High-performance and Low-power Clock Network Synthesis in the Presence of Variation

by

Dong Jin Lee

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2011

Doctoral Committee:

Associate Professor Igor L. Markov, Chair
Professor David T. Blaauw
Professor Dennis M. Sylvester
Associate Professor Marina A. Epelman

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

High-performance and Low-power Clock Network Synthesis
in the Presence of Variation

by
Dong Jin Lee

Chair: Igor L. Markov

Semiconductor technology scaling requires continuous evolution of all aspects of physical design of integrated circuits. Among the major design steps, clock-network synthesis has been greatly affected by technology scaling, rendering existing methodologies inadequate. Clock routing was previously sufficient for smaller ICs, but design difficulty and structural complexity have greatly increased as interconnect delay and clock frequency increased in the 1990s. Since a clock network directly influences IC performance and often consumes a substantial portion of total power, both academia and industry developed synthesis methodologies to achieve low skew, low power and robustness from PVT variations. Nevertheless, clock network synthesis under tight constraints is currently the least automated step in physical design and requires significant manual intervention, undermining turn-around-time. The need for multi-objective optimization over a large parameter space and the increasing impact of process variation make clock network synthesis particularly challenging.

Our work identifies new objectives, constraints and concerns in the clock-network synthesis for systems-on-chips and microprocessors. To address them, we generate novel clock-network structures and propose changes in traditional physical-design flows. We develop new modeling techniques and algorithms for clock power optimization subject to tight skew constraints in the presence of process variations. In particular, we offer SPICE-accurate optimizations of clock networks, coordinated to reduce nominal skew below 5 ps, satisfy slew constraints and trade-off skew, insertion delay and power, while tolerating variations. To broaden the scope of clock-network-synthesis optimizations, we propose new techniques and a methodology to reduce dynamic power consumption by 6.8%-11.6% for large IC designs with macro blocks by integrating clock network synthesis within global placement. We also present a novel non-tree topology that is 2.3$\times$ more power-efficient than mesh structures. We fuse several clock trees to create large-scale redundancy in a clock network to bridge the gap between tree-like and mesh-like topologies. Integrated optimization techniques for high-quality clock networks described in this dissertation strong empirical results in experiments with recent industry-released benchmarks in the presence of process variation. Our software implementations were recognized with the first-place awards at the ISPD 2009 and ISPD 2010 Clock-Network Synthesis Contests organized by IBM Research and Intel Research.

PART I

Introduction & Background

CHAPTER I

Clock Network Synthesis in the Physical Design Flow

Synchronous systems consist of sequential registers (latches, flip-flops) and combinational logic connecting registers [76]. While the functional requirements of a digital system are satisfied by the register transfer level (RTL) and logic synthesis, the overall performance and timing constraints require insertion of pipeline registers to ensure that the latencies of critical paths between registers satisfy timing constraints [76]. Clock signals are delivered from a clock generator to sequential elements by a clock distribution network, which must optimize important parameters such as clock skew, slew rate, insertion delay, power dissipation, area and sensitivity to variations [6, 43]. In a modern EDA flow, the number, type and netlist of combinational logic and sequential elements are defined after RTL and logic synthesis [8, 9]. The physical locations of sequential elements

become known after the placement stage, which consists of global placement, legalization and detail placement [6, 43]. For ASIC and SoC designs, clock-network synthesis is traditionally performed after placement [43].

A clock distribution network is typically the largest net in the circuit netlist and operates at the highest speed of any signal within the entire synchronous system, hence the clock network often takes a significant fraction of the power consumed by a chip [26, 31, 66, 96]. Clock waveforms must be sharp and noise-free since all the data signals are referenced by the clock signals. Technology scaling has made long global interconnect wires significantly more resistive as wires become thinner [34]. Clock signals are particularly affected by this increased wire resistance, and precise control of clock-signal arrival times has grown in importance since they severely limit the maximum performance of the entire system.

To ensure performance and reliability, proper design and effective optimization of clock distribution networks are crucial; therefore, clock network synthesis is excluded from other signal-net routing and processed by specialized algorithms and techniques prior to global routing of signal nets [6, 43].

1.1 Industry trends

Processor-based systems fueled the development of electronics since the 1960s. PCs were the main driver of growth in electronics in the 1990s, and in the 2000s mobile phones and other battery-powered consumer devices became a significant market segment, followed by automotive electronics. These electronic systems are controlled by synchronous CPUs and ASIC chips, whose clock frequency has steadily increased for many years.

However, semiconductor scaling in the 1990s made clock optimization more challenging. While transistors continued scaling, interconnect lagged in performance [34]. The maximal length of a wire that can be driven by an inverter started a steady decrease. This phenomenon boosted demands for repeaters in clock networks, raised their power profile, and complicated their synthesis. Research in delay-driven buffering of single signal nets — arguably an easier problem and on a smaller scale — has blossomed well into the late 2000s, leaving clock-tree synthesis a difficult, high-value target. As the accuracy of compact delay models for transistors and wires deteriorated, clock-network design in the industry moved to SPICE-driven optimizations [33, 82].

A variety of clock network topologies and deskewing techniques were developed for microprocessors previously. Table 1.1 shows key parameters of clock networks in the microprocessors designed by IBM and Intel from the late 1990s to early 2000s [6, Chapter 43]. All those clock networks are regular, and only minimally adapt to sink locations. IBM S/390 used two-level balanced H-like trees. The clock network of the IBM Power4 processor consists of tuned H-trees driving a single full-chip grid. Active deskewing and wire-width tuning were employed to reduce skew. Alpha 21264 utilized hierarchical structures consisting of a global grid, six major grids and local clocks. The Intel Pentium series used spine (tall tree) structures driven by balanced binary trees. Adaptive deskewing technique based on a delay-locked loop (DLL) reduced skew from 100 ps to 15 ps in Pentium III. Deskewing by a 5-bit domain deskew register (DDR) was employed in Pentium 4. The clock network of the Intel Itanium microprocessor series features three levels of global distribution by two identical and balanced H-trees, regional clock distribution by the regional

clock driver (RCD) and regional clock grid and local clock distribution by local clock buffers (LCBs) and local clock routings. A fuse-based deskewing technique was developed for Intel Itanium 2, reducing skew from 71 ps to 24 ps. In recent high-performance microprocessors, clock signals are distributed using two-level hybrid networks consisting of a global grid and local buffered gated trees connecting to the grid [81, 82].

Processors	Year	Node, nm	Freq., MHz	Clock Topology	Deskew	Skew, ps
IBM S/390	1997	200	400	tree	—	30
IBM Power4	2002	180	1300	tree+grid	—	25
Alpha 21264	1998	350	600	grid	—	65
Pentium 2	1997	350	300	spine	—	140
Pentium 3	1999	250	650	spine	active	15
Pentium 4	2001	180	2000	spine	active	16
Itanium	2000	180	800	tree+grid	active	28
Itanium 2	2003	130	1500	tree+grid	fuse	24
ISPD 2010	2010	45	2000	tree	—	7.5

Table 1.1: Clock networks in industry CPUs [6, Chapter 43] and ISPD 2010 benchmarks from Intel and IBM (Table 2.1).

In the early 2000s, the emphasis in CPU design has shifted from high performance to power-performance-cost trade-offs, including the advent of multicore CPUs and the growing popularity of low-power ARM CPUs. In the netbook market, the low-power 1.6GHz Atom CPU from Intel is currently competing with ARM's multicore 2GHz Cortex-A9 CPUs and the 1GHz Cortex-A8, but 98% of world's mobile phones rely on ARM-based CPUs [49] which offer better power-performance-cost trade-offs than Intel CPUs [90].

ARM cores often drive system-on-chip (SoC) designs, laid out using low-power ASIC methodologies. Such methodologies perform automated clock-tree synthesis *after placement*, whereas traditional high-performance CPU methodologies pre-design clock networks and use active deskewing to lower clock skew and susceptibility to process varia-

tions [82]. Clock trees are more susceptible to variations than meshes (common in CPUs), but are 2-4 times more power-efficient. This is significant because clock networks and corresponding sequential elements consume up to 50% of CPU power and can affect power-performance comparisons between CPUs [83]. Unused parts of the clock network can be temporarily turned off (clock gating), but this does not always reduce peak power.

1.2 Research challenges

In high-quality synchronous VLSI designs, clock network synthesis is becoming a more important problem as it significantly impacts the performance, area and power dissipation of the design. The trend of increasing system complexity in conjunction with architectural-level pipelining increases the number of clocked elements [24, 108]. Semiconductor scaling facilitates smaller cycle times, but this trend assumes increasingly reliable clock distribution. The design of clock networks directly influences the maximum operating clock frequency because it determines clock skew, slew rate and insertion delay of the clocked elements [10]. Decreasing power consumption has become one of the main objectives in IC design today. The benefits of voltage reduction and device size scaling are often overwhelmed by the increase in the number of gates and clock frequency. The high costs of system cooling have also increased the importance of low-power design. Clock networks consume a significant fraction of the total system power due to its very high capacitive load and frequent switching. Being responsible for 30-50% of chip power [27, 60], clock networks require careful optimization. With shrinking cycle times, the impact of process, voltage and temperature (PVT) variation is becoming more serious and complicates the design of reliable clock networks [82]. The time it takes to design

and synthesize clock networks is becoming significant, because laborious accurate timing analysis is often required to satisfy the tight constraints for clock signals.

Nominal clock skew is usually improved first during clock routing since in GHz-range systems, performance can be seriously affected by skew in tens of picoseconds. For skew optimization, highly accurate timing analysis tools (e.g., SPICE) are required, but they are slow and dominate the runtime of clock network synthesis. Therefore, choosing appropriate timing analysis tools and how to utilize them is also important in clock network synthesis. Skew is affected by PVT variations. Hence, skew optimization based on only nominal parameters (no variation, single corner) does not guarantee a reliable clock network. We distinguish two approaches to the design of reliable clock networks. First, one can use strong devices or thick wires that are less affected by variations. Second, one can build a *redundant* clock network with multiple paths from the clock source to each clock sink, or only some clock sinks. The impact of variation on one path can be compensated for by the clock signals from the other less-affected paths. In modern clock network design, this is mostly done by using mesh/grid type structures. However, neither method can avoid increase in total capacitance, which results in an increase in total power consumption. In general, making a more robust clock network requires a significant increase in power consumption. Since reducing power dissipation is another primary goal of clock network synthesis, careful analysis of the optimal point between reliability and power consumption is mandatory in modern clock routing. Mesh/grid structure is utilized when a tree structure is insufficient to ensure a robust clock network, even after best possible optimizations. However, meshes require a dramatic resource overhead compared to

tree structures. Although some publications propose adding cross-links to harden the tree structure [36, 37, 50], recent studies suggest that these proposals are unworkable. Therefore, designing effective clock tree structures that combine the reliability of a mesh with the small footprint of a tree remains an open challenge.

Clock network synthesis for commercial designs is verified with respect to multiple process corners (or scenarios). Each corner represents a different operation environment of the chip and commercial clock network synthesis tools try to optimize the clock network based on multi-corner optimization. However, this multi-corner analysis cannot model intradie-process variations and decreases the accuracy of skew analysis as the impact of variations increases. One can utilize Monte-Carlo simulations for accurate estimation of the impact of variations, but this method is too time-consuming and remains impractical within clock network synthesis. Statistical timing analysis can model the impact of timing variations more efficiently, but remains relatively unexplored in the context of state-of-the-art clock network synthesis.

In a physical design flow, clock routing is performed after cell placement, which determines the physical locations of registers [6, 43]. Most academic/commercial placement tools do not distinguish clocked elements from combinational logic cells [20, 64, 103]. Hence, even though it is possible to improve the quality of a clock network (especially in terms of power) by modifying the locations of registers, clock network synthesis techniques are often prevented from altering the locations of registers. Some researchers proposed techniques like leaf-level register clustering [16, 75], but finding optimal register locations during placement remains an open challenge.

Producing high-quality clock networks is becoming more difficult, and related challenges may soon overwhelm those at other stages of a physical design flow because of conflicting objectives. Novel multi-objective methods are needed that can generate clock networks satisfying tight design constraints. Our solutions to these challenges are addressed in this dissertation, whose structure we outline next.

1.3 Our contributions

The contributions of this dissertation can be summarized as follows.

SPICE-accurate SoC clock network synthesis. Most existing algorithms and techniques establish fundamental methodologies for clock network synthesis, but perform large-scale optimization using analytical models that lose accuracy at recent technology nodes, and are not always validated by realistic SPICE simulations on large industry designs. In Chapter III we propose specialized optimization algorithms necessary to bridge the gaps between existing point-optimizations. We develop an EDA methodology for integrating clock-network optimization steps and describe a robust software implementation called Contango. We then extend our implementation to large industrial clock networks.

Optimization of clock trees for microprocessors. Clock networks account for a significant fraction of system power dissipation while limiting CPU performance. Therefore, power-performance-cost trade-offs are becoming a major issue in modern high-performance CPU clock design. On the other hand, the increasing impact of process variation makes clock network synthesis particularly challenging. Mesh structures are often utilized to improve robustness to variations, but significant additional power consumption is unavoid-

able. In Chapter IV we propose a tree-based solution for CPU clock routing that improves power consumption under tight skew constraints in the presence of variations. We introduce the notion of local-skew slack for clock trees, modeling and optimization of variational skew, a path-based technique to enhance robustness, a new time-budgeting algorithm for clock-tree tuning and accurate optimizations that satisfy budgets. Our strong empirical results suggest that clock trees constructed using accurate variational skew modeling and optimizations have distinct advantages in power consumption and similar robustness as meshes.

Clock network optimization during placement. Most of the existing literature for clock network synthesis assumes that register locations are given and cannot be changed. While clock networks can be improved by finding better register locations during placement, most publications do not propose such optimization, hence the quality of resulting clock networks is limited by un-optimized locations of the clocked elements. In Chapter V, we propose to optimize the locations of registers at the placement stage for power-efficient high-quality clock networks.

Closing the gap between tree and mesh structures. Common clock-network topologies can be categorized into two major types: trees and meshes. While older chips relied on trees, mesh structures were utilized to satisfy tight variation-related constraints in high-performance microprocessor designs where performance is emphasized over power consumption. However, implementation of mesh-type clock networks requires substantial amount of total wire/buffer capacitance, which significantly increases power dissipation.

In Chapter VI, we propose a novel flexible structure that maintains many advantages of tree structures, but is more robust to variations. Through in-depth structural analysis of a given clock tree, we quantitatively diagnose where and why it fails to satisfy variation-related constraints. We then go on to enhance the tree structure to attain required power-performance-robustness trade-offs.

1.4 Organization of the dissertation

The remaining part of the dissertation is organized as follows:

- Part I reviews relevant background in clock-network synthesis in Chapter II.
- Part II lays the foundation for our research. Chapter III describes our method for SPICE-accurate SoC clock network synthesis. Chapter IV describes optimizations of clock trees for microprocessors.
- Part III proposes new techniques that broaden the scope of optimization for clock network synthesis. Chapter V introduces placement optimization for registers to reduce clock-network size and total power consumption. In Chapter VI, we propose algorithms and techniques for a novel non-tree clock network structure that bridges the gap between trees and meshes.
- The dissertation concludes in Chapter VII with a summary of contributions and an outline of future research directions.

CHAPTER II

State of the Art in Clock Network Synthesis

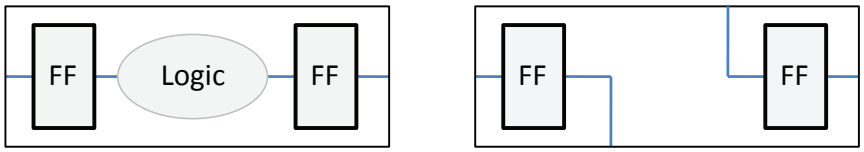
As clock networks distribute clock signals to numerous clocked elements all over the chip, they consume a sizable portion of routing resources. Their high switching activity implies significant power consumption. Hence clock networks must be carefully designed to optimize the performance of the chip, routing resource usage, and power.

This chapter covers basic terminology, core algorithms, prior work and other prerequisite topics in clock network design. Additional background information relevant to our contributions appears in further chapters. Section 2.1 discusses key parameters of a clock network and reviews the ISPD clock network synthesis contests which were held in 2009 and 2010. In Section 2.2, general types of clock networks are presented. Section 2.3 covers algorithms for clock-tree generation. In Section 2.4, existing techniques for clock network optimization during/after placement are discussed.

2.1 Key parameters of a clock network and the ISPD contests

Clock skew between two clock sinks connected to the same clock source is the absolute value of the difference in transition arrival times. The clock skew of an entire clock network is the maximum pairwise clock skew between any two sinks (more details and fur-

ther definitions are given in Section 4.2.1). *Clock jitter* refers to the time variation of the clock period at a given clock sink on the chip. The term *slew* characterizes how quickly a rising-edge or falling-edge transition occurs in a given wire. For 0V to 1V transition, 10%-90% slew can be measured by the time taken to change the value from 0.1V to 0.9V. Clock skew, jitter and slew are major issues in digital circuits, and can fundamentally limit the performance of a digital system. Therefore, clock-network synthesis must limit skew, jitter and slew. When a clock network is designed to have zero nominal skew, permanent (static) skew can occur as a result of manufacturing device and interconnect variations (i.e., process variations). Temperature gradients across a chip also contribute to skew. On the other hand, power-supply variations are the major source of jitter in clock distribution networks.[1] In this dissertation, we evaluate our clock networks using Monte-Carlo simulations with PVT variations to effectively measure clock skew affected by jitter. In other words, when we improve robustness of clock networks, we reduce not only permanent skew induced by process variations, but also temporal skew and jitter induced by voltage and temperature variations.



(a) (b)

Figure 2.1: Eligible clock sink pairs. (a) There is combinational logic between two sinks, which make the skew between these two sinks affects the useful portion of clock cycle time. (b) This sink pair is not eligible because the sinks are not logically dependent.

