065	
	r3	180060	120625		36939	36939	36939	
	r4	239142	159738		48097	48097	48097	
	r5	266388	175903		50918	50918	50918	
	primary1	15531	10274		3425	3425	3425	
	primary2	16895	9731		4975	4975	4975	

Table 6.2: Experimental Comparisons of DBMST and BRBC Algorithms

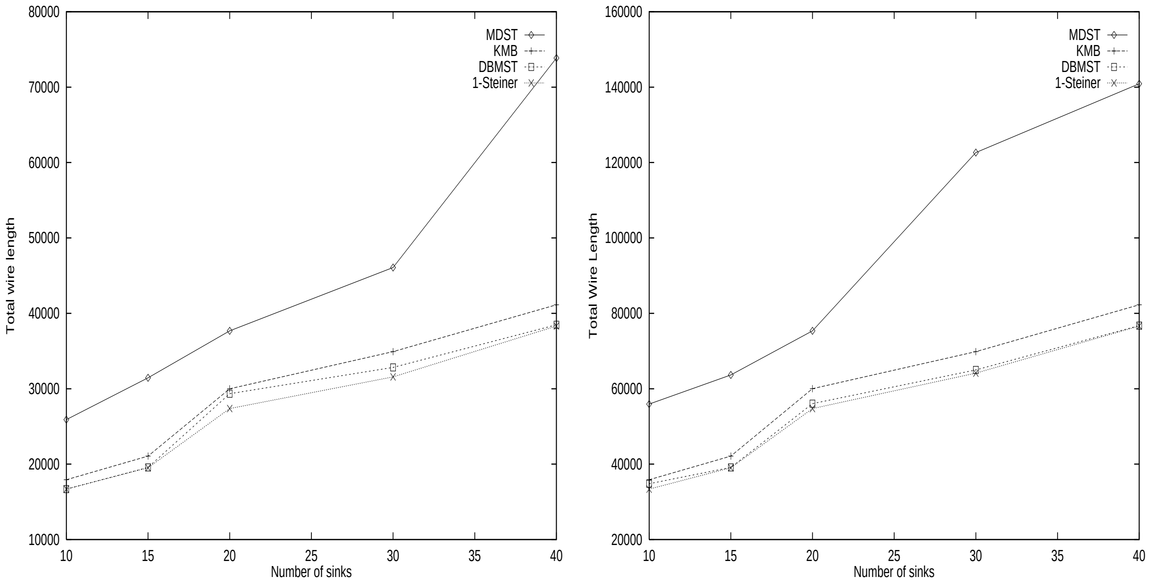
More random examples are tested to evaluate DBMST on different technologies of ICs and MCMs are shown in Table 6.4 which are obtained from MOSIS and AT&T Microeletronics Division. The compared algorithms are summarized in Table 6.3. Over a wide range of sink numbers as shown in Figure 6.12 and Figure 6.13, DBMST consistently constructs the interconnect trees with the total wire length very close to that by 1-Steiner, and smaller than that by KMB. The iterative refinement process is very efficient that reduces the total wire length on average about 60% (comparing the total wire lengths of MDST and DBMST in Figure 6.12 and Figure 6.13).

MDST	max-delay-slack tree algorithm proposed in this paper, which is the initial tree in the DBMST
DBMST	delay bounded minimum Steiner tree algorithm proposed in this paper
1-Steiner	Iterative 1-Steiner minimum Steiner tree algorithm [KR92]
KMB	minimum Steiner tree algorithm based on the minimum spanning tree approach [KMB81]