[1] Another major contributor to jitter is a clock-signal generator, but optimization of such a generator is beyond the scope of this dissertation.

Sink pairs eligible for local-skew calculation. In a large clock network, skew between adjacent and connected sinks is a more meaningful optimization objective than global skew [32, 81]. When two clock sinks are connected by combinational logic (Figure 2.1a) the clock skew between two sinks directly affects the useful portion of clock cycle time for the combinational logic. Otherwise, where there is no combinational logic between two sinks (Figure 2.1b), the skew between them is not a source of performance degradation, therefore we do not need to optimize the clock network to reduce the skew between those sink pairs. Eligible sink pairs for skew can be defined based on the netlist after Register-Transfer Level (RTL) synthesis so that only sink pairs that are connected by combinational logic are considered for skew calculation. In the ISPD 2010 Clock Network Synthesis (CNS) Contest, *local skew distance limit* was introduced to define the eligible sink pairs and local skew [95]. If the Manhattan distance between two sinks is less than the local skew distance limit, it is assumed that there is combinational logic between the two sinks and otherwise, there is no logic dependency. We use the same notion of local skew in our work, but do not rely on the metric definition, and all our techniques apply in a more realistic context where eligible pairs of sinks are derived directly from the netlist.

The ISPD 2009 clock-network synthesis contest organized by IBM Austin Research Lab was based on a 45 nm technology [94]. Sink latencies were evaluated by SPICE. The main objective was the difference between the least sink latency at 1.2V and the greatest sink latency at 1V. This *Clock Latency Range* (CLR) metric was intended to capture the impact of multiple power modes with different supply voltages [65], but nominal skew was also recorded. Total power was limited and the 10%-90% slew rate of 100 ps was

enforced. The CLR objective attracted significant criticism, which we share. Therefore we also evaluate our techniques in terms of nominal skew. The benchmarks were derived from industry SoC designs and include dozens to hundreds fixed rectilinear obstacles.

The ISPD 2010 high-performance clock-network synthesis contest used several 2 GHz CPU benchmarks from IBM and Intel to compare tools submitted by 10 teams across the world (down-selected from 20 initial registrants). To evaluate the quality of the clock networks, difficult slew and skew constraints were checked against 45 nm Monte-Carlo SPICE simulations that modeled PVT variations. The 100 ps slew constraints were unchanged from the ISPD 2009 contest. Clock networks that cleared all constraints were compared by their total capacitance — a proxy for dynamic power. Table 2.1 shows the statistics of the ISPD 2010 contest benchmarks.

ISPD'10 Bench.	Provider	Area, mm^2	Num. sinks	Obstacles	Δ, μm	Ω_Δ, ps
CNS01	IBM	64	1107	4	600	7.5
CNS02	IBM	91	2249	1	600	7.5
CNS03	IBM	1.51	1200	2	370	4.999
CNS04	IBM	5.73	1845	2	600	7.5
CNS05	IBM	5.9	1016	1	600	7.5
CNS06	Intel	1.74	981	0	600	7.5
CNS07	Intel	3.67	1915	0	600	7.5
CNS08	Intel	2.99	1134	0	600	7.5

Table 2.1: ISPD 2010 benchmarks based on 45 nm microprocessor designs. Ω_Δ is the *local skew limit*, and Δ is the *local skew distance limit* respectively (see Section 4.2.1). Nominal voltage is 1.0V and on-chip variations (ν) are accounted by 15% voltage variation and 10% variation of wire parasitics [95].

2.2 Clock-network topologies

The choice between a tree and non-tree topology is a central question in modern clock-network design. High-performance microprocessors typically use meshes due to their

robustness to late design changes and process variations, but at a great cost in terms of capacitance. Tree topologies offer many advantages, including simplicity, symmetry, faster timing analysis and amenability to incremental tuning. We start by surveying general types of clock networks, and will describe details of relevant algorithms in Section 2.3.

Clock trees have been widely supported by academic and commercial EDA tools. Simple methods including *H-tree* [11], the *method of means and medians* (MMM) [40], the *geometric matching algorithm* (GMA) [22] and *path length balancing method* (PLB) [42] were commonly utilized before the *deferred merge embedding* (DME) algorithm [12, 28] was introduced. Recently several methodologies for SoC clock-tree tuning have been developed with robustness improvement. A clock-synthesis methodology for SPICE-accurate skew optimization with tolerance to voltage variations was proposed in [51]. The Dynamic Nearest-Neighbor Algorithm (DNNA) to generate tree topology and the Walk-Segment Breadth First Search (WSBFS) for routing and buffering were proposed in [87]. A three-stage CTS flow based on an obstacle-avoiding balanced clock-tree routing algorithm with monotonic buffer insertion is proposed in [61]. A Dual-MST (DMST) geometric matching approach is proposed in [63] for topology construction and recursive buffer insertion. Modeling techniques and algorithms for microprocessor clock power optimization subject to local skew constraints in the presence of variations are proposed in [53].

Meshes. From the mid 1990s when the impact of PVT variation became significant, clock networks were more affected by PVT variations than random logic, due to their structure and more stringent timing constraints. In a tree network, such unexpected changes

are likely to propagate to the sinks. Mesh (or grid) structures have emerged to address the structural drawbacks of trees. In meshes, there are multiple paths from the clock source to individual clock sink; thus, the impact of variations on one path can be averaged out by multiple redundant paths [107]. However, meshes require significant overhead in terms of on-chip resources and power. Published examples suggest that mesh-type clock networks suffer much greater power consumption. Nevertheless, mesh structures were utilized to satisfy tight variation-related constraints in high-performance microprocessor designs where performance is more emphasized than power consumption [6, 43]. Some methods to analyze the characteristics of mesh structures are proposed in [19, 106] and a combinatorial algorithm to optimize a clock mesh is proposed in [98]. An obstacle-avoiding clock mesh synthesis method which applies a two-stage approach of mesh construction followed by driving-tree synthesis is proposed in [86,105]. A methodology based on binary linear programming for clock mesh synthesis is described in [21].

Trees with cross-links. The dichotomy between meshes and trees is striking, and several researchers attempted to find intermediate topologies that would retain the advantages of meshes but reduce capacitance overhead. A key idea in the literature is to insert cross-links into clock trees, creating redundant paths to sinks that contribute to nominal or variational clock skew [77, 78]. These methods are later extended to handle buffered clock trees in [79, 99]. Most publications discuss cross-links that directly connect pairs of sinks. Surprisingly, none of these techniques were useful at the ISPD 2009-2010 clock-network contests [94, 95] despite diligent attempts, as improved tree-tuning methods were sufficient. Careful experiments and analytical estimates [67] have shown that direct cross-links are

only effective in poorly tuned clock trees and/or at relatively short distances. However, in high-quality clock trees it is rare to find a critical pair of sinks at a short distance. A recent proposal [67] suggests adding cross-links higher in the tree to connect entire branches. As several other publications with strong empirical results, [67] uses unrealistically large composite buffers, and arranges them in a unique two-layer configuration (10+40 small inverters). Given that the ISPD 2010 contest infrastructure does not adequately model such configurations, the competitiveness of cross-links in practice remains unclear.

2.3 Algorithms for clock tree construction and buffering

The first geometric algorithms for clock routing evaluated skew in terms of wirelength from the source to sinks and produced minimum-wirelength trees for a given sink clustering using *the deferred merging and embedding* (DME) principle [12].

DME algorithms [43]. The deferred-merge embedding (DME) algorithm defers the choice of merging (tapping) points for subtrees of the clock tree. DME optimally embeds any given topology over the sink set S: the embedding has minimum possible source-sink linear delay, and minimum possible total tree cost. The algorithm was independently proposed by several groups - Boese and Kahng [12], Chao et al. [17], and Edahiro [28].

In the Manhattan geometry, two sinks in general position will have an infinite number of midpoints, creating a tilted line segment, or Manhattan arc (Figure 2.2 [43]); each of these midpoints affords the same minimum wirelength and exact zero skew. Ideally, the selection of embedding points for internal nodes will be delayed for as long as possible.

The DME algorithm embeds internal nodes of the given topology G via a two-phase process. The first phase of DME is bottom-up, and determines all possible locations of

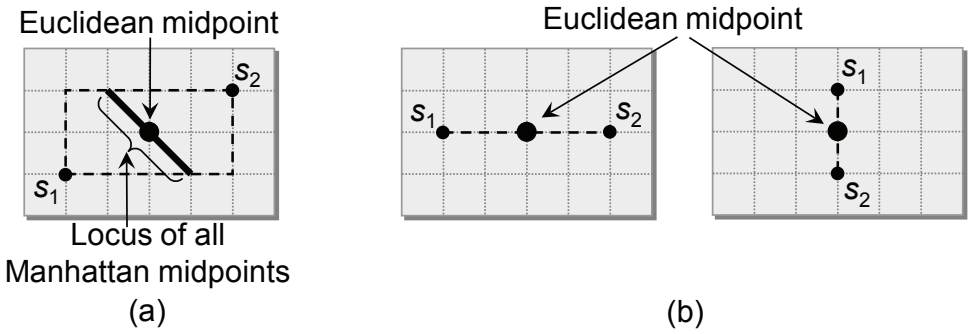


Figure 2.2: The locus of all midpoints between two sinks s_1 and s_2 is a Manhattan arc in the Manhattan geometry. On the other hand, the midpoint is unique in Euclidean geometry. (a) Sinks s_1 and s_2 are not horizontally aligned. Therefore, the Manhattan arc has non-zero length. (b) Sinks s_1 and s_2 are horizontally (left) and vertically (right) aligned. Therefore, the Manhattan arc for both cases has zero length. Source: [43].

internal nodes of G that are consistent with a minimum-cost ZST T. The output of the first phase is a tree of line segments, with each line segment being the locus of possible placements of an internal node of T. The second phase of DME is top-down, and chooses the exact locations of all internal nodes in T. The output of the second phase is a fully embedded, minimum-cost ZST with topology G.

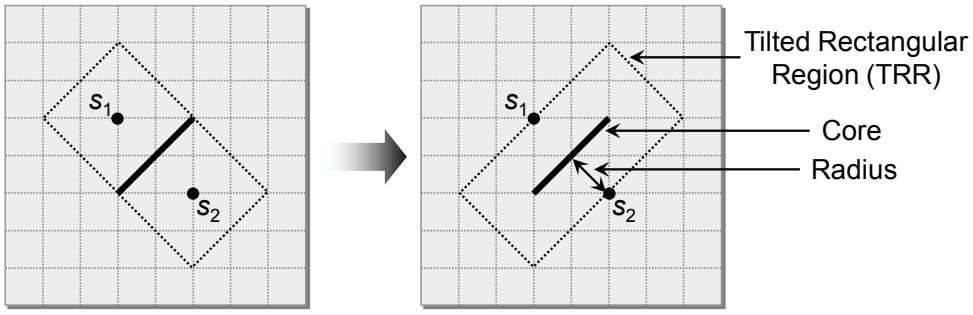


Figure 2.3: (a) Sinks s_1 and s_2 form a Manhattan arc. (b) An example of a tilted rectangular region (TRR) for the Manhattan arc of s_1 and s_2 with radius of two units. Source: [43].

A tilted rectangular region (TRR) is a collection of points within a fixed distance of a Manhattan arc (Figure 2.3 [43]). The core of a TRR is the subset of its points at maximum

distance from its boundary, and its radius is the distance between its core and boundary.

The merging segment of a node v in the topology, denoted by $ms(v)$, is the locus of feasible locations for v, consistent with exact zero skew and minimum wirelength (Figure 2.4 [43]). The following presents the sub-algorithms used for DME.

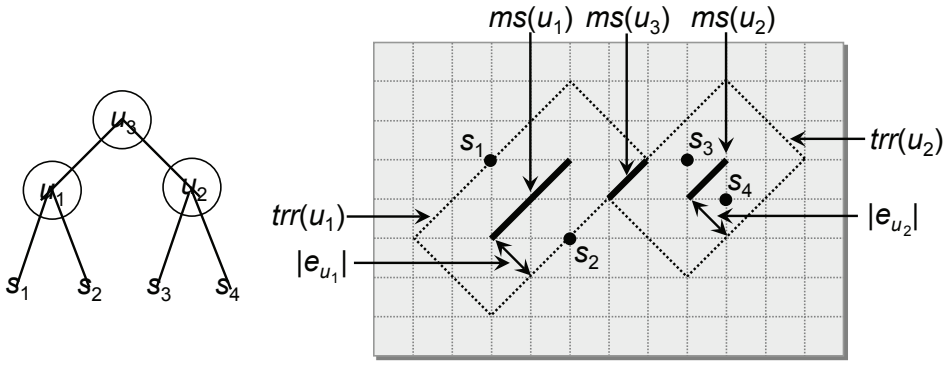


Figure 2.4: A bottom-up construction of the merging segment $ms(u_3)$ for node u_3, the parent of nodes u_1 and u_2, given the topology on the left. The sinks s_1 and s_2 form the merging segment $ms(u_1)$, and the sinks s_3 and s_4 form the merging segment $ms(u_2)$. The two segments $ms(u_1)$ and $ms(u_2)$ together form the merging segment $ms(u_3)$. Source: [43].

The bottom-up phase of DME (building a tree of segments) starts with all sink locations S given. Each sink location is viewed as a (zero-length) Manhattan arc. If two sinks have the same parent node u, then the locus of possible placements of u is a merging segment (Manhattan arc) $ms(u)$. In general, given the Manhattan arcs that are the merging segments of two nodes a and b, the merging segment of their parent node is uniquely determined due to the minimum-cost property, and is itself another Manhattan arc (Figure 2.4 [43]). The edge lengths $|e_a|$ and $|e_b|$ are uniquely determined by the minimum-length and zero-skew requirements. As a result, the entire tree of merging segments can be constructed bottom-up in linear time (Figure 2.5 [43]).

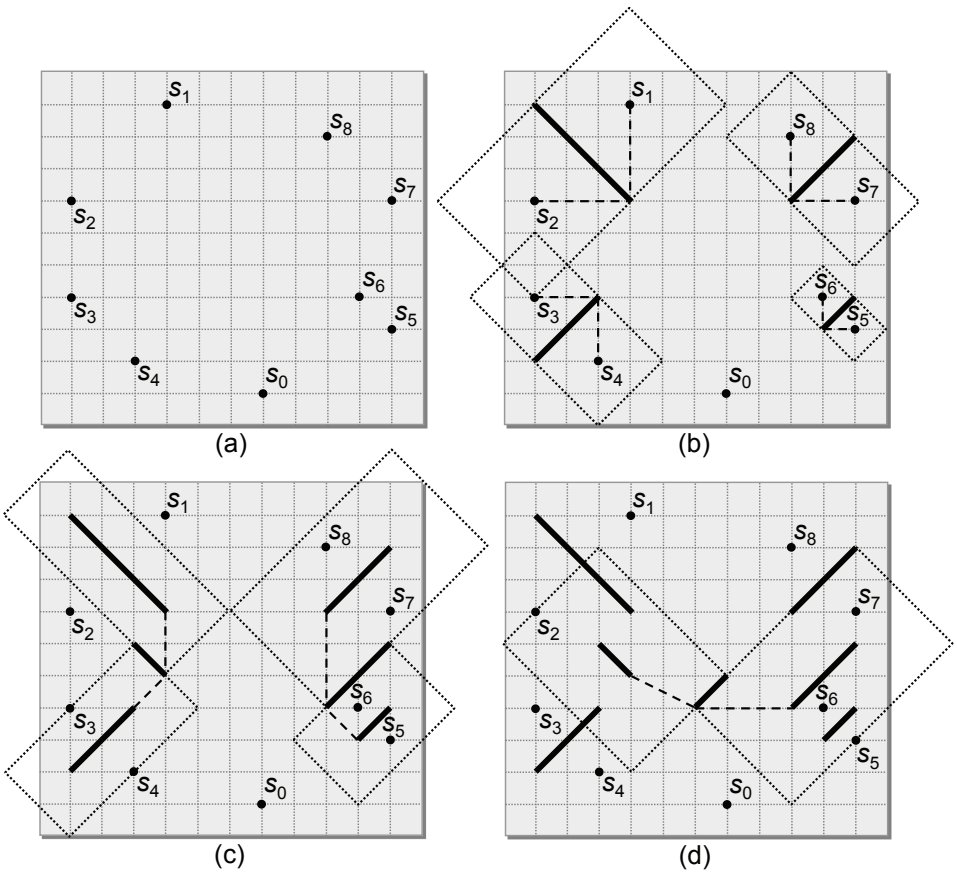


Figure 2.5: Construction of a tree of merging segments (DME bottom-up phase). Solid lines are merging segments, dotted rectangles are the tilted rectangular regions (TRR), and dashed lines are edges between merging segments; s_0 is the clock source, and $s_1 - s_8$ are sinks. (a) The eight sinks and the clock source. (b) Construct merging segments for the eight sinks. (c) Construct merging segments for the segments generated in (a). (d) Construct the root segment, the merge segment that connects to the clock source. Source: [43].

In the DME top-down phase (finding exact locations), exact locations of internal nodes in G are determined, starting with the root. Any point on the root merging segment from the bottom-up phase is consistent with a minimum-cost ZST. Given that the location of a parent node par has already been chosen in the top-down processing, the location of its child node v is determined from two known quantities: (1) $|e_v|$, the edge length from v to its parent par, and (2) $ms(v)$, the locus of placements for v consistent with a minimum-

cost ZST. The location of v, i.e., $pl(v)$, can be determined as illustrated in Figure 2.6 [43]. Thus, the embeddings of all internal nodes of the topology can be determined top-down in linear time (Figure 2.7).

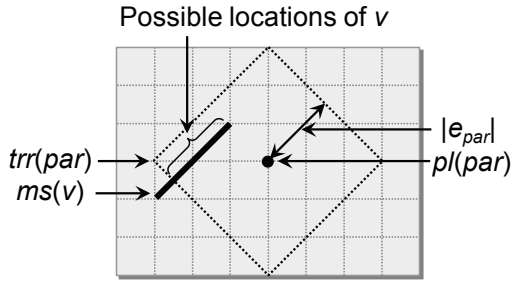


Figure 2.6: Finding the location of child node v given the location of its parent node par. Source: [43].

The Deferred Merge Embedding (DME) algorithm was extended to the bounded-skew tree (BST) problem. BST/DME algorithms [23,38] generalize merging segments to merging regions. When BST/DME algorithms were introduced in the early 1990s, many chip designs included one large central buffer to drive clock signals through the entire chip. Today traditional clock trees cannot satisfy slew constraints in large ICs because the maximal length of unbuffered interconnect decreased significantly due to technology scaling [34]. Furthermore, the Elmore delay model used by published clock-tree optimizations lost accuracy due to resistive shielding and the impact of slew on delay.

BSTs allow one to trade off a small increase in skew for reduced total wirelength. Figure 2.3 shows that BSTs are shorter than ZSTs. However, BSTs are less balanced than ZSTs and Elmore delay used in BST generation is inaccurate, thus the capacitance saved on wires can be lost when compensating for skew with accurate timing analysis. After initial buffer insertion, slow sinks and fast sinks are more clustered in ZSTs. Since our

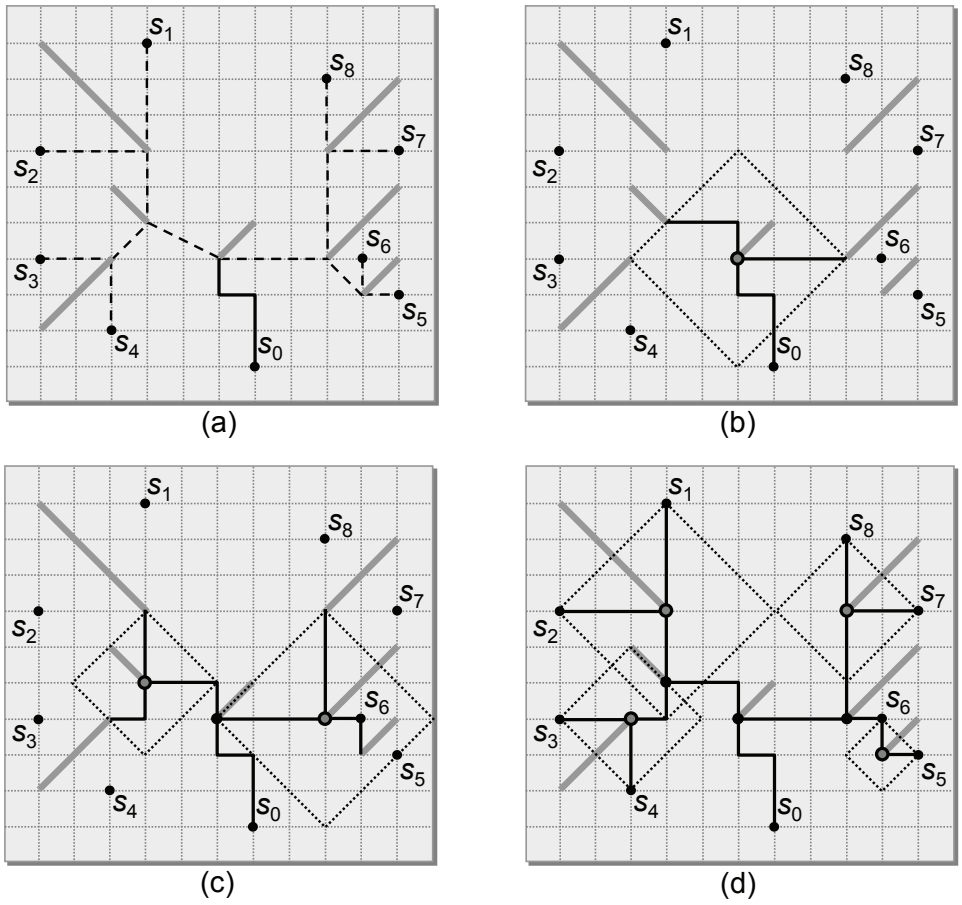
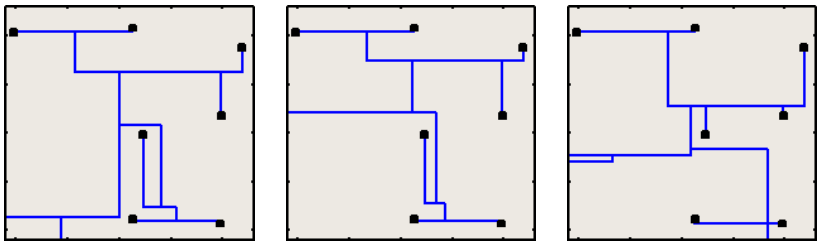


Figure 2.7: Embedding the clock tree during the DME top-down phase. Gray lines indicate merging segments, dotted lines show connections between merging segments, and black lines indicate routing segments. (a) Connecting the clock source to the root merging segment. (b) Connecting the root merging segment to its children merging segments. (c) Connecting those merging segments to its children. (d) Connecting those merging segments to the sinks. Source: [43].

skew optimization techniques exploit these clusters, BSTs need greater resources to reach near zero-skew than ZSTs. Table 2.2 shows the impact of BST skew bounds on final results (CLR is defined in Section 2.1). The skew bounds during BST construction are based on Elmore delay, and the final results are based on SPICE simulations. Based on overwhelming empirical evidence against BSTs, Contango does not use them.

Obstacle-avoiding clock trees. The concept of merging regions in BST/DME was ex-



(a) ZST (b) 3 ps BST (c) 9 ps BST

Figure 2.8: Min-wirelength trees with zero and bounded skew (Elmore delay). Only fragments of actual clock trees are shown.

Skew Bound, ps	Initial CLR, ps	After skew and CLR optimizations CLR, ps	Skew, ps	Cap., fF
0	**52.01**	**13.75**	**1.633**	**77653**
3	57.87	16.33	3.106	74606
6	68.06	18.91	6.004	79955
9	69.64	31.51	18.403	78779

Table 2.2: The impact of skew bounds on *ispd09f22*.

tended to obstacle-avoiding trees in [44], where (i) obstacles were assumed rectangular, (ii) no routing over obstacles was allowed, and (iii) buffering was not considered. The authors noted that obstacle processing slowed down their BST/DME algorithm and hinted at more advanced geometric data structures. Unlike in [44], the ISPD 2009 contest allowed *routing* but not *buffering* over obstacles, with SoCs in mind. ISPD 2009 benchmarks included abutting obstacles that formed monolithic rectilinear obstacles.

Fast buffer insertion. L. van Ginneken introduced an algorithm for buffering RC-trees [30], which minimizes Elmore delay and runs in $O(n^2)$ time, given n possible buffer locations and buffer specification. While not intended for clock trees, it minimizes worst delay rather than skew. The $O(n \log n)$-time variant of van Ginneken's algorithm proposed in [84] is more appropriate for large trees. A key insight into van Ginneken's algorithm and its faster variant makes them applicable to our work — while trying to minimize source

to sink latencies, these algorithms insert almost same number of buffers on every path and therefore result in low skew if the initial tree was already balanced.

Other buffering techniques have been proposed as well, e.g., a linear-time algorithm from [7] that minimizes the number of buffers while bounding capacitive load and slew rate, but does not minimize delay or skew. A dynamic program from [3] inserts a limited number of buffers subject to a maximal skew in buffer counts on source-to-sink paths. At the ISPD 2009 contest, slew constraints were checked by SPICE, but capacitance limits were relatively generous. Our competitors predominantly used greedy bottom-up buffer-insertion algorithms that added each buffer as high in the tree as possible, while satisfying slew constraints. Such technique seek to minimize capacitance as the top priority. However, we chose the (faster variant of) van Ginneken's algorithm, which seeks to minimize worst sink latency. Our rationale was that process variations can be moderated by lowering sink latency and that it is relatively easy to slow down paths that are too fast, but it is harder to speed up slow paths. It is difficult to make a rigorous comparison with slew-based buffering. In particular, some of our competitors at the ISPD 2009 contest relied on it and produced relatively poor results, but others did better. In any case, our overall results compare favorably to the best published results, especially in terms of nominal skew, and we were unable to improve them further by using slew-based buffering.

Several methodologies for clock-tree tuning have recently been developed for the ISPD 2009 clock-network synthesis contest which focused on ASIC and SoC designs. A clock-synthesis methodology for SPICE-accurate skew optimization with tolerance to voltage variations called Contango was proposed in [51]. Dynamic Nearest-Neighbor Algorithm

to generate tree topology and Walk-Segment Breadth First Search for routing and buffering were proposed in [87]. A three-stage CLR-driven CTS flow based on an obstacle-avoiding balanced clock-tree routing algorithm, monotonic buffer insertion, as well as wire-sizing and wire-snaking is proposed in [61]. A Dual-MST geometric matching approach is proposed in [63] for topology construction, along with recursive buffer insertion and a way to handle blockages. SoC methodologies often spend significant effort dealing with hundreds of layout obstacles, while CPU layouts include very few obstacles. However, skew constraints are more difficult in CPU clock synthesis. Because of these differences and due to the incorporation of process variation into the ISPD 2010 contest, most of the above techniques were not adopted by the contestants.

2.4 Interactions between placement and clock-network synthesis

Power consumption is one of the primary optimization objectives for modern IC designs [76]. It includes three basic components: *short-circuit* power, *leakage* power and *net-switching* power [62]. Net-switching power is usually the largest contributor, and clock networks are often responsible for over 30% of total power consumption due to their high capacitance and frequent switching [26, 31, 66, 96]. The quality of clock networks is greatly affected by register placement, but mainstream literature on placement and most commercial EDA tools have largely overlooked this fact by focusing on wirelength of signal nets [48], routability [102] and circuit timing [35]. As far as we know, high-quality register placement cannot be achieved by easy pre- or post-processing of existing techniques. To this end, most appropriate changes to cell locations that reduce the clock network may depend on the current structure of the clock network, which is not

accounted for in existing placement tools. However, over-emphasizing the placement of clock sinks may harm the overall design performance by making signal nets longer.

To address the apparent conflict between clock-net optimization and traditional placement objectives, some researchers proposed techniques and algorithms for better register placement without intrusive interference in traditional placement objectives. Lu [64] proposed several techniques including Manhattan ring-based register guidance, center-of-gravity constraints for registers, pseudo-pins and register-cluster contraction. Cheon [20] proposed power-aware placement that performs both activity-based register clustering and activity-based net weighting to simultaneously reduce the clock and signal net-switching power. In order to reduce the clock network size, Wang [103] proposed dynamic clock-tree building (DCTB), multi level bounding box (MLBB) and multi level attractive force (MLAF), and integrated them into a force-directed placement (FDP) framework [101].

Clock-network optimization after placement can be performed by clustering nearby flip-flops [16, 75] to share inverters (inside flip-flops) and shorten the clock tree. This clustering does not adversely affect signal nets, but is rather limited by the locations of combinational gates. In high-performance CPUs flip-flops are often replaced by single latches, which reduces savings from clock-sink clustering.

PART II

Clock Network Synthesis for SoCs and CPUs

CHAPTER III

Integrated Optimization of SoC Clock Networks

On-chip clock networks are remarkable in their impact on the performance and power of synchronous circuits, in their susceptibility to adverse effects of semiconductor technology scaling, as well as in their strong potential for improvement through better CAD algorithms and tools. Existing literature is rich in ideas and techniques, but performs large-scale optimizations using analytical models that lost accuracy at recent technology nodes, and have rarely been validated by realistic SPICE simulations *on large industry designs*.

This chapter offers a methodology for SPICE-accurate optimization of clock networks, coordinated to satisfy slew constraints and achieve best trade-offs between skew, insertion delay, power, as well as tolerance to variations. Our implementation, called Contango,

is evaluated on 45 nm benchmarks from IBM Research and Texas Instruments with up to 50K sinks. It outperforms all published results in terms of skew and shows superior scalability on the ISPD 2009 benchmarks.

3.1 Introduction

Clock networks were among the first circuits to suffer the impact of process, voltage and temperature variations. Systematic variations can affect paths to different sinks in different ways, making effective skew higher than nominal skew. Intra-die variations may be stronger on some paths than on others, which would further increase effective skew. These challenges have motivated research at the device, circuit and algorithm levels [45]. In general, smaller sink latencies and shorter tree paths decrease exposure to variations. Some researchers tried to increase the tolerance of buffers to CD changes and temperature variation [39], some proposed to tune wires or buffers based on post-silicon measurements [92], and some developed methodologies for inserting cross-links into the trees [36, 37, 50], arguing that such links can decrease the impact of variation on skew. Existing literature tends to (1) rely on closed-form delay models during large-scale optimization, (2) frequently focus on a single optimization technique in analysis and evaluation, (3) neglect the difficulties in modifying highly optimized clock trees. Our work seeks to address these omissions and develops a practical methodology for effective SPICE-accurate optimization, rather than just elegant algorithms with provable abstract properties. With process variation in mind, microprocessor designers combine regular meshes with local or global trees [82]. However, meshes have much higher capacitance and use more power.

This chapter focuses on clock-network synthesis for ASICs and SoCs, where clock frequencies are not as aggressive as in high-performance CPUs, but power is limited, especially for portable applications. In this context, tree topologies remain the most popular choice, but may require accurate tuning and further enhancements. The SoC context introduces another twist — layout obstacles. SoCs include numerous pre-designed blocks (CPUs, RAMs, DSPs, etc) and datapaths. While it may be possible to route wires over such obstacles, buffer insertion is typically not allowed. One can fathom the difficulty of such optimization through comparison to signal-net routing, where obstacle-avoiding Steiner trees currently remain an active area of research [59]. Our contributions include

- A careful analysis of design steps and optimizations for high-performance clock trees, including the range, accuracy, and substitutability of specific techniques
- Notions of *slow-down & speed-up slack* for clock trees
- Tree optimizations driven by accurate delay models
- A simple and robust technique for obstacle avoidance in clock trees subject to slew constraints
- A provably-good sink-polarity correction algorithm
- A methodology for clock-tree optimizations that outperforms the best results at the ISPD 2009 contest *on every benchmark* by 2.15-3.99 times, while reducing skew to 2.2-4.6 ps. On newer Texas Instruments benchmarks with up to 50K sinks, skew remains < 11 ps.

Selecting best parameters for each benchmark can further improve results, at the cost of increased runtime. But global skew $< 20\ ps$ is considered very small for ASICs and SoCs.

In the remainder of this chapter, Section 3.2 describes our analysis of the clock-network synthesis problem and introduces slow-down & speed-up slacks. Major optimization steps are described in Section 3.3, and Section 3.4 presents empirical results. Section 3.5 summarizes this chapter and raises several intriguing research questions.

3.2 Problem analysis and a strategy for solutions

The design of a clock network offers a large amount of freedom in topology selection, spacing and sizing of inverters, as well as the sizing of individual wires. Traditionally, network topology is decided first. Trees offer unparalleled flexibility in optimization because latency from the root to each sink can be tuned individually, while large groups of sinks can be tuned by altering nodes and edges high up in the tree.

Composite buffers can be built by stacking up inverters in parallel and/or in series. Parallel composition decreases driver resistance, but increases input pin capacitance, while leaving the intrinsic delay intact. The spacing of buffers is largely responsible for preventing slew violations and also affects clock skew. It is sensitive to driver resistances, the maximal capacitance (wire and input pins) that can be driven by a given composite buffer, as well as branches in the buffer's fanout, which determine the number of input pins driven. A single wire segment can be split into smaller segments, and each can be sized independently.

3.2.1 Optimization objectives and timing analysis techniques

Accurate clock network design is complicated by the fact that the optimization objectives are not available in closed form and take significant CPU resources to evaluate. Skew optimization requires much higher accuracy than popular Elmore-like delay models. For example, a 5 ps error represents only 1% of 500 ps sink latency, but 50% of 10 ps skew. Closed-form models do not capture resistive shielding in long wires, do not propagate slew with sufficient accuracy, and do not account for slew's impact on delay well. Newer, more sophisticated models are laborious to implement and only available in modern commercial tools. Our strategy is to use simple analytical models at the first steps of the proposed flow — (1) to construct zero-skew clock trees and (2) to perform initial fast buffer insertion, — but drive further optimizations by SPICE runs, Arnoldi approximation, or any other available timing analysis tool/model.

To minimize the number of time-consuming SPICE invocations, we pursued several techniques. Runtime can be significantly reduced using *localization* and *batch-mode evaluation*. During localization, one prunes large portions of the clock tree that do not affect latencies to the sinks impacted by the changes in question [36]. This does not reduce the number of SPICE calls, but rather decreases the complexity of each run. On the other hand, a batch of changes can be evaluated by a single SPICE run, as long as multiple changes do not affect the same path from root to a sink.

Another avenue to streamlined SPICE-driven optimizations is to use mathematical properties of circuit delay, such as monotonicity, convexity, and linearity with respect to some parameters. Monotonicity and convexity support binary search, where an optimal

value is sought on a certain interval. At each step of the search, the middle point of the interval is evaluated by SPICE (e.g., a wire can be sized half-way) and the result determines whether to recur to the left or right half-interval. Linearity enables extrapolation of multiple values based on several SPICE runs.

3.2.2 Nominal skew optimization

An initial buffered clock tree is constructed early in the design flow. Assuming no slew violations, the latency of each sink s (T_s) is known from SPICE simulations (or faster techniques, such as Arnoldi-based delay calculations), at which point minimal and maximal latencies (T_{max} and T_{min}) can be found.[1] Since absolute sink latencies are not as important as skew ($T_{max} - T_{min}$), skew can be improved by either decreasing T_{max} (speeding up the slowest sinks) or increasing T_{min} (slowing down the fastest sinks).