Table 6.3: List of Compared Algorithms

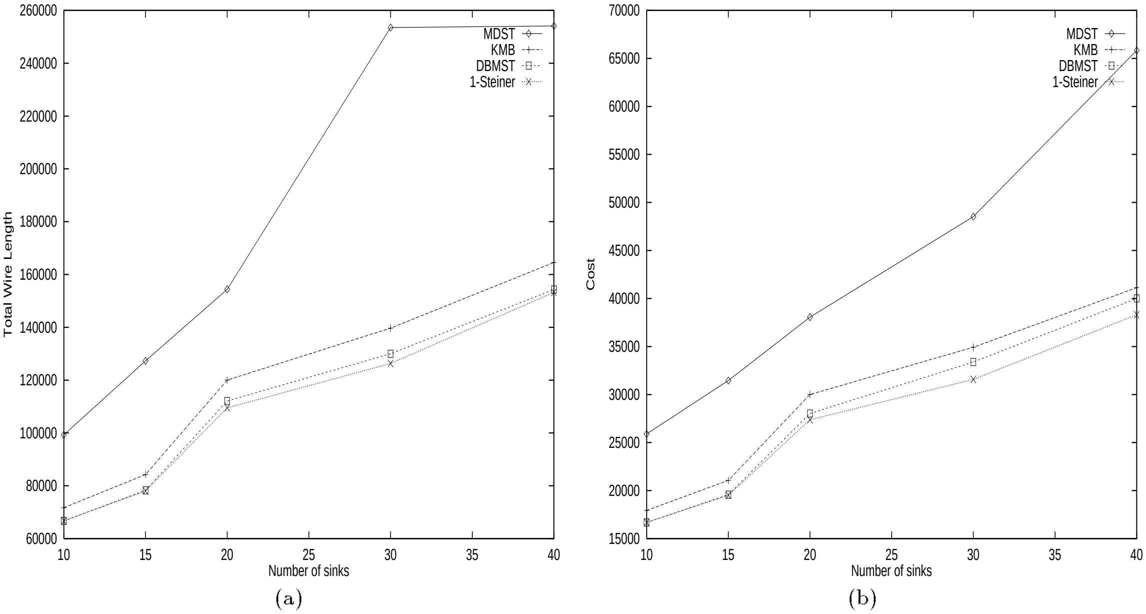
	chip size	R_b $(m\Omega/\mu m)$	C_b $(fF/\mu m)$	R_d (Ω)	C_t (pF)
1 μm CMOS	10 × 10 mm^2	30	0.02	100	0.02
0.5 μm CMOS	20 × 20 mm^2	120	0.02	100	0.01
0.3 μm CMOS	40 × 40 mm^2	480	0.02	100	0.005
MCM	100 × 100 mm^2	8	0.06	25	0.2

Table 6.4: The technologies tested in the experiments.



(a) (b)

Figure 6.12: Comparison of costs. (a) 1μm CMOS. (b) 0.5 μm CMOS.



(a) (b)

Figure 6.13: Comparison of costs. (a) 0.3μm CMOS. (b) thin-film MCM.

CHAPTER 7. Further Work and Summary

We briefly touch on three interesting aspects that are worth investigating further based on the work that has been presented in this dissertation.

7.1 Power and Ground Synthesis Based on Area I/Os

Problems associated with power and ground distribution become serious because of simultaneous switching (di/dt) noise and IR voltage drops in large chips. Since low supply voltage is becoming common for low power design, the noise margin is further reduced, and the lead inductance of wire bonds may no longer be tolerable. Flip chip technology provides a 10-20 times reduction in lead inductance compared with wire bonding. The chip and package co-synthesis scheme can be extended to power and ground networks. Exclusive package layers can be used for power mesh and ground mesh, and they can be connected to chips by area I/Os. This approach is shown in Figure 7.1.

Flip chip technology combined with this co-design scheme provides significant advantages for noise reduction. The amount of di/dt noise is proportional to the effective inductance of the power distribution network. The effective inductance is further reduced due to multiple power and ground connections from the chip to the package. Note that flip chip mounting can provide many more I/O connections than other attachment techniques. Because the power and ground nets are distributed only in local regions of the chip, the inductance

the substrate has a much larger feature size and wire spacing), so the cost of the entire VLSI system is reduced. Developing CAD tools for chip and package interconnect co-design is expected.

7.3 Interconnect Electromigration

Interconnect electromigration, caused by high current density, needs to be accounted for in the design of a large size clock network [BBB$^+$89]. Electromigration is a wear-out phenomenon, in which the *mean time before failure* (MTBF) of a metal wire predominantly depends on the current density. The current density of a wire is inversely proportional to the width of a wire. The maximum allowable current density provides lower bounds on the wire widths for the clock sizing optimization discussed in Chapter 5.

7.4 Summary

The main contributions of this Ph.D. dissertation are summarized below:

1. Since the package layer has far smaller RC interconnect parameters than a chip layer, assigning the global clock tree to the package layer can decrease the clock skew, path delay and interconnect capacitance (power) significantly. A two level clock distribution scheme is proposed based on a new design concept: chip and package co-design of clock networks. Local clock trees are routed on chip and the global clock tree is routed on the package layer. The performance advantages are demonstrated using industrial chips and a set of benchmarks.
2. Meeting the tolerable skew constraints that exist between clocked registers in digital circuits (instead of minimizing the clock skew) can simplify the clock distribution design and reduce the total wire length of the clock network. Clock net cluster-

ing/partitioning in the two level clock distribution scheme is done based on the tolerable skew constraints. Clock trees are handled differently at two levels of the clock distribution. Because of the tolerable skew and the (small) localized routing area in each cluster, the local level clock trees are routed using Manhattan minimum spanning trees so as to reduce the total wire length of the clock network. However, the inter-cluster skew constraints are usually tighter than the intra-skew constraints, and the global clock tree has a large span to balance the delays from the clock source to clusters. So, the global clock tree is initially routed as a planar equal path length clock tree. On the package layer, the clock skew of a equal path length clock tree is close to zero.

3. Using a single layer for the clock tree reduces the delay and attenuation through vias as well as the sensitivity to process variation. When the global clock tree is placed on the package, a single package layer can be used exclusively for the clock network. The global clock tree is routed as a planar equal path length tree for single layer routing. A novel algorithm is presented which constructs a planar clock tree with equal path lengths — the length of the path from the clock source to each destination is exactly the same. In addition, the path length from the source to destinations is minimized. These properties are maintained in the geometrical embedding that transforms each branch of the planar clock tree to a set of rectilinear wires.
4. An equal path length global clock tree may have much more total wire length than a minimum Steiner tree due to the equal path length requirement for meeting the skew constraints. Skew constrained iterative refinement of the global clock tree decreases the total wire length monotonically while satisfying the inter-cluster skew constraints. Skew constrained cut-and-link tree is the key operation in the iterative tree refinement.

5. To achieve path delay balance, instead of making the faster path slower by elongating wires as done in most zero skew clock routing methods, we use a wire sizing technique to make slow paths faster. This wire sizing technique can be used to reduce the clock skew of the equal path length global clock tree. This is necessary when the global clock tree has untolerable skew because it is assigned to a chip layer instead of a package layer. The Gauss-Marquartd's least square estimation method has been used to solve the clock sizing optimization problem.
6. Delay constrained interconnect tree construction is a key procedure for timing-driven layout. Two major impacts of our formulation on this problem are: (a) using delay bounds from the logic circuit static timing analysis to guide the physical interconnect tree construction; (b) obtaining a low cost interconnect tree by satisfying the given delay bounds instead of using a minimum delay Steiner tree. An iterative approach is proposed for the delay bounded minimum Steiner tree (DBMST) construction.

CHAPTER 8. Appendix: Clock Network Modeling and Delay Analysis

For a multilayer embedded wire [Fry94] as shown in Fig. 8.1, the capacitance per unit length (neglecting the sidewall capacitance) is

$$C \approx \epsilon w / t_i \tag{8.1}$$

where w is the line width, t_i the insulator thickness, and ϵ the dielectric permittivity. The inductance per unit length (neglecting the fields inside the metal itself) is

$$L \approx \mu t_i / w \tag{8.2}$$

Here, μ is the dielectric permeability. The resistance per unit length is

$$R \approx \rho / (w t_m) \tag{8.3}$$

where ρ is the metal resistivity, and t_m the wire thickness.