Definition III.1 *Consider a clock tree and its sink s. The* slow-down slack $Slack_s^{slow}$ *(*speed-up slack $Slack_s^{Fast}$*) of s is the amount in ps by which the sink latency can be unilaterally increased (decreased) without increasing clock skew. In other words, $Slack_s^{slow} = T_{max} - T_s$ and $Slack_s^{Fast} = T_s - T_{min}$.*

Slow sinks often cluster together, and so do fast sinks. Hence, clock skew can be improved by modifying a few nodes or edges high in the tree. To find desired delay change, we propagate slack information up the tree as follows.

Let $Sinks_e$ be the set of downstream sinks for edge e.

[1] Separately for rising and falling transitions, for each PVT corner.

Definition III.2 *Consider a clock tree and its edge e. The* slow-down slack $Slack_e^{slow}$ *(*speed-up slack $Slack_e^{Fast}$*) of e is the amount in ps by which the edge delay can be unilaterally increased (decreased) without increasing clock skew.*

Lemma 1 *For any edge e in the tree*

- $Slack_e^{slow} = \min_{s \in Sinks_e} Slack_s^{slow}$
- $Slack_e^{Fast} = \min_{s \in Sinks_e} Slack_s^{Fast}$

Given slacks on n sinks, all edge slacks can be computed in $O(n)$ time.

Lemma 2 *For any edge e and its parent in the tree, $Slack_e^{slow} \geq Slack_{parent(e)}^{slow}$ and $Slack_e^{Fast} \geq Slack_{parent(e)}^{Fast}$.*

The flexibility of a tree edge is limited by each downstream sink. Therefore, for edges close to the root we often have $Slack_e^{slow} = Slack_e^{Fast} = 0$. It is important to note that the validity of slacks-related calculations does not depend on the use of specific delay models or SPICE simulations. When visualizing clock trees, we color their edges with a red-green gradient, indicating low slack with red and high slack with green, as shown in Figure 3.4.

Lemma 2 suggests that, instead of changing the delay of an edge, one can change the delay of its downstream edges by an equal amount, as long as only one delay change is applied on each root-to-sink path. When choosing between tree edges on the same path, we prefer (at early stages of optimization) to tune edges as high in the tree as possible, so as to minimize (i) the amount of change, (ii) the risk of introducing slew violations and (iii) power overhead. However, in a highly optimized tree, we tune bottom-level edges

where we can better predict the impact on skew. The preference for high-level tree edges can be formalized as follows.

Proposition 1 *For each edge e in the tree, define $\Delta_e^{slow} = Slack_e^{slow} - Slack_{parent(e)}^{slow}$. If every edge is slowed down exactly by Δ_e^{slow}, the tree's skew will become zero, and both slow-down and speed-up slacks will become zero.*

Naturally $\Delta_e^{fast} = Slack_e^{fast} - Slack_{parent(e)}^{fast}$, and a mirror statement holds. For a tree edge e, it is possible that $\Delta_e^{fast} > 0$ and $\Delta_e^{slow} > 0$, facilitating conflicting optimizations. If optimizations are not coordinated well, some edges may be sped up and some slowed down, while the overall skew is unchanged. To avoid such conflicts, one can perform rounds of speed-up and rounds of slow-down, separated by SPICE-based analysis and slack update. In practice, it is easier to slow down an edge than to speed it up. Thus, any possible speed-up, e.g., by using stronger buffers, is performed first. Rounds of speed-up and slow-down are more conveniently performed top-down, so that when an edge cannot be tuned by the desired amount, the remainder is passed to its downstream edges.

We found that after nominal skew is sufficiently optimized, both rising and falling transitions can individually limit speed-up and slow-down slacks. We handle the two transitions separately and define edge slacks as the smaller of rise-slack and fall-slack. Furthermore, speed-up and slow-down slacks can be computed for each process corner given (two in the ISPD 2009 contest). In order to improve the multicorner CLR objective, a tree edge can be sped up conservatively by the minimum of its speed-up slacks, and can be slowed down by the minimum of its slow-down slacks.

3.2.3 Clock latency range (CLR) optimization

Our methodology pursues two objective functions —- nominal skew and the ISPD 2009 CNS contest metric, CLR, introduced in Section 2.1. Due to significant correlation between CLR and nominal skew, some of the optimizations in our flow target skew optimization, some target CLR, and some address both (see Table 3.3). In practice this approach achieves a good trade-off between the two optimization objectives, and is representative of multi-objective optimization required in many practical settings. Recall that the CLR calculation is based on the sink latencies at two different supply voltage settings. There are mainly two strategies to reduce CLR. First, reducing skew directly contributes to reducing CLR until skew becomes very small (e.g. less than 5 ps). Let sink L be the sink with the least sink latency @1.2V ($T_L^{1.2V}$) and sink G be the sink with the greatest sink latency @1.0V ($T_G^{1.0V}$). Then $CLR = T_G^{1.0V} - T_L^{1.2V}$. When we consider the latency of sink G @1.2V ($T_G^{1.2V}$), then $CLR = (T_G^{1.0V} - T_G^{1.2V}) + (T_G^{1.2V} - T_L^{1.2V})$. We call ($T_G^{1.0V}$-$T_G^{1.2V}$) the variational part of CLR and ($T_G^{1.2V}$-$T_L^{1.2V}$) the skew part of CLR. The skew part of CLR can be reduced by skew optimization techniques. Since the corner sinks of skew are not always same to the corner sinks of CLR (sink L and G), CLR needs to be measured after any skew optimization to check CLR improvement. The second strategy for CLR optimization targets the variational component of CLR. The detailed descriptions of optimizations for the skew and variational part of CLR are discussed in Section 3.3.

3.2.4 Coordinating multiple optimizations

We found that different clock-tree optimizations exhibit different *strength/range* and different *accuracy* (see Tables 3.3 and 3.6). For example, buffers can be inserted to in-

crease delay with the purpose to decrease nominal skew. This optimization offers a great range (significant strength) because a buffer's intrinsic delay can be significant. However its accuracy is low because buffer delay cannot be accurately controlled. Our strategy in coordinating clock-tree optimizations is to start with optimizations that offer the greatest range, and then transition to optimizations with greater accuracy. Each step should decrease the main optimization objective sufficiently to be within the range of the next optimization. For example, in the ISPD 2009 contest, top-down wiresizing can decrease nominal skew from hundreds of ps to 10 - 20 ps. This is sufficient for top-down wiresnaking to take over and reduce nominal skew to 5 ps where more accurate techniques can be used. Here we observed that wiresnaking also exhibits a significant range of optimization, but we sequenced it *after* wiresizing because it offer a greater accuracy and because it increases capacitance, whereas wiresizing decreases it.

3.3 Proposed SoC clock-synthesis methodology

Our proposed clock-network synthesis methodology and its major algorithmic steps are shown in Figure 3.1. Contango first builds an initial tree using a ZST/DME algorithm [23] and alters it to avoid obstacles. It then uses an $O(n \log n)$-time variant of van Ginneken's buffer insertion algorithm [84] to ensure small insertion delay and to satisfy slew constraints. A series of novel clock-tree optimizations are applied next.

3.3.1 Obstacle-avoiding clock trees

As we pointed out in Section 2.3, obstacle-avoiding clock trees can be built by repairing obstacle violations in ZSTs. This approach is attractive when large obstacles abut the

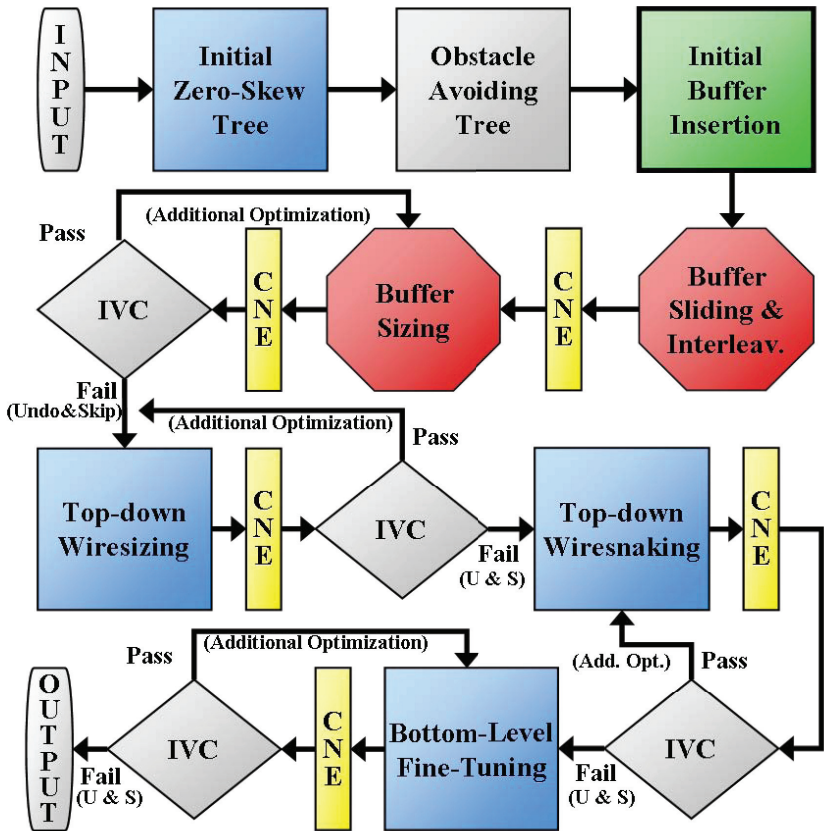


Figure 3.1: Key steps of the Contango methodology. Blue boxes represent *skew reduction* techniques, red octilinear shapes show *CLR reductions*, and the green box with thick border reduces both objectives. An Improvement- & Violation-Checking (IVC) step follows each Clock-Network Evaluation (CNE) using circuit simulation tools, e.g., SPICE. "Fail" indicates no improvement or having slew violations, leading to a transition to the next optimization.

chip's periphery because ZSTs naturally avoid areas without clock sinks. This approach is also attractive when obstacles are small or thin enough that a buffer inserted immediately before the obstacle can drive the wire over the obstacle, so that no rerouting is necessary. A third convenient case occurs when a wire can be rerouted around the obstacle without an increase in length. Most obstacles are rectangular in shape, but such rectangles may abut, creating rectilinear-shaped obstacles. When two obstacles abut, we cannot place a buffer between them, and therefore handle them as one compound obstacle. Contango detours

wires using the following algorithm, illustrated in Figure 3.2 for a composite obstacles.

Step 1. Identify all wires that intersect obstacles. For each point-to-point connection, perform *shortest-path maze routing* around the obstacles. For subtrees that cross an obstacle, find L-shaped segments that link points inside and outside the obstacle. For each L-shape, choose one of the two possible configurations that minimizes overlap with the obstacle.

Step 2. When a wire crosses an obstacle, Contango captures an entire subtree enclosed by the obstacle (see Figure 3.2). The total capacitance of the subtree is then measured and compared to the capacitance that can be driven by the driving buffer without risking slew violations. Sub-trees that can be driven by the driving buffer do not require detours.

Step 3. For obstacles crossed by a subtree that cannot be safely driven by the driving buffer, Contango establishes a detour along the contour of the obstacle as follows. First, the entire contour is considered a detour. Then, to ensure that the clock network remains a tree, one segment is removed between tree sinks adjacent along the contour. If we were to minimize total capacitance, we would remove the longest segment of the contour between two adjacent tree sinks. However, *we minimize the longest detoured source-to-sink path*, and therefore *remove the segment furthest from the tree source* (counting distances along the contour). In other words, we first find the sink most distant from the source along the contour, and include in the detour the entire shortest path to the source. The other segment incident to the sink is removed, but the shortest path from its other end to the source is included (see Figure 3.2).

Modern SoC layouts are littered with obstacles, which upset regular structures such as meshes and H-trees. In the ISPD 2009 contest, such layouts required numerous de-

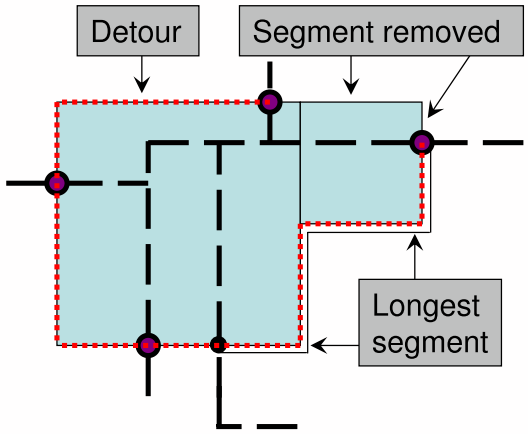


Figure 3.2: An illustration of our detouring algorithm. Small solid circle indicates the source of detour, larger circles indicate sinks. The detour is shown with red dotted lines.

tours. Detouring may significantly increase skew, but the subsequent skew optimization techniques can compensate for that.

3.3.2 Composite inverter/buffer analysis

Most technology libraries support dedicated clock buffers or inverters that are larger and more reliable than those for signal nets. Industry designs usually offer at least six different sizes. Parallel composition of buffers increases driver strength, helping with slew constraints and improving robustness to variations. Yet, buffer sizes must be moderated to satisfy total power limits. For a given buffer library, we consider many possible composite buffers. Using dynamic programming, we select several non-dominated configurations that can be further evaluated during buffer insertion. Algorithmic details are omitted here because the ISPD 2009 contest used only two inverter types — $large$ and $small$. Table 3.1 shows that eight parallel $small$ inverters exhibit smaller output resistance than one $large$ inverter, and smaller input/output capacitance. Hence Contango used $8\times$ $small$ inverters

INVERTER TYPE	INPUT Cap., fF	OUTPUT Cap., fF	OUTPUT Res., Ω
1X Large	35	80	61.2
1X Small	4.2	6.1	440
2X Small	8.4	12.2	220
4X Small	16.8	24.4	110
8X Small	**33.6**	**48.8**	**55**

Table 3.1: Inverter analysis for ISPD 2009 CNS benchmarks.

instead of $large$ inverters, in batches of $16\times$, $24\times$, etc. This benchmark-independent optimization, along with buffer sizing, plays an important role in our methodology.

3.3.3 Initial buffer insertion with sizing

Given a clock tree with buffers, it is easy to increase the latency of a given sink, but it is difficult to speed up a sink. Therefore, our strategy is to first make sinks as fast as possible, and then reduce skew with wiresnaking and wiresizing. When buffers are inserted into an Elmore-balanced tree, source-to-sink paths contain practically the same numbers of buffers (can be off by one in some cases).

We adapted the $O(n \log n)$-time variant of van Ginneken's algorithm from [84]. Due to its speed, it can be launched with different inverter configurations, effectively performing simultaneous optimization across multiple parameters. Our experiments indicate that driver strength is a major factor in moderating the impact of supply-voltage variations. Therefore, to reduce the variational part of CLR, Contango performs fast buffer insertion with different composite buffers until it finds the best-performing solution with strongest composite buffers within 90% of the power limit. Slew-constraint violations are not a concern at this point since minimizing delay involves avoiding high slew-rate (recall that there is positive correlation between delay and slew-rate). The experiments on various

clock trees with initial buffer insertion suggest that even the worst slew-rate is well under 60% of the slew limit. We reserve $\gamma = 10\%$ of power budget to facilitate more accurate optimizations.

The $O(n \log n)$ variant of van Ginneken's algorithm [84] used in our work assumes that all available clock buffers preserve polarity. However, when polarity-changing inverters are used, as in the ISPD 2009 contest, it will typically produce trees with incorrect sink polarity (inverted sinks). While the algorithm can be extended to account for sink polarity, we found this unnecessary. Even a simple patch — placing additional inverters at each of $n_\times$ inverted sinks — works reasonably well, because the skew introduced by new inverters can be fixed by downstream optimizations. This technique inserts inverters at half the sinks ($n/2$) on average. To reduce the added capacitance in cases when $n_\times > n/2$, Contango inserts one inverter at the top of the tree, leaving only $n_\sharp = (n - n_\times) < n/2$ sinks with wrong polarity. The average number of inserted inverters would now be $(n + 2)/4$. Instead, Contango traverses the tree bottom-up and marks each node (i) whose all sinks have equal polarity, but (ii) whose parent does not satisfy (i). An inverter is inserted at each marked node with downstream sinks of incorrect polarity. As a result, the number of added inverters is significantly reduced, as shown in Table 3.2. The skew induced by new inverters is not significant and fixed by the skew optimization algorithms later.

Proposition 2 *The above algorithm runs in $O(n)$ time, fixes all inverted sinks and minimizes the number of added inverters, subject to ≤ 1 inverter on every root-to-sink path.*

	f11	f12	f21	f22	f31	f32	fnb1
Inverted sinks	77	71	46	57	140	47	153
Added inverters	9	7	8	9	16	13	2

Table 3.2: Inverted sinks in ISPD 2009 benchmarks (after buffer insertion) vs. polarity-correcting inverters.

3.3.4 Buffer sliding and interleaving

We now discuss targeted improvement of robustness to variations in device performance. The iterative buffer sizing introduced in Section 3.3.5 is primarily used to reduce the variational component of CLR, while buffer sliding and interleaving are applied as preliminary steps. Extensive experiments suggest that the impact of variations on skew is best reduced by (i) decreasing sink latency (insertion delay), and (ii) using the strongest possible buffers. Since our initial buffer insertion algorithm focuses on the former metric with the latter metric as a secondary objective, it is possible to further improve the variational component of CLR by emphasizing the latter metric. Therefore, based on the results of initial buffer insertion, Contango attempts to size buffers up.

Sizing up a single inverter increases its input pin capacitance and can lead to slew violations. To prevent such violations, it is often possible to slide the inverter up the tree to reduce upstream wire capacitance and interleave an inverter when two inverters move too far apart after sliding. The increase in downstream wire capacitance is balanced with the increase in the inverter's driving strength. Sizing a single inverter may increase the skew and require further correction. Therefore, we focused on the top-most levels of the tree, whose impact on skew is relatively small. Given a clock source at the chip boundary, DME algorithms generate a long wire leading to the center of the chip, and the tree branches out from the center. This long wire — the *tree trunk* — is later populated with a chain of

inverters, which can be up- or down-sized without significant impact on skew because this equally affects all sinks. However, since roughly 1/3 to 1/2 of sink latency is due to the tree trunk, it accounts for a large fraction of variational impact on latency.

The trunk's variational impact is different for voltage and process variations, and this must be accounted for during optimizations. Stronger buffers in the trunk reduce the sensitivity of latency to *supply voltage* (e.g., in the case of different power modes), and help optimizing the CLR objective from the ISPD 2009 contest. However, process variations in the trunk do not affect skew. In the ISPD 2010 contest, *process* variations were included in the skew constraint, while the primary objective was to *minimize total capacitance*. Therefore, one of successful strategies to *weaken* the buffers in the tree trunk and avail the capacitance saved to other optimizations.

3.3.5 Iterative buffer sizing

After sliding and interleaving top-level buffers, we invoke iterative buffer sizing. First, this algorithm sizes up buffers in the tree trunk. At the i-th iteration of buffer sizing, Contango sizes up the composite inverters by at most $p_i = 100/(i+3)\%$. The iterations continue until results improve without slew violation. Buffer sizing in tree branches incurs a greater capacitance penalty. To compensate, Contango borrows capacitance by downsizing bottom-level buffers.

However, sizing up buffers after the trunk often makes the tree unbalanced in terms of skew and results in greater load for the skew optimization algorithms. For better performance of skew optimizations, typically 4 or 5 levels after the first branch are sized up by capacitance borrowing buffer sizing algorithm.

3.3.6 Iterative top-down wiresizing

Before skew optimization, Contango computes slow-down slacks at every edge as described in Section 3.2, and the Δ_e^{slow} parameters. This suggests the amount by which a given tree edge can be slowed down before skew would be negatively affected. Since fast sinks often cluster together, skew can be lowered by slowing down either many bottom-level wires or few wires higher in the tree. Our top-down algorithm pursues the latter, seeking to minimize tree modifications.

We build an *ad hoc* linear model based on the impact of downsizing a unit-length (l_{ws}) wire segment. Contango chooses several independent wire segments with same length (l_{ws}) in the middle of the tree and downsizes them to observe the impact on latencies of downstream sinks, ensuring that every sink is affected by only one downsized wire. This requires a single SPICE run and produces a single parameter T_{ws} — maximal latency increase by downsizing a unit-length (l_{ws}) wire segment. When downsizing a wire, the scaling factor k is calculated based on $Slack_e$ divided by T_{ws} and $k \times l_{ws}$ of the wire is down-sized. When k is small, the latency increases almost linearly since the down-sized length is much smaller than the length of the wire. Therefore we can estimate that the maximum latency increase is equal to or less than $k \times T_{ws}$. To utilize this linearity, we limit k by k_{max}. k_{max} is experimentally determined by observing the threshold at which the linearity breaks significantly. Also, the scaling factor k can be limited by slew constraints. Wiresizing typically increases slew rate because of increase in resistance. Even though $k < k_{max}$ holds, Contango does not allow any downsizing on a wire whose downstream node has slew rate above 80% of the slew limit.

Algorithm 1 IterativeWireSizing

```
T_ws = TwsEstimation();
repeat
  SaveSolution(); ComputeWireSlacks();
  Q = {root}; RSlack = {0}; i = 0;
  while i < size(Q) do
    if (Slack[Q_i] − RSlack_i > T_ws) then
      k = (Slack[Q_i] − RSlack_i)/T_ws;
      DownSize(Wire[Q_i],k); RSlack_i+ = kT_ws;
    end if
    for j = 1 to Size(Child[Q_i]) do
      Q.push(Child[Q_i][j]); RSlack.push(RSlack_i);
    end for
    ++i;
  end while
  SpiceSimulation();
until (no improvement || slew violation)
```

Since we selected T_{ws} as the maximal latency increase from the SPICE simulation, the actual increase (calculated by SPICE) is smaller — our modifications are intentionally conservative to avoid excessive increase of latency, which increases the maximal latency of the tree and consequently causes increase of slack for the entire tree. After running SPICE, collecting sink latencies and recomputing slow-down slacks, Contango repeats top-down wiresizing to reduce skew based on current data. This process is performed iteratively until the objective function (CLR or nominal skew) stops improving. Iterative wiresizing is detailed in Algorithm 1.

3.3.7 Iterative top-down wiresnaking

Wiresizing can reduce large skew by applying small changes, which is appropriate after the initial tree construction. An experienced clock-network designer suggested to us that a small amount of wire-snaking is often used to improve clock skew, as long as added capacitance does not significantly affect power. Wiresnaking alters a given route so as to

increase its length and can be applied on fast paths.

We develop an accurate top-down wiresnaking process, which we invoke *after* top-down wiresizing. This step uses the same slow-down slack computation we described earlier. A SPICE simulation is performed (other accurate delay model can be used) to measure T_{wn}, the worst-case delay of wiresnaking with unit length l_{wn}. l_{wn} affects the accuracy of the wiresnaking algorithm; smaller l_{wn} offers greater accuracy but typically leads to more SPICE runs since skew reduction in each round of top-down wiresnaking is smaller. l_{wn} was set based on empirical analysis of the 45 nm technology used at the ISPD contest before contest benchmarks became available. The applicability of wiresnaking depends on the VLSI context. If the clock tree is competing for routing resources with signal nets, then every effort should be taken to reduce the utilization of routing resources. In particular, wiresnaking cannot be used in areas of routing congestion (also, clock trees should avoid such areas to minimize crosstalk noise). On the other hand, some ICs include abundant routing resources. This is the case for pad-limited designs and designs whose area is determined by large IP blocks. The number of available metal layers also plays a major role in the design of clock trees, and can vary dramatically between different designs, ranging from 6 to 12 layers as of 2010. In some high-performance designs, clock networks are given a dedicated metal layer, which makes wiresnaking much more attractive.

One of the top-three teams at the ISPD 2009 clock-tree routing contest (NTU [87]) used *dangling wires* instead of wiresnaking. Rather than elongate a route, this strategy adds a dead-end branch. The goal is to increase wire capacitance, and therefore increase

	ISPD09F11		ISPD09F12		ISPD09F21		ISPD09F22		ISPD09F31		ISPD09F32		ISPD09FNB1	
	CLR	Skew	CLR	Skew	CLR	Skew	CLR	Skew	CLR	Skew	CLR	Skew	CLR	Skew
INITIAL	56.2	30.6	75.8	49.0	89.3	59.2	52.0	31.6	152	117	122	88.2	31.9	21.2
TBSZ	55.6	46.8	80.0	66.2	89.5	76.3	43.2	33.7	140	129	111	98.3	31.5	21.1
TWSZ	23.4	15.1	19.7	8.13	26.0	12.3	16.4	6.93	43.1	32.2	27.2	14.8	30.8	20.4
TWSN	13.8	2.93	16.2	3.38	17.6	2.83	12.6	**1.99**	12.8	3.91	17.9	4.59	13.9	**3.15**
BWSN	**13.4**	**2.87**	**15.3**	**2.61**	**17.4**	**2.74**	**12.4**	2.23	**12.8**	**3.91**	**17.9**	**4.59**	**13.4**	3.5

Table 3.3: Progress achieved by individual steps of Contango on ISPD 2009 benchmarks: the first letter in each acronym indicates top-down (T) or bottom-level (B) optimization, second letter differentiates wires (W) from buffers (B), while "Sz" stands for "sizing" and "Sn" stands for "snaking". Gray highlights indicate whether skew or CLR was the primary optimization objective.

the delay. In comparing dangling wires to wire-snaking, we note that the former does not alter the resistance that affects propagation delay. Therefore, to achieve a particular slowdown, a much longer wire-branch is needed. On the positive side, the dependence of delay increase on branch length is linear, and this may allow for more accurate tuning. In other words, this technique offers a potentially *greater accuracy*, but *smaller range* because the range of such optimizations is limited by the capacitance budget. Therefore, if dangling wires are found useful, they should be used at a later stage in the optimization flow.

3.3.8 Bottom-level fine-tuning & limits to further optimization

After two top-down skew reduction phases, skew becomes small enough to perform bottom level optimizations. Bottom-level wiresnaking optimize the wires directly connected to sinks. This technique is more accurate than the top-down optimizations since each sink is tuned individually. Contango performs SPICE-driven bottom-level wiresnaking until the results stop improving. Typically the gain of bottom-level tuning is under 2 ps, but can be a significant fraction of remaining skew. We found that with skew < 5 ps, the corner sinks of rising transition and falling transition are often different. This *rise-fall divergence* makes further improvements to the clock tree very difficult. Indeed, reducing

rising skew by slowing down a *fast sink for rising transition* may increase *falling skew* due to excessive slowdown of a *slow sink for falling transition*. In the Contango flow, the average skew after bottom-level tuning is 3.21 ps on ISPD 2009 CNS contest benchmarks.

Table 3.3 shows the improvement of CLR and skew by each optimization algorithm. Note that after iterative buffer sizing (TBSz), skew is increased but CLR does not change much. This implies that TBSz reduced the variational part of CLR significantly. The increased skew is reduced below 5 ps after our skew optimizations.

3.4 Empirical validation: Contango 1.0

To validate our proposed techniques, we first present results on ISPD 2009 benchmarks according to the contest protocol, then discuss the significance of specific optimizations used by Contango, and then evaluate the scalability of our C++ implementation on larger benchmarks from our industry colleagues. We measured runtimes on a 2.4 GHz Intel QuadCore CPU running Linux, similar to CPUs used at the ISPD contest.

ISPD 2009 benchmarks include seven 45 nm chips up to 17 mm $\times$ 17 mm in size, with up to 330 selected clock sinks [94]. Table 3.4 compares results of our software Contango to the top three teams of the ISPD 2009 clock-network synthesis contest. On average, Contango reduces CLR by **2.15**$\times$, **3.99**$\times$ and **2.35**$\times$ versus contest results by NTU, NCTU and U. of Michigan respectively, excluding failures of NTU and NCTU on benchmarks with many obstacles. All results are within the capacitance limits, but Contango nearly exhausts the limits as a part of its strategy. On ISPD 2009 benchmarks, maximum sink latency averages 1120 ps, while the average number of composite-buffer locations is 223. A clock tree built by Contango is shown in Figure 3.4.

Benchmark	CONTANGO(THIS WORK) 9/10/2009			NTU 3/30/2009			NCTU 3/30/2009			U. OF MICHIGAN 3/30/2009		
	CLR	Cap.	◷	CLR	Cap.	◷	CLR	Cap.	◷	CLR	Cap.	◷
ispd09f11	**13.36**	99.61	6488	26.71	85.53	14764	**22.31**	89.90	23358	32.29	73.86	3892
ispd09f12	**15.27**	99.99	6564	25.73	84.72	13934	**22.18**	87.86	14992	32.17	73.45	3944
ispd09f21	**17.40**	96.74	6673	30.54	80.79	14978	**19.61**	86.65	26420	34.31	74.30	4587
ispd09f22	**12.36**	97.43	3618	24.51	81.82	7189	**16.38**	85.01	9432	30.45	70.01	2005
ispd09f31	**12.81**	98.29	21379	**45.07**	73.49	40088	212.0	92.38	1.29	51.34	81.53	17333
ispd09f32	**17.92**	99.24	12895	**36.90**	80.14	3566	fail	-	-	40.32	77.39	10599
ispd09fnb1	**13.40**	78.38	778	fail	-	-	fail	-	-	**19.84**	63.10	477
Average	**14.65**	**95.66**	**8342**	**31.57**	**81.08**	**15753**	**58.49**	**88.36**	**14841**	**34.39**	**73.38**	**6120**
Relative	1.0	1.0	1.0	2.15	0.85	1.89	3.99	0.92	1.78	2.35	0.77	0.73

Table 3.4: Results on the ISPD 2009 Contest benchmark suite. CLR is reported in ps, capacitance in % of the limit specified in benchmarks, and CPU time in s. Best results from the ISPD 2009 contest and best results overall are shown in bold. Runtime is dominated by SPICE runs. It was not used for scoring at the ISPD 2009 contest and can be improved by using FastSPICE, Arnoldi approximation.

More recent results for ISPD 2009 benchmarks from ASPDAC'10 [61, 63, 87] are summarized in Table 3.5. The results in Table 3.5 show that Contango outperforms NTU and NCTU by skew and CLR. HKPU [63] claims a 20% advantage in CLR, but more than doubles nominal skew. Another interesting aspect of the HKPU work is that they rely on SPICE very little in their optimizations and instead use the Elmore delay model, which explains their low runtimes. The algorithms in [63] focus entirely on the optimization of nominal skew, which does not explain the results — high nominal skew and low CLR. As the authors of [63] have kindly provided their clock trees on our request, we observed that those trees use very large buffers at the top levels of the tree (including but not limited to the trunk) and small buffers toward the sinks. This strategy minimizes the impact of *supply voltage* variations, but makes it more difficult to optimize nominal skew given a limited capacitance budget.

Significance of individual optimizations. Several optimizations we have implemented were superseded by more powerful techniques. For example, *skew reduction by buffer insertion* was unnecessary and undermined the robustness to variations. However, it can

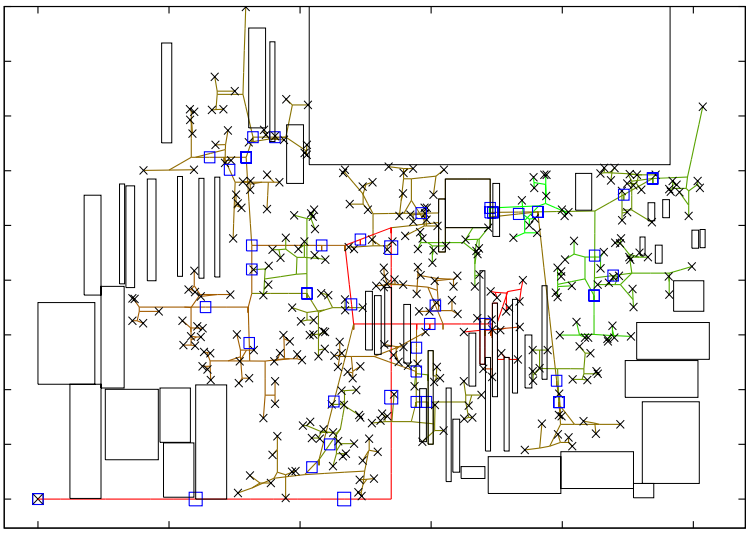


Figure 3.3: The clock tree produced by Contango on $ispd09fnb1$. Sinks are indicated by crosses, buffers are indicated by blue rectangles. L-shapes are drawn as "diagonal wires" to reduce clutter. Wires are colored by a red-green gradient to reflect slow-down slacks, as described in Section 3.2.2. The impact of wiresnaking is too small to be visible.

Benchmark	CONTANGO (this work)			NTU [87]			NCTU [61]			HKPU [63]		
	CLR	Skew	◷	CLR	Skew	◷	CLR	Skew	◷	CLR	Skew	◷
ispd09f11	13.36	2.867	6488	19.71	4.478	4639	18.77	7.12	30787	12.2	—	180
ispd09f12	15.27	2.611	6564	17.46	4.088	4231	15.5	3.06	27622	10.9	—	213
ispd09f21	17.40	2.738	6673	19.92	3.868	4629	17.04	3.02	33056	12.1	—	210
ispd09f22	12.36	2.227	3618	16.47	3.671	3937	16.25	4.11	19136	9.9	—	113
ispd09f31	12.81	3.91	21379	31.13	4.762	11112	22.63	7.58	66588	13.4	—	777
ispd09f32	17.92	4.594	12895	23.04	4.234	7293	20.59	5.52	49907	11.5	—	420
ispd09fnb1	13.40	3.5	778	15.73	6.798	3719	14.32	3.77	7643	13.8	—	82
Average	**14.65**	**3.207**	**8342**	**20.49**	**4.56**	**5651**	**17.87**	**4.88**	**33534**	**11.97**	**7.72**	**285**

Table 3.5: Results from ASPDAC'10 clock routing papers on the ISPD 2009 Contest benchmark suite [61, 63, 87]. Runtimes may be from different workstations. CLR and skew are reported in ps and CPU time in s. Only average skew was published for HKPU [63].

be used as a last resort when detours around obstacles introduce extremely high skew. Our wiresizing can be refined, but probably not beyond the accuracy of subsequent wiresnaking. In practice, wiresnaking is very limited, so as to preserve the routability of signal wires (unless clock wiring is given a dedicated metal layer). Dangling wires, used by NTU instead of wire snaking, would be even less acceptable.

ispd09f12	**Full flow**	**w/o TWSz**	**w/o TWSn**	**w/o BWSn**
TWSz	-58.11	-	-58.11	-58.11
TWSn	-4.740	-33.51	-	-4.740
BWSn	-0.773	0	-2.494	-
Skew	**2.611**	**14.92**	**5.633**	**3.384**

Table 3.6: The 'Full flow' column shows *skew change* at each step in the Contango flow, and the final skew in ps. Acronyms are decoded in the caption of Table 3.3. Subsequent columns show the impact of removing one optimization. These results illustrate the *range* of each optimization and its *impact on final results*.

To further study the relative significance of optimizations in Contango, we show in Table 3.6 the impact of removing each skew optimization step from the flow. It can be seen that each step is necessary to achieve competitive results. Removing top-down wire-sizing effects the greatest impact because this optimization offers the greatest range, and subsequent optimizations cannot fully compensate for its omission.

Scalability studies. The ISPD 2009 contest was limited to unrealistically small numbers of sinks due to limitations of the open-source ngSPICE software [71] it relied upon. To evaluate the scalability of our optimizations, we replaced ngSPICE with industry-standard HSPICE software [93].[2] Working with a recent Texas Instruments chip sized $4.2mm \times 3.0mm$, we identified locations of 135K sinks and randomly sampled them to create a family of benchmarks. For this experiment, our algorithm used groups of large inverters instead of groups of 8 parallel small inverters, improving runtime eightfold at the cost of increasing CLR and skew by 1 - 2 ps and increasing capacitance by 15%. It produced highly-optimized clock trees with up to 50K sinks. Table 3.7 shows that total capacitance scales linearly with the number of sinks, and skew remains in single ps. The number of HSPICE runs grows very slowly, but HSPICE remains the bottleneck.