Each branch (wire) of a clock distribution network is modeled as a distributed RLC line, as shown in Fig. 8.2(c). The line resistance, line inductance and line capacitance is the RLC per unit length values multiplied by the line length. A RC line is used to model a branch when the inductance L is not considered. A via, with a short wire through it, can

References

[ABP90] B. Awerbuch, A. Baratz, and D. Peleg. Cost-sensetive analysis of communication protocols. In *Proc. ACM Symp. on Principles of Distributed Computing*, pages 177–187, 1990.

[AHHK93] C.J. Alpert, T.C. Hu, J.H. Huang, and A.B. Kahng. A direct combination of the prim and dijkstra constructions for improved performance-driven global routing. In *Proceedings of IEEE Intl. Symposium on Circuits and Systems*, pages 1869–1872, 1993.

[Anc82] F. Anceau. A synchronous approach for clocking vlsi systems. *IEEE Journal of Solid-State Circuits*, SC(17):51–56, 1982.

[Bak87] H. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley Publishing Company, 1987.

[BBB$^+$89] 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.

[BE94] D.E. Brueske and S.H.K. Embabi. A dynamic clock synchronization technique for large systems. *IEEE Transactions on Components, Packaging, and Manufaturing Technology. Part B: Advanced Packaging*, 17(3):350–361, Aug. 1994.

[BK92] K.D. Boese and A.B. Kahng. Zero-skew clock routing trees with minimum wirelength. In *Proc. 5th IEEE Intl. Conf. on ASIC*, pages 17–21, 1992.

[BKMR93] K. D. Boese, A. B. Kahng, B.A. Mccoy, and G. Robins. Near-optimal critical sink routing tree constructions. In *technical report TR-930027, UCLA CS Department*, pages 1–43, 1993.

[BKMR94] K. D. Boese, A. B. Kahng, B.A. Mccoy, and G. Robins. Rectilinear steiner trees with minimum elmore delay. In *Proc. of 31th Design Automation Conf.*, pages 381–386, 1994.

[BKR93] K. D. Boese, A. B. Kahng, and G. Robins. High-performance routing trees with identified critical sinks. In *Proc. of 30th Design Automation Conf.*, pages 182–187, 1993.

[Cay89] A. Cayley. A Theorem on Trees. *Quart. J. Math.*, 23:376–378, 1889.

[CC93] Nan-Chi Chou and Chung-Kuan Cheng. Wire length and delay minimization in general clock net routing. *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 552–555, 1993.

[CHJ91] 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.

[CHJ$^+$92] T.H. Chao, Y.C. Hsu, J.M.Ho, Kenneth D. Boese, and Andraw B. Kahng. Zero skew clock net routing. *IEEE Transactions on Circuits and Systems*, 39(11):799–814, November 1992.

[CK86] H.H. Chen and E.S. Kuh. Glitter: A gridless variable-width channel router. *IEEE Transactions on Computer-Aided Design*, 5(4):459–465, 1986.

[CKR90] Jason Cong, Andrew B. Kahng, and Gabriel Robins. Matching-based methods for high-performance clock routing. *IEEE Trans. on Computer-Aided Design*, CAD-12(8):1157–1169, 1990.

[CKR+92] J. Cong, A.B. Kahng, G. Robins, M. Sarrafzadeh, and C.K. Wong. Provably good performance-driven global routing. *IEEE Trans. on Computer-Aided Design*, CAD-11(6):739–752, 1992.

[CL93] Jason Cong and Kwok-Shing Leung. Optimal wiresizing under the distributed elmore delay model. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 634–639, 1993.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[CLZ92] Jason Cong, Kwok-Shing Leung, and Dian Zhou. Performance-driven interconnect design based on distributed rc delay model. In *Technical Report, University of California, Los Angeles*, pages 1–36, 1992.

[CLZ93] Jason Cong, Kwok-Shing Leung, and Dian Zhou. Performance-driven interconnect design based on distributed rc delay model. In *Proceedings of 30th ACM/IEEE Design Automation Conference*, pages 606–611, 1993.

[CR91] J. P. Cohoon and L. J. Randall. Critical net routing. In *Proc. IEEE Intl. Conf. on Computer Design*, pages 174–177, 1991.

[DF83] D.F.Wann and M.A. Franklin. Asynchronous and clocked control structures of vlsi-based interconnection networks. *IEEE Transactions on Computers*, C-32(3):284–293, March 1983.