[2]The numbers produced by ngSPICE and HSPICE were fairly close, with the main difference being runtime and scalability.

# sinks	CLR, ps	Skew, ps	Latency, ps	Cap., pF	◷, min
200	13.47	2.124	506.8	52.21	2.2 (21)
500	14.84	2.174	528.0	99.53	6.28 (20)
1K	17.53	3.138	543.1	162.3	12.5 (20)
2K	16.56	3.136	543.9	276.1	19.3 (15)
5K	23.20	3.853	538.5	591.1	99.6 (22)
10K	25.54	5.562	538.0	1130	352.8 (23)
20K	32.47	10.46	546.8	2243	1867 (35)
50K	31.52	8.774	545.1	5243	16027 (45)

Table 3.7: Scalability on Texas Instruments benchmarks. The "Latency" column represents maximum 1.2V latencies. SPICE runs are counted in parenthesis.

3.5 Summary

Existing literature on clock networks offers several elegant algorithms, but does not describe end-to-end solutions to clock-network synthesis that can handle modern interconnect. Our work makes several contributions to this end. $First$, we develop specialized optimization algorithms necessary to bridge the gaps between well-known point-optimizations. Our emphasis is on robust techniques, that do not require benchmark-specific tuning and are amenable to embedding into design flows. $Second$, we develop an EDA methodology for integrating clock-network optimization steps. $Third$, we describe a robust software implementation, called Contango, that outperforms the best results from the ISPD 2009 contest [94] by a factor of two.[3] $Fourth$, we scale our implementation to large industrial clock networks. Based on their strong empirical results, our techniques may improve timing and power of future ASICs and SoCs [33]. In CPU designs, our trees can be integrated with meshes [82]. Here, better trees may facilitate smaller meshes and reduce power consumption, which can be traded off for higher performance or longer battery life in portable applications. Optimization techniques presented in this chapter serve as a foundation for research reported in the remaining chapters of this dissertation.

[3]The use of two wire sizes, two inverter types, and two process corners at the ISPD 2009 contest is not a limitation of our algorithms and methodology. Likewise, any accurate delay evaluator can be used, including FastSpice, Arnoldi approximations, etc.

CHAPTER IV

Low-power Clock Trees for CPUs

Clock networks contribute a significant fraction of dynamic power and can be a limiting factor in high-performance CPUs and SoCs. The need for multi-objective optimization over a large parameter space and the increasing impact of process variation make clock network synthesis particularly challenging. In this chapter, we develop new modeling techniques and algorithms, as well as a methodology, for clock power optimization subject to tight skew constraints in the presence of process variations. Key contributions include a new time-budgeting step for clock-tree tuning, accurate optimizations that satisfy budgets, modeling and optimization of variational skew. Our implementation, Contango 2.0, outperforms the winners of the ISPD 2010 clock-network synthesis contest on 45 nm benchmarks from Intel and IBM. To support emerging SPICE-accurate circuit optimizations, we propose Chop-SPICE, a divide-and-conquer technique for scaling SPICE simulations to large, buffered RC trees, that trades off precision for speed.

4.1 Challenges addressed

Recent developments in embedded CPU design stress the need for low-power clock trees, yet also impose stringent skew limits, especially in the presence of process, voltage

and temperature (PVT) variation for sub-45 nm CMOS technology. Previous clock-tree methodologies rely on symmetric and regular tree topologies, such as H-trees and fish-bones [6, chapter 43], which do not require sophisticated design algorithms (see Section 1.1). However, these topologies experience difficulties with layout obstacles, non-uniform sink distributions, and varied sink capacitances. Fully-automated clock-tree synthesis supported by commercial EDA tools offers clear advantages in terms of capacitance, but may not be able to ensure sufficiently low skew for use in a 2 GHz CPU. For example, the authors of [65] report clock trees generated by Cadence tools with skew that is orders of magnitude higher than the single-ps skew provided by clock meshes.

4.1.1 Research questions

In this chapter, we pursue the following research questions.

- How far can the skew of a high-performance clock tree be optimized?
- How can one minimize the impact of PVT variations in a clock tree?
- Given a single-picosecond skew requirement, how competitive are clock trees with clock meshes?

Our approach to answering these questions is inspired by the ISPD 2010 clock-network synthesis contest, which used several 2 GHz CPU benchmarks from IBM and Intel to compare tools submitted by 10 teams across the world (downselected from 20 initial registrants). To evaluate the quality of the clock networks, difficult slew and skew constraints were checked against 45 nm Monte-Carlo SPICE simulations that modeled PVT variations. Clock networks that cleared all constraints were compared by their total capacitance

— a proxy for dynamic power. In this context, we developed a suite of algorithms for the design and thorough optimization of clock trees. The results of the ISPD 2010 contest offer a rare opportunity to compare multiple strategies for clock-network synthesis — the third-place team used symmetric trees [85], the second-place team used clock meshes, and our team won the contest by optimizing clock trees built by the DME algorithm [12, 28].

Our comprehensive methodology for clock-network synthesis integrates the following specific innovations.

- The notion of *local-skew slack* for clock trees.
- A tabular technique to estimate the impact of variations on skew between two sinks.
- A path-based technique to enhance the robustness of a clock tree to PVT variations.
- A time-budgeting algorithm for clock-tree tuning that distributes delay targets to individual edges of the tree so as to improve skew with minimal power resources. This algorithm can be used in the context of PVT variations and is not specific to our methodology.
- Fine tuning of optimized clock trees by gentle wire snaking, sufficiently accurate to satisfy delay budgets.

Our empirical results are compared to those of the winners of the ISPD 2010 clock-network contest, where each team violated prescribed skew constraints (7.5 ps in most cases) on at least some benchmarks in the presence of variations. However, results reported in this chapter satisfy skew constraints on every benchmark. Our clock trees have

4.2× smaller capacitance than clock meshes produced by CNSrouter [95], while exhibiting smaller skew.

4.1.2 Trading accuracy for runtime in SPICE analysis

The SPICE circuit simulator has been routinely used to evaluate electronic circuits since its initial release in 1973 [68]. SPICE represents circuits by nonlinear differential equations and then solves them numerically. While highly accurate, this technique is prohibitively expensive for use in many circuit optimizations. Instead, the Elmore delay model [29] is commonly used, but suffers several known shortcomings, such as its neglect for the resistive shielding effect. More advanced compact delay models, such as fitted Elmore [1] and models based on higher-order moments — S2M [2], D2M [4], LnD [5], — and more accurate models [72] [74] require heavier computation, but are still much faster than SPICE. While such models have proven accurate and useful in certain well-defined circumstances, the transition to the 45 nm technology node exposed a widening gap between the accuracy of SPICE simulation and available surrogates. In particular, many closed-form models do not capture the impact of slew rate and the direction of signal transition. The impact of process variation is another major challenge to using non-SPICE models. Additionally, accurate modeling requires a large number of device and interconnect parameters included in SPICE inputs that are not accounted for by simpler models. The need for fast SPICE-accurate circuit analysis was recently underlined by the ISPD 2009 and 2010 clock-network synthesis contests [95].

Achieving the necessary accuracy required by contest guidelines *without* SPICE-level simulation proved difficult, partly due to the presence of numerous buffers in relevant

clock networks and due to the impact of process variation. We also note that low-skew clock trees are especially unforgiving to timing-analysis inaccuracies. For example, a 5 ps error in a 500 ps path delay is only 1%, but two paths with similar delay from the clock source to sinks may result in 10 ps skew, where a 5 ps error is 50%. On the other hand, the 12-hour runtime limit could not be met with conventional SPICE-based RC clock tree generation algorithm [51] because simulation must be invoked many times per benchmark during the optimization process.

We developed the Chop-SPICE technique as a compromise (fast yet sufficiently accurate) simulator for use by our implementation *Contango 2.0*. Our divide-and-conquer approach is chosen among several other candidates for its flexible trade-off between runtime and solution quality. Chop-SPICE not only enables Contango 2.0 to meet runtime limits, but also delivers highly-accurate delay and slew estimates.

The remainder of this chapter is organized as follows. Section 4.2 describes optimization objectives and variation modeling. Section 4.3 explains initial tree construction with buffering. Section 4.4 details our techniques for robustness improvements. Section 4.5 outlines our skew optimization techniques. Section 4.6 describes our Chop-SPICE technique. Section 4.7 reports our empirical results. Summary is given in Section 4.8.

4.2 Modeling and objectives

Before introducing our clock-tree synthesis methodology in Sections 4.3—4.5, we review key optimization objectives (global and local skew), define the notion of local-skew slack, and propose a simple yet effective model of process variation.

4.2.1 Global and local skew

Common terminology and notation are introduced next.

Definition IV.3 *Given a clock tree Ψ, let $\lambda(s_i)$ be the clock latency (insertion delay) at sink $s_i \in \Psi$. Then the skew between two sinks s_i and $s_j \in \Psi$ is defined as*

$$\text{skew}^{\Psi}(s_i, s_j) = |(\lambda(s_i) - \lambda(s_j)| \tag{IV.1}$$

Global skew is defined as

$$\omega^{\Psi} = \max_{s_i, s_j \in \Psi} \text{skew}^{\Psi}(s_i, s_j) = \max_{i \in \Psi} \lambda(s_i) - \min_{i \in \Psi} \lambda(s_i) \tag{IV.2}$$

Nominal *values of* $\text{skew}^{\Psi}(s_i, s_j)$ *and* ω^{Ψ} *are computed neglecting the impact of variations.*

Global skew can be improved by decreasing $\max_{i \in \Psi} \lambda(s_i)$ (speeding up the slowest sinks) or increasing $\min_{i \in \Psi} \lambda(s_i)$ (delaying the fastest sinks). Previous publications on clock network synthesis were focused on reducing *global skew* with or without the presence of variations [12, 18, 38, 44, 51, 61, 63, 87, 97]. However, in a large clock network, skew between adjacent and connected sinks is a more meaningful optimization objective [32, 81]. *Local skew* is defined by restricting eligible sink pairs to be within distance $\Delta > 0$, which is determined for a given circuit after timing-driven placement.

Definition IV.4 *Given a clock tree Ψ and a* local skew distance bound $\Delta > 0$*, let* $\text{dist}(s_i, s_j)$ *be the Manhattan distance between sinks s_i and $s_j \in \Psi$. Then* the worst local skew *[95] is defined as*

$$\omega_{\Delta}^{\Psi} = \max_{\text{dist}(s_i, s_j) < \Delta} \text{skew}^{\Psi}(s_i, s_j) \tag{IV.3}$$

Reducing skew down to single picoseconds in the presence of variations may require a significant increase in power consumption. Since more than 30% of total power in modern microprocessors is consumed by clock networks, minimizing clock-network capacitance is as important as skew minimization. Therefore modern circuit designs can tolerate a certain amount of clock skew, and power can be reduced provided that the clock network remains below a given skew bound, even in the presence of variations.

Definition IV.5 *Consider a clock tree Ψ, a local skew distance bound $\Delta > 0$, variation model ν and target yield $0 < y \leq 1$. Let Ψ_ν be the clock tree Ψ with variation ν and $f(t)$ be the cumulative distribution function of $\omega_\Delta^{\Psi_\nu}$. Then* the worst local skew with variation *is defined as*

$$\omega_{\Delta,\nu,y}^{\Psi} = f^{-1}(y) \tag{IV.4}$$

Viewing *the local skew limit* Ω_Δ as a design constraint (see Table 2.1), we pursue the following goals.

1. Building variation-tolerant clock networks with $\omega_{\Delta,\nu,y} < \Omega_\Delta$, subject to slew constraints.

2. Minimizing clock-tree power.

4.2.2 Local-skew slack

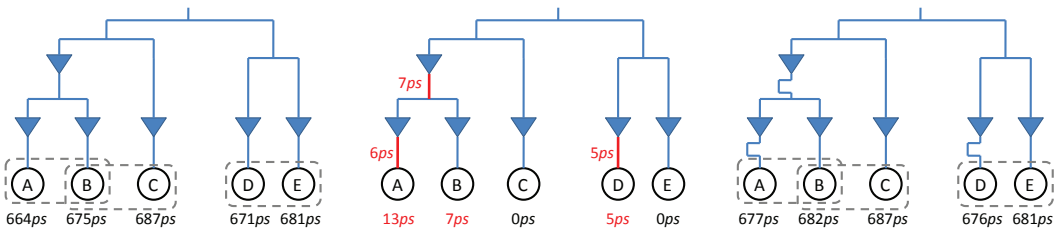
Given a clock tree with known sink latencies, one can optimize it using delay budgets derived from the sink- and edge-slack calculation [51, Section 3], followed by global skew optimization to reduce global skew below Ω_Δ. This strategy is sound because local

skew ω_Δ cannot exceed global skew. However, global skew optimizations attempt to reduce skew between sinks more distant than Δ, which may require unnecessary increase in power. To tune the clock tree on a tight power budget, we propose the concept of *local-skew slack*.

Definition IV.6 *Given a clock tree Ψ and local-skew constraints Ω_Δ, the* local-skew slack *$\sigma(s)$ for a sink $s \in \Psi$ is the minimum amount of additional delay in picoseconds for s, so that the tree satisfies $\omega_\Delta^\Psi < \Omega_\Delta$.*

The Δ-neighborhood of sink s_i is $\mathcal{N}(s_i) = \{\ s \in \Psi \mid \mathrm{dist}(s, s_i) < \Delta\}$. It is used in Algorithm 2 to calculate $\sigma(s)$ for every sink. This algorithm uses $\mathrm{varEst}(s_i, s_j) = 0$ in the absence of variations, and otherwise the definition in Section 4.2.3.

Once local-skew slacks $\sigma(s)$ are computed for all sinks, we define local-skew slack of tree edge e as the smallest slack of a downstream sink. Edge slacks in the entire tree can be computed by one recursive tree traversal in linear time, giving the optimal amount of tuning to improve worst local skew [51, Section 3]. Figure 4.1 illustrates the computation of local-skew slack for sinks and edges.



(a) Sink latencies (b) Local-skew slack on sinks and edges (c) Sink latencies after optimization

Figure 4.1: Local-skew slack for sinks and edges when Ω_Δ = 5 ps. (a) Sink pairs within distance Δ are enclosed by dashed lines. ω_Δ = 12 ps based on sink latencies and Δ. (b) Local skew-slack for sinks are computed by Algorithm 2. The algorithm for edge-slack computation is described in [51, Section 3]. (c) ω_Δ is reduced to 5 ps after optimizations, which satisfies the local skew constraints.

Algorithm 2 Computing local-skew slack for sinks

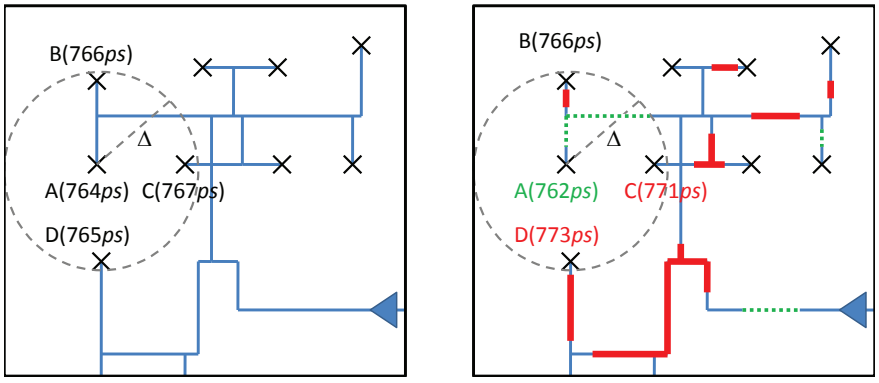
$\sigma = 0$; //Set of minimum local slack
$SinkQ = \emptyset$; //Sinks to be optimized
for each sink s_i **do**
  for each s_j in $\mathcal{N}(s_i)$ **do**
    if ($\lambda(s_i) < \lambda(s_j)$ **and**
     $\text{skew}(s_i, s_j) + \text{varEst}(s_i, s_j) > \Omega_\Delta$) **then**
      $SinkQ$.enqueue(s_i);
    end if
  end for
end for
while size($SinkQ$) $\neq 0$ **do**
  $s_i = SinkQ$.dequeue(); $MaxSlack$=0;
  for each s_j in $\mathcal{N}(s_i)$ **do**
    if ($MaxSlack < \text{skew}(s_i, s_j) + \text{varEst}(s_i, s_j) - \Omega_\Delta$) **then**
      $MaxSlack = \text{skew}(s_i, s_j) + \text{varEst}(s_i, s_j) - \Omega_\Delta$;
    end if
  end for
  $\sigma_{s_i} = MaxSlack$;
  for each s_j in $\mathcal{N}_i$ **do**
    if ($s_j \notin SinkQ$ **and** $\lambda(s_j) + \sigma_{s_j} < \lambda(s_i) + \sigma_{s_i}$ **and** $|(\lambda(s_j)$ + σ_{s_j} - $(\lambda(s_i) + \sigma_{s_i})| +$
    $\text{varEst}(s_i, s_j) > \Omega_\Delta$) **then**
      $SinkQ$.enqueue(s_j);
    end if
  end for
end while

4.2.3 Modeling process variation

Designing low-capacitance low-skew clock trees without considering process, voltage and temperature variations often results in significant skew in each chip. However, variation-aware optimization has not been explored until recently and requires reliable estimation techniques. Monte-Carlo simulations are slow and not suitable to clock network optimization. Instead, we develop a tabular technique to account for variation in single-shot timing analysis.

Our key insight is that the impact of variations on skew between two sinks is closely

correlated with tree path length and how the tree path is buffered. When two sinks can be connected by a short path in the tree, variation of skew between them is small. On the other hand, variational skew between sinks that are geometrically close can be significant if the unique tree-path between them is long. This is illustrated in Figure 4.2. For a given technology node, buffer library, wires and variation model, we propose to build a look-up table with comprehensive information regarding the worst-case variation on skew for various paths between pairs of sinks.



(a) Nominal sink latencies (b) Latencies with variation

Figure 4.2: The impact of variations on local skew. Sinks are indicated by crosses, the clock source is indicated by a solid triangle. Nominal skew of 3 ps is shown in (a). Full skew of 11 ps is shown in (b), where some tree edges are delayed (thick red) and some are sped up (dotted green) by random variations. Only sink A is within the local skew distance from sinks B, C and D.

Definition IV.7 *Given a technology node $\mathcal{T}$, buffer and wire library $\mathcal{B}$, variation model ν and desired yield $0 < y \leq 1$, let $\Xi_{\mathcal{T},\mathcal{B},\nu,y}[w, b, t]$ be the variation-estimation table which returns the worst-case increase in skew (with probability y) between two sinks connected by a tree path of length w with b buffers and the buffer type t. When multiple buffer types are used in the tree path, t is the smallest type in the tree path, so as to avoid under-estimation of variation.*

To build the table, we generated a large number of test trees on public CNS benchmarks and randomly generated benchmarks. The initial tree-construction method explained in Section 4.3 with various buffer types is utilized for the test trees. The number of Monte-Carlo SPICE simulations is determined based on the given variation model ν. Variational skew between any two sinks during the simulations is recorded in the table with classification by w, b and t. The table is later restructured to represent a probability density function for each (w, b, t) entry in order to look up with yield y. Building the variation-estimation table requires extensive simulations, but once the table is built, it can be used for many clock trees. To determine the impact of variation on skew between sinks in a clock tree, a function $\mathrm{varEst}(s_i, s_j)$ is defined as follows. Given a clock tree Ψ and a variation table $\Xi_{\mathcal{T},\mathcal{B},\nu}$, let $\mathcal{L}(s_i, s_j)$ be the total length of wires, $\mathrm{b_n}(s_i, s_j)$ be the total number of buffers and $\mathrm{b_t}(s_i, s_j)$ be the largest buffer type in the tree path between two sinks s_i and $s_j \in \Psi$. The variation table is accessed by the function $\mathrm{varEst}(s_i, s_j)$ = $\Xi_{\mathcal{T},\mathcal{B},\nu}[\mathcal{L}(s_i, s_j), \mathrm{b_n}(s_i, s_j), \mathrm{b_t}(s_i, s_j)]$.

To estimate the impact of variations when optimizing clock trees we utilize $\mathrm{varEst}()$ when computing local-skew slack for each sink (Algorithm 2). Without considering variations, it is sufficient to satisfy $\mathrm{skew}(s_i, s_j) < \Omega_\Delta$ for all pairs of sinks within Δ. However, in the presence of variations, we have the following result.

Theorem IV.1 $\omega^{\Psi}_{\Delta,\nu,y} < \Omega_\Delta$ *only if*

$$\mathrm{skew}(s_i, s_j) + \mathrm{varEst}(s_i, s_j) < \Omega_\Delta \; \forall s_i, s_j \in \Psi \tag{IV.5}$$

4.3 Initial tree construction and buffer insertion

We invoke the unmodified ZST-DME algorithm [23, 38] and perform initial buffer insertion to minimize source-to-sink Elmore delay, rather than skew or capacitance [30, 84]. Elmore delay is too inaccurate for skew optimization, but our approach creates significant room for tuning the clock tree by delaying fast paths [51]. In the presence of layout obstacles, proper obstacle-handling is required to avoid violations due to obstacles. The ISPD 2010 benchmarks include obstacles over which wire-routing is possible but buffer insertion is not allowed. We adapted a simple and robust technique for obstacle avoidance in clock trees from [51] which repairs obstacle violations in the trees obtained by the ZST-DME algorithm.

When multiple wire types are available, the choice of wires affects both total power and susceptibility to variations. Under tight skew constraints in high-performance CPU designs, thicker wires (on a given metal layer) are preferable because they limit the impact of variations and still allow for future power-performance trade-offs by wire sizing. In less aggressive ASIC and SoC designs, power optimization may motivate thinner wires. But upsizing wires in a reasonably tuned clock tree may be of limited use because it increases capacitance, potentially leading to slew violations.

Selecting buffer types for initial buffer insertion is also important. Given an initial tree without buffers Ψ_0, let $t(s_i, s_j)$ be the type of a buffer required for the tree path between two sinks s_i and $s_j \in \Psi_0$ to satisfy $\mathrm{varEst}(s_i, s_j) < \Omega_\Delta$. $t(s_i, s_j)$ can be found from the variation-estimation table $\Xi_{\mathcal{T},\mathcal{B},\nu}$ with $\mathcal{L}(s_i, s_j)$. Since $\mathrm{b_n}(s_i, s_j)$ is not available at this step, it is difficult to find the exact required $t(s_i, s_j)$. However, because $\mathrm{b_n}(s_i, s_j)$ and

$\mathcal{L}(s_i, s_j)$ are highly correlated with each other, $\mathrm{b_n}(s_i, s_j)$ can be estimated by modeling it with the average number of buffers corresponding to $\mathcal{L}(s_i, s_j)$. Once $\mathrm{b_n}(s_i, s_j)$ is estimated, $t(s_i, s_j)$ can be computed as described in Section 4.4. The initial buffer type (t_0) for a given initial tree is computed as

$$t_0 = \mathrm{Avg}_{s_i, s_j \in \Psi_0} t(s_i, s_j) \tag{IV.6}$$

Once t_0 is determined, we adopt the fast variant of van Ginneken's algorithm from [84] for initial buffer insertion. The main objective of van Ginneken's algorithm is to minimize insertion delay of a given RC tree. When clock-power reduction is more important than insertion delay, the buffering algorithm can be modified to place buffers with worse insertion delay but significantly smaller total capacitance. We modified the fast variant of van Ginneken's algorithm to minimize total capacitance subject to slew constraints in three phases. First, parameters of input instances for the buffering algorithm are adjusted to reduce buffer numbers based on given a slew constraint and a variation model. The fast variant of van Ginneken's algorithm [84] is utilized in the second phase. In the third phase, sizes of inserted buffers are adjusted to further reduce total capacitance while satisfying slew constraints.

After initial buffer insertion, $\mathrm{b_n}(s_i, s_j)\ \forall s_i, s_j \in \Psi$ is determined, and more accurate $t(s_i, s_j)$ can be obtained. For sink pairs that do not satisfy $\mathrm{varEst}(s_i, s_j) < \Omega_\Delta$, we use the robustness-improvement algorithm from Section 4.4 to ensure that the tree eventually satisfies $\omega_\Delta^\Psi < \Omega_\Delta$.

4.4 Improving robustness to variations

The initial buffer insertion algorithm cannot accurately estimate buffer types required for local-skew constraints for a given initial tree. Therefore robustness-improvement must follow after initial buffer insertion so that $\omega^{\Psi}_{\Delta,\nu,y} < \Omega_{\Delta}$ holds after all the skew optimization techniques are applied.

In an ideal situation in which we can reduce all the skew down to 0, $\mathrm{varEst}(s_i, s_j) < \Omega_{\Delta}\ \forall s_i, s_j \in \Psi$ is sufficient to satisfy $\omega^{\Psi}_{\Delta,\nu,y} < \Omega_{\Delta}$. In practice we must estimate nominal local skew $\mathrm{skew}^{\Psi}_{est}$ after accurate optimizations, which we upper-bound by 5 ps based on experience.

Theorem IV.2 *If* $\mathrm{skew}^{\Psi}_{est}$ *is an upper bound of* ω^{Ψ}_{Δ} and $\mathrm{skew}^{\Psi}_{est}$+$\mathrm{varEst}(s_i, s_j) < \Omega_{\Delta}$ *for all* s_i *and* s_j then

$$\omega^{\Psi}_{\Delta,\nu,y} < \Omega_{\Delta} \tag{IV.7}$$

The target buffer type for the tree-path between sink s_i and s_j, $t(s_i, s_j)$ can be computed as the smallest t such that

$$\Xi_{\mathcal{T},\mathcal{B},\nu}[\mathcal{L}(s_i, s_j), \mathrm{b_n}(s_i, s_j), t] < \Omega_{\Delta} - \mathrm{skew}^{\Psi}_{est} \tag{IV.8}$$

From the above method, the minimum size of buffer type which satisfies $\mathrm{varEst}(s_i, s_j) < \Omega_{\Delta}$ - $\mathrm{skew}^{\Psi}_{est}$ is selected to reduce capacitance. Once $t(s_i, s_j)$ is determined, the buffers in the tree path between sink s_i and s_j are substituted with type $t(s_i, s_j)$ buffers. This step is repeated for all eligible pairs of sinks within distance Δ.

4.5 Skew optimizations

In this section, several local skew optimization techniques are described. Each technique is designed to reduce skew under different circumstances, but the primary objective is to optimize the skew of given tree to below the local skew limit in the presence of variations. The target tuning amount for each edge of the tree can be determined by local-skew slack including variation modeling described in Section 4.2.

4.5.1 Wire snaking

Wire sizing and wire snaking are popular techniques for skew optimization and are often able to reduce global or local skew down to the practical skew limit. In this context, however, we exclude wire sizing because narrowing down a wire in the middle of a clock tree is risky due to the impact of variations. We extend the wire snaking technique from [51] to improve its speed and accuracy, while limiting its use of routing resources.

The optimal tuning amount for each edge can be obtained by the top-down slack computation explained in Section 4.2.2. Let $T_{target}(e)$ be the amount of time in ps by which the edge e must be delayed to achieve legal ω_{Δ} under local skew constraints. $L_{sn}(e)$ denotes the length of the wire determined by the wire snaking algorithm to delay the edge e by $T_{target}(e)$. Let $T_{actual}(e)$ be the amount of time in ps which the edge e is actually delayed by $L_{sn}(e)$ of a wire. Ideally, the wire snaking algorithm can estimate $L_{sn}(e)$ so that $T_{target}(e) = T_{actual}(e)$. $L_{ideal}(e)$ is the length which satisfies $T_{target}(e) = T_{actual}(e)$. The total additional capacitance from wire snaking $TotalCap_{sn}$ is

$$TotalCap_{sn} = \sum_{e_i \in E} \kappa(L_{sn}(e_i)) \tag{IV.9}$$

where $\kappa(w)$ denotes the capacitance of a wire w, and the ideal total additional capacitance $TotalCap_{ideal}$ is

$$TotalCap_{ideal} = \sum_{e_i \in E} \kappa(L_{ideal}(e_i)) \tag{IV.10}$$

Practically, $T_{actual}(e) \neq T_{target}(e)$ unless extensive SPICE simulations are performed for finding $L_{sn}(e)$, which is unrealistic in terms of runtime for a clock network synthesis flow. When $T_{actual}(e) < T_{target}(e)$, another round of wire snaking is required to bring $T_{actual}(e)$ closer to $T_{target}(e)$. $L^i_{sn}(e)$ denotes the length of the wire determined at i^{th} iteration of the wire snaking algorithm to delay the edge e. $T^i_{actual}(e)$ is the amount of time in ps by which the edge e is actually delayed by $L^i_{sn}(e)$ of a wire. $T^i_{target}(e)$ is $T_{target}(e)$ when $i = 1$ and otherwise, it is $T^{i-1}_{target}(e) - T^{i-1}_{actual}(e)$. After N iterations of wire snaking,

$$T_{actual}(e) = \sum_{i=1}^{N} T^i_{actual}(e), \quad L_{sn}(e) = \sum_{i=1}^{N} L^i_{sn}(e) \tag{IV.11}$$

Theorem IV.3 $T_{actual}(e) \leq T_{target}(e)$ *if and only if*

$$L_{sn}(e) \leq L_{ideal}(e)$$

Theorem IV.4 *If* $T_{actual}(e) \leq T_{target}(e)$ *for every* e *in the clock tree, then*

$$TotalCap_{sn} \leq TotalCap_{ideal}$$

If the wire snaking algorithm over-estimates $L^i_{sn}(e)$ and results in $T^i_{actual}(e) > T^i_{target}(e)$ for any edge e in any i-th iteration, then $TotalCap_{sn}$ exceeds $TotalCap_{ideal}$ after all the iterations of the wire snaking algorithm because it means there exists excessive delay of a wire which results in excessive delay of some sinks and possibly increases local skew

around the sinks. Therefore the wire snaking algorithm must produce $L^i_{sn}(e)$ which satisfies $T^i_{actual}(e) \leq T^i_{target}(e)$ for optimized power consumption by the clock tree. However, if the gap between $T^i_{actual}(e)$ and $T^i_{target}(e)$ is too big, more iterations will be needed for $T_{actual}(e)$ to approach $T_{target}(e)$. We improve the accuracy of wire snaking in two ways.

Delay model for wire snaking. To keep $T^i_{actual}(e) \leq T^i_{target}(e)$ with optimal quality, we define α where,

$$\alpha \leq \frac{T^i_{actual}(e)}{T^i_{target}(e)} \leq 1.0 \tag{IV.12}$$

Wire snaking algorithm aims for $T^i_{actual}(e)$ to satisfy the above inequality with the highest α possible. When α is specified, the required worst-case number of iterations of wire snaking N to make $T_{actual}(e)$ to $T_{target}(e)$ within error rate ε is

$$N = \left\lceil \frac{\log(\varepsilon)}{\log(1-\alpha)} \right\rceil \tag{IV.13}$$

Closed-form delay models like Elmore delay are not accurate enough to keep $T^i_{actual}(e) \leq T^i_{target}(e)$ and α high. To enhance the quality of estimation by the wire snaking algorithm, look-up tables for $L^i_{sn}(e)$ are built by performing a set of SPICE simulations for each technology environment which includes technology model, types of buffers and wires, variation specification. In the simulations, $T^i_{actual}(e)$ is tested with different snaking lengths on various locations of nodes in various types of clock trees. The results of simulations are stored in a look-up table, used by wire snaking during local skew optimization. We achieved α values between 60% and 70%, therefore $4 \leq N \leq 6$. Only one technology environment was used at the ISPD 2010 CNS contest, requiring a single set of simulations.

Optimal node selection for wire snaking. Figure 4.3 compares two different styles of wire snaking. Figure 4.3(b) illustrates undesired delay of sinks after wire snaking on non-

buffer-output nodes. The increased capacitance and resistance by wire snaking affects the driving buffer which results in additional delay of slow sinks. Wire snaking at buffer output nodes, as in Figure 4.3(c), is much more accurate than wire snaking at any branch. Limiting wire snaking to buffer output nodes reduces the number of SPICE calls required for clock-tree tuning. This also reduces the number of simulations for building the look-up table by limiting the number of target nodes to be tested. Wire snaking usually increases slew rate of input nodes of downstream buffers. To prevent slew violation, slew rate numbers of downstream buffers are checked and if the worst slew rate is more than 70% of the given slew limit, the target node is excluded from wire snaking.

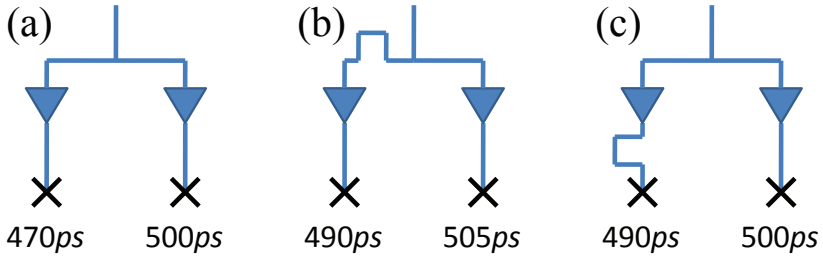


Figure 4.3: Comparison of different wire snaking strategies to satisfy $\Omega_{\Delta} = 10\ ps$. (a) Unoptimized sink latencies are shown. 20 ps of additional delay is required for the left sink. (b) Wire snaking at non-buffer output nodes results in undesired delay at the right sink. (c) The snaked wire is isolated from the right sink by the left buffer, therefore only the left sink is delayed and ω_{Δ} satisfies local skew constraints.

4.5.2 Delay buffer insertion

The local skew of a sink cluster driven by the same final buffer is often negligible. However, highly unbalanced sink capacitances or layout obstacles in those clusters can result in significant local skew. An alternative technique is needed because wire snaking in Section 4.5.1 is inapplicable. In this case, inserting a buffer at the target node is very efficient for two reasons. First, skew can be reduced by the delay of the inserted buffer.

Second, further precise wire snaking is possible because the inserted buffer isolates the target node from the remainder of the cluster.

Let $\mathcal{W}(B)$ be the set of sinks driven by a final buffer B and $d(B)$ be the delay of the buffer B. Delay buffer insertion is required if there exists $s_i, s_j \in \mathcal{W}(b)$ where $\mathrm{skew}(s_i, s_j)$ + $\mathrm{varEst}(s_i, s_j)$ - $\Omega_\Delta > d(B)$.

For each path from the buffer to the sinks, inserting at most one buffer is sufficient since the wire snaking algorithm in Section 4.5.1 can be invoked again at the output node of inserted buffers. Figure 4.4 illustrates delay buffer insertion algorithm followed by wire snaking. When a delay buffer is inserted, it is placed at the node so that the input capacitance of a delay buffer is comparable to the sum of downstream sink and wire capacitance of the target node, thus sink latency in the other path changes very little. See Figure 4.4b.