[DW92] D. Dobberpuhl and R. Witek. A 200mhz 64b dual-issue cmos microprocessor. In *Proc. IEEE Intl. Solid-State Circuits Conf.*, pages 106–107, 1992.

[Eda93a] M. Edahiro. A clustering-based optimization algorithm in zero-skew routings. In *Proceedings of 30th ACM/IEEE Design Automation Conference*, pages 612–616, 1993.

[Eda93b] M. Edahiro. Delay minimization for zero-skew routing. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 563–566, 1993.

[Eda94] M. Edahiro. An efficient zero-skew routing algorithm. In *to appear on Proceedings of 31th ACM/IEEE Design Automation Conference*, pages 1–13, 1994.

[Elm48] W.C. Elmore. The transient response of damped linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[Fis90] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, 1990.

[FK85] Allan L. Fisher and H.T. Kung. Synchronizing large vlsi processor arrays. *IEEE Transactions on Computers*, c-34(8):734–740, August 1985.

[FP84] 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.

[Fra92] J. Frankle. Iterative and adaptive slack allocation for performance-driven layout and fpga routing. In *Proc. of 29th Design Automation Conf.*, pages 536–542, 1992.

[Fry93] Robert C. Frye. Trends in silicon-on-silicon multichip modules. In *IEEE Design & Test of Computers*, pages 8–16, 1993.

[Fry94] Robert C. Frye. Physical scaling and interconnection delay in multichip modules. *IEEE Transactions on Components, Packaging, And Manufacturing Technology. Part B: Advanced Packaging*, 17(1):30–37, February 1994.

[GFK⁺93] T. Gabara, W. Fischer, S. Knauer, R. Frye, K. Tai, and M. Lau. An i/o cmos buffer set for silicon multi chip modules(mcm). In *Proceedings of IEEE Multi-Chip Module Conference*, pages 147–152, 1993.

[HLCW89] J.M. 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.

[HNY87] P.S. Hauge, R. Nair, and E.J. Yoffa. Circuit placement for predictable performance. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 88–91, 1987.

[HVW89] J.M. Ho, G. Vijayan, and C.K. Wong. A new approach to the rectilinear steiner tree problem. In *Proceedings of 26th ACM/IEEE Design Automation Conference*, pages 161–166, 1989.

[HXK⁺93] X. Hong, T. Xue, E.S. Kuh, C.K. Cheng, and J. Huang. Performance-driven steiner tree algorithms for global routing. In *Proc. of 30th Design Automation Conf.*, 1993.

[JHVW90] G. J.M. Ho, Vijayan, and C.K. Wong. New algorithms for the rectilinear steiner tree problem. *IEEE Trans. on Computer-Aided Design*, 9:185–193, Feb. 1990.

[JSK90] 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.

[KCR91] 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.

[KL70] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49, 1970.

[KMB81] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Information*, 15:141–145, 1981.

[KR92] Andrew B. Kahng and G. Robins. A new class of iterative steiner tree heuristics with good performance. *IEEE Trans. on Computer-Aided Design*, 11(7):893–901, July 1992.

[KT94] A. Kahng and C.-W. A. Tsao. Low-cost single-layer clock trees with exact zero elmore delay skew. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 213–218, 1994.

[Law76] E. Lawler. *Combinational Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[LC92] Andrew Lim and Siu-Wing Cheng. Performance oriented rectilinear steiner trees. In *Proc. of 30th Design Automation Conf.*, pages 171–176, 1992.

[LDWC93] H. Liao, W. Dai, R. Wang, and F.Y. Chang. S-parameter based macro model of distributed-lumped networks using exponentially decayed polynomial function. In *Proceedings of 30th ACM/IEEE Design Automation Conference*, pages 726–731, 1993.

[LM92] Y.M. Li and M.A.Jabri. A zero-skew clock routing scheme for vlsi circuits. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 458–463, 1992.

[Lu91] Yizhi Lu. Dynamic constrained delaunay triangulation and application to multichip module layout. *Master Thesis, University of California, Santa Cruz*, 1991.

[Luk91] W. K. Luk. A fast physical constraint generator for timing driven layout. In *Proc. of 28th Design Automation Conf.*, pages 626–631, 1991.

[Mar63] D.W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *J. Soc. Indust. Appl. Math.*, 11:431–441, 1963.

[NBHY89] R. Nair, C.L. Berman, P.S. Hauge, and E.J. Yoffa. Generation of performance constraints of layout. *IEEE Trans. on Computer-Aided Design*, CAD-8:860–874, 1989.

[NF95] Jose Luis Neves and Eby G. Friedman. Design methodology for synthesizing clock distribution networks exploiting non-zero localized clock skew. *to appear on IEEE Transactions on VLSI Systems*, 1995.

[Oht85] T. Ohtsuki. Gridless routers – new wire routing algorithms based on computational geometry. In *Proc. of International Conference on Circuits and Systems, China*, 1985.

[Oht86] T. Ohtsuki. *Layout Design and Verification, Advances in CAD for VLSI, Volume 4, Chapter 9.* North-Holland, 1986.

[PK90] Somchai Prasitjutrakul and William J. Kubitz. A timing-driven global router for custom chip design. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 48–51, 1990.

[PMOP94] S. Pullela, N. Menezes, J. Omar, and L.T. Pillage. Skew and delay optimization for reliable buffered clock trees. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 556–562, 1994.

[PMP93] Satyamurthy Pullela, Noel Menezes, and Lawrence T. Pillage. Reliable non-zero skew clock trees using wire width optimization. In *Proceedings of 30th ACM/IEEE Design Automation Conference*, pages 165–170, 1993.

[Rhe94] WanSoo T. Rhee. A matching problem and subadditive euclidean functionals. *to be appeared in The Annals of Probability*, 1994.

[RPH83] 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.

[RR78] A. Ralston and P. Rabinowitz. *First Course in Numerical Analysis.* McGraw Hill Book Company, NewYork, 1978.

[SJDD93] David Staepelaere, Jeffrey Jue, Tal Dayan, and Wayne W-M Dai. Surf: A rubber-band routing system for multichip modules. *IEEE Design & Test Of Computers*, 10(4):18–26, Dec. 1993.

[Ste81] J. M. Steele. Subadditive euclidean functionals and nonlinear growth in geometric probability. *The Annals of Probability*, 9(3):365–376, 1981.

[Tsa91a] R.-S. Tsay. Exact zero skew. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 336–339, 1991.

[Tsa91b] R.S. Tsay. Exact zero skew. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 336–339, 1991.

[Tsa93] R.-S. Tsay. An exact zero-skew clock routing algorithm. *IEEE Trans. on Computer-Aided Design*, 12(3):242–249, 1993.

[TT94] M.Y. Mike Tsai and R.S. Tsay. Ic layout shift at deep-submicron level. *Electronic Engineering Times*, 820:66, Oct. 1994.

[YLS92] H. Youssef, R.B. Lin, and E. Shragowitz. Bounds on net delays for vlsi circuits. *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, 39(11):815–824, 1992.

[ZD92] 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.

[ZDX93] Qing Zhu, Wayne W.M. Dai, and Joe G. Xi. Optimal sizing of high speed clock networks based on distributed rc and transmission line models. In *Digest of Tech. Papers of IEEE Intl. Conf. on Computer Aided Design*, pages 628–633, 1993.

[Zhu94] Qing Zhu. Techniques for design of high-speed low-power asics. *Technical Report, LSI LOGIC Corporation*, Sep. 1994.

[ZPD94] Qing Zhu, Mehrdad Parsa, and Wayne W.M. Dai. An iterative approach for delay bounded minimum steiner tree construction. *Technical Report, UCSC-CRL-94-39, University of California, Santa Cruz.*, Oct. 1994.

[ZXDS94] Qing Zhu, Joe G. Xi, Wayne W.M. Dai, and Rama Shukla. Low power clock distribution based on area pad technology for multichip modules. In *Intl. Workshop on Low Power Design*, pages 87–92, 1994.