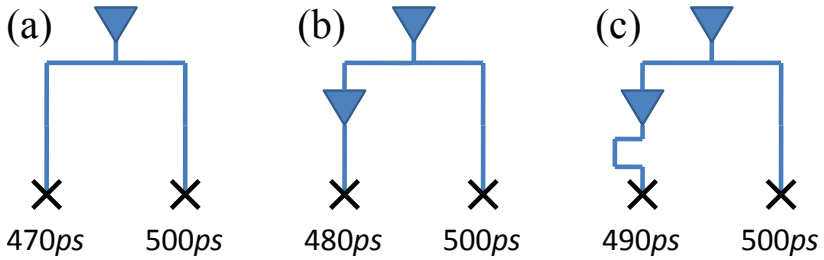


Figure 4.4: Delay buffer insertion and subsequent wire snaking when Ω_Δ= 10 ps, the delay of the buffer $d(B)$= 10 ps. (a) Unoptimized sink latencies are shown. (b) Delay buffer insertion for skew reduction and isolation of the target node. (c) The snaked wire is isolated from the right sink by the delay buffer.

4.6 Chop-SPICE: an efficient SPICE simulation technique

To ensure SPICE-accurate clock-network optimization that adequately accounts for numerous relevant technology parameters, we embed a simulator in the optimization loop. However, running SPICE directly requires unacceptably large runtime. Therefore, we develop a technique to reduce SPICE runtime at the modest cost in terms of accuracy.

4.6.1 Runtime versus accuracy

To select an appropriate approximation paradigm for faster circuit simulation, it is important to identify key problems with available solutions. SPICE produces highly accurate results and is uncontested on very small (sub)circuits. However, its runtime scales worse than linear as circuits grow [41]. Compact and algorithmic delay models scale much better, but their accuracy lags behind SPICE results even for fairly small (sub)circuits. Several factors contribute to loss of accuracy: (i) the combination of buffers and inverters in clock trees makes it difficult to precisely calibrate technology parameters, (ii) closed-form models do not account for sufficiently many technology parameters, (iii) many closed-form models do not adequately model slew propagation and its impact on delay, and (iv) systematic computational inaccuracies tend to snowball in RC trees, especially without averaging or corrections.

Given our initial emphasis on runtime scaling, rather than actual runtimes, using SPICE appears the best option on small (sub)circuits. To avoid non-linear runtime scaling, we partition the original RC tree into sub-circuits, invoke SPICE on them, and assemble results to approximate full SPICE simulation. Partitioning must be performed so that resulting (sub)circuits are sufficiently independent. This step can be performed by splitting the original tree at buffers, which effectively shields downstream capacitance from upstream drivers. On the other hand, the tree topology prevents complicated min-cut configurations, and sub-circuit selection can be based largely on the desired granularity as a means to control the accuracy-runtime trade-off.

4.6.2 The Chop-SPICE algorithm

The divide-and-conquer framework we propose is simple as well as practical. While it is not as mathematically sophisticated as alternative delay models, it is considerably easier to implement and inherits the accuracy and reliability of the SPICE simulator.

Using any tree traversal technique, such as BFS, Chop-SPICE divides the circuit based on *target granularity* γ (number of probing points). Once γ is met, Chop-SPICE (i) stops the tree traversal, (ii) performs the SPICE simulation on the sub-circuit, (iii) retrieves the relevant simulation results, and (iv) propagates the delay and slew information to input nodes of subsequent sub-circuits. Such iterations continue until the original RC tree is exhausted. Further details of the Chop-SPICE algorithm are presented in Algorithm 3.

Formally, given an RC tree Ψ, let ω^{Ψ} be the set of probing points in Ψ, which are defined as input nodes of buffers or sink nodes (Figure 4.5). Let $\lambda(s_i)$ be the number of fanouts at node $s_i \in \Psi$ to probing points in ω^{Ψ}. Then, the maximum granularity $\alpha(\Psi)$ and minimum granularity $\beta(\Psi)$, and granularity range $\theta(\Psi)$ of Ψ are defined as

$$\alpha(\Psi) = \sum_{s_i \in \Psi} \lambda(s_i) \tag{IV.14}$$

$$\beta(\Psi) = \min_{s_i \in \Psi} \lambda(s_i) \tag{IV.15}$$

$$\theta(\Psi) = \alpha(\Psi) - \beta(\Psi) \tag{IV.16}$$

Details of sub-circuit generation. In generation of sub-circuits, if a probing point $p_i \in \omega^{\Psi}$ is an input node of buffer(s), all fanout buffers are also explicitly included in the current sub-circuit (Figure 4.6). Hence, sub-circuits are always delimited by buffers and/or sinks, and a buffer at the boundary of a sub-circuit may also appear in another sub-circuit. This is convenient for delay modeling because buffers effectively shield most of

downstream capacitance (in terms of first-order analysis) from upstream circuit elements, facilitating accurate reconstruction of circuit delay from sub-circuit simulation data.

Delay and slew propagation. Our implementation of Chop-SPICE relies on the standard Π modeling of interconnect [76]. After the probing points' delay and slew are calculated by SPICE, they can be propagated in order to accurately capture delay and slew for probing points in subsequent sub-circuits. Note that probing points in ω^{Ψ} can become input nodes s_i of sub-circuits at the subsequent stage, so long as they are input nodes of buffers. Therefore, measured delays can be directly propagated to the following stages. Delay from the root node s_0 to another node s_j is calculated in separate steps. First, we find the sub-circuit containing s_j. Second, we identify the shortest tree path from s_0 to s_j, and the earliest node s_i in the sub-circuit that lies on this tree path. We now assume that signal delay at s_i was computed at previous stages of Algorithm 3, i.e., recursively. The delay from s_i to s_j is obtained by SPICE simulation and added to delay at s_i.

Slew at a given node can be expressed as a function of input slews of an upstream sub-circuit. Therefore, slew measured at the previous stage (up to the root node in a given sub-circuit) should be accounted for when stimuli for the current sub-circuit are generated. Slew rate for stimuli should be the same as slew rate at the input node s_i of the sub-circuit. Then, slew at a node is directly calculated by SPICE simulation.

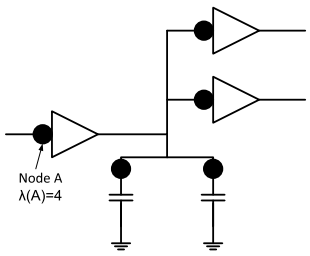


Figure 4.5: Probing points in an RC network.

Algorithm 3 Chop-SPICE

Circuit instance Ψ
Target granularity γ, where $\beta(\Psi) \leq \gamma \leq \alpha(\Psi)$
Root node of RC tree=s_0
Tree traversal queue Q=$\emptyset$

1: Q.enqueue(s_0)
2: **while** !Q.empty() **do**
3: $s_i = Q$.dequeue()
4; **if** s_i is sink **then** continue
5: Empty sub-circuit instance Ψ_0
6: Probing point queue P=$\emptyset$
7: Fanout node queue F=$\emptyset$
8: F.push(fanout nodes of s_i)
9: **while** P.size() $< \gamma$ **do**
10: Subsequent input node queue E={}
11: **while** !F.empty() **do**
12: $f_o = F$.dequeue()
13: Copy the delay of s_i to the delay of f_o
14: **if** f_o is $\in \omega^{\Psi}$ **then**
15: P.enqueue(f_o)
16: E.enqueue(fanout nodes of f_o)
17: write the specification of the buffer / sink to Ψ_0
18: **else if** f_o is a wire segment **then**
19: write the specification of the wire segment to Ψ_0
20: F.enqueue(fanout nodes of f_o)
21: **end while**
22: F.enqueue(every e $\in E$)
23: **end while**
24: Generate stimuli based on slew of s_i
25: Invoke SPICE simulation on Ψ_0
26: Retrieve simulation results
27: Update delay and slew information on P
28: Q.enqueue(every e $\in E$)
29: **end while**

4.6.3 Integration with Contango

While Chop-SPICE exhibits good fidelity of delay estimation as discussed in Section 4.7.4, source-to-sink delay errors reported in Table 4.4 cannot be ignored during skew optimizations because typical target nominal skews are below 5 ps. Therefore we propose a technique that estimates delay errors and compensates them for higher accuracy. Due to high fidelity of Chop-SPICE, actual delays measured by full-scale SPICE can be estimated from delays measured by Chop-SPICE.

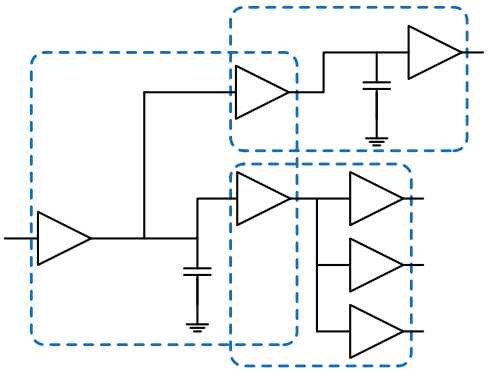


Figure 4.6: Sub-circuits are delimited by boundary buffers.

During SPICE-driven skew optimizations, we invoke Chop-SPICE in two steps. First, before SPICE-driven skew optimizations, we run full-scale SPICE and Chop-SPICE and compare sink delays measured by both methods. Then we calculate *scaling factors* for each sink for delay compensation. For example, a delay from a source to a sink s may be measured as 1010 ps and 1000 ps by full-scale SPICE and Chop-SPICE respectively. Then we set the scaling factor for sink s to 1.01. Second, scaling factors are utilized to estimate actual sink delays based on delays measured by Chop-SPICE. In the previous example, it may turn out that the delay of sink s must increase at least to 1020 ps to satisfy a given skew constraint. After iterations of skew optimizations, the delay of sink s is measured as 1010 ps by Chop-SPICE. Based on the scaling factor 1.01, we estimate that the actual delay of sink s is 1020.1 ps, which satisfies the target delay. We achieve nominal skews under 5 ps based on the above method while runtime improvement is more than 2.5$\times$.

4.7 Empirical validation: Contango 2.0

Our implementation, Contango 2.0, is written in C++ and is based on our software Contango 1.0 [51] that shared the first place at the ISPD 2009 clock-network synthesis

	CNSROUTER 3/16/2010				NTUCLOCK 3/16/2010				CONTANGO 2.0 4/14/2010					
	Skew+Var		Tot.		Skew+Var		Tot.		Nom.	Skew+Var			Total Cap.	
	Mean	95%	Cap.	◷	Mean	95%	Cap.	◷	skew	Mean	95%	Yield	(wire snaking)	◷
cns01	5.27	7.32	841	20	6.71	~~8.66~~	294	15	2.51	5.16	7.01	97.4%	**198**(2.6)	12015
cns02	6.27	~~8.33~~	1454	81	8.27	~~10.7~~	833	176	2.99	5.58	7.34	95.8%	**376**(1.3)	25006
cns03	2.07	2.70	163	8	6.82	~~8.63~~	167	6	1.5	3.03	4.18	99.6%	**55.9**(1.0)	3840
cns04	2.83	3.98	277	32	7.45	~~9.55~~	325	58	2.07	3.26	4.46	99.9%	**71.84**(1.46)	6075
cns05	2.88	4.38	176	7	5.30	6.98	130	11	1.50	3.01	4.41	99.9%	**37.7**(5.43)	2406
cns06	12.4	~~14.0~~	133	11	407	~~417~~	1577	10	4.29	5.03	6.05	99.9%	**47.8**(3.0)	2660
cns07	12.4	~~15.7~~	309	45	6.17	~~8.12~~	276	66	2.22	3.41	4.58	99.9%	**72.7**(1.1)	2351
cns08	6.15	7.33	223	19	5.94	~~7.64~~	166	7	3.42	4.15	5.15	99.9%	**52.5**(1.9)	1987
Rel.			**4.22**				**4.13**						**1.0**	

Table 4.1: Results on the ISPD 2010 Contest benchmark suite. Skew numbers are reported in ps, capacitance in pF and CPU time in s. '95%' represents $\omega_{\Delta,\nu,95}$. The numbers in parentheses of the capacitance column refer to the fraction of capacitance of the snaked wires in %. Skew constraint violations are shown in strikethrough font. Otherwise, skew results are not comparable because skew can be traded for capacitance, which was the primary objective of the contest. All networks produced by these tools satisfy slew constraints imposed at the ISPD 2010 contest. Due to limited page space, we do not include results for the other teams, but significantly outperform them in solution quality.

contest. Contango 2.0 was the sole winner of the ISPD 2010 contest, but we now report significantly stronger results.

4.7.1 ISPD 2010 benchmarks

Table 2.1 lists the statistics of all benchmarks from the ISPD 2010 contest. The insertion-delay driven buffer insertion algorithm [84] is utilized to produce our clock trees in this experiment. The contest limited slew to 100 ps, and all reported clock networks satisfy this constraint. Slews in Contango 2.0 trees do not exceed 81 ps. Table 4.1 compares Contango 2.0 with CNSrouter and NTUclock. Clock networks produced by our software have smaller capacitance than CNSrouter and NTUclock on average by **4.22**$\times$ and **4.13**$\times$ respectively. The contest imposed local skew constraints with yield $y = 95\%$. Our clock trees always yield $> 95\%$, while CNSrouter violates yield constraints on three benchmarks and NTUclock on all benchmarks except one. All three teams satisfied the 12-hour

runtime limit for all benchmarks. Our data suggest that wire snaking usually increases wire length by 1-3% (5.43% in one case), which is small enough to neglect the negative effects of wire snaking. Figure 4.8 compares probability density functions (pdf) produced by Monte-Carlo SPICE simulations of our clock trees to those of clock meshes produced by CNSrouter. One such clock tree is illustrated in Figure 4.7. Despite the dramatic differences in network topology and total capacitance between trees and meshes, some of the plots in Figure 4.8 bear striking resemblance (cns01, cns02, cns04, cns05). To explain this phenomenon, we recall that meshes cannot be buffered directly and are therefore driven by a buffered clock tree. Such a clock tree can be constructed by the same DME algorithm that we use, which is why the pdf profiles in Figure 4.8 reflect the pointset of sink locations. Apparently, the mesh does not significantly change this profile.

4.7.2 Using our slew-constrained buffering algorithm

Table 4.2 compares our clock trees with the slew-constrained buffering algorithm in Section 4.3 to the existing state-of-the-art clock networks [13,67] on the ISPD 2010 benchmarks. Compared to our clock trees in Table 2.1, we reduce total capacitance by 32.8% on average. Clock networks produced by our software exhibit smaller capacitance than the clock networks in [13] and [67] on average by 23.0%, 1.2% respectively. All the results in the Table 4.2 satisfy given skew and slew constraints.

4.7.3 Power versus robustness to variations

Figure 4.9 describes experiments on benchmark *ispd10cns08* with different local skew constraints. When tight local skew constraints are given, large buffers are required to ensure robustness to variations, increasing the capacitance of the clock tree. On the other

ISPD'10 Bench.	[13] 3/27/2011			[67] 3/27/2011			CONTANGO 2.0 9/18/2011		
	Skew+Var		Total	Skew+Var		Total	Skew+Var		Total
	Mean	$\omega_{\Delta,\nu,95}$	Cap.	Mean	$\omega_{\Delta,\nu,95}$	Cap.	Mean	$\omega_{\Delta,\nu,95}$	Cap.
cns01	4.01	5.79	177.5	5.06	7.32	142.6	5.20	7.25	**141.5**
cns02	4.98	6.69	329.9	5.81	7.42	265.2	5.79	7.42	**264.4**
cns03	2.44	3.46	50.81	2.74	4.49	36.61	3.39	4.63	**35.81**
cns04	2.84	3.79	57.44	3.96	6.70	51.07	3.91	5.26	**47.90**
cns05	2.72	3.68	28.93	2.16	4.78	**25.13**	4.31	6.57	25.45
cns06	3.03	4.01	36.12	4.73	6.41	**32.68**	4.56	6.37	35.02
cns07	3.81	5.65	57.93	4.04	5.86	48.32	4.48	5.99	**47.09**
cns08	2.89	4.24	40.43	3.41	5.07	32.70	4.02	5.71	**31.18**
Relative			1.230			1.012			**1.000**

Table 4.2: Our clock trees for the ISPD 2010 benchmarks, buffered by our slew-constrained algorithm, versus existing state-of-the-art clock networks [13, 67]. Skew numbers are reported in ps, capacitance in pF. All networks produced by these tools satisfy slew constraints from the ISPD 2010 contest.

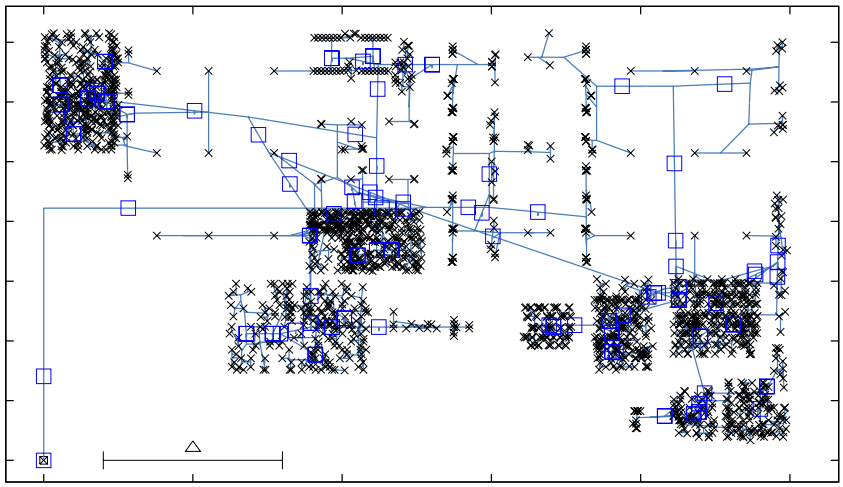


Figure 4.7: Our clock tree for *ispd10cns07*. Sinks are indicated by crosses, buffers are indicated by blue rectangles. $\Delta = 600\mu m$ is shown near the left-bottom corner.

hand, a large portion of capacitance can be saved when local skew constraints are loose.

To clarify the impact of variation, we plot variational skew (y-axis), defined as $\omega_{\Delta,\nu,y}$ - ω_{Δ} for Δ, ν, y from Table 2.1.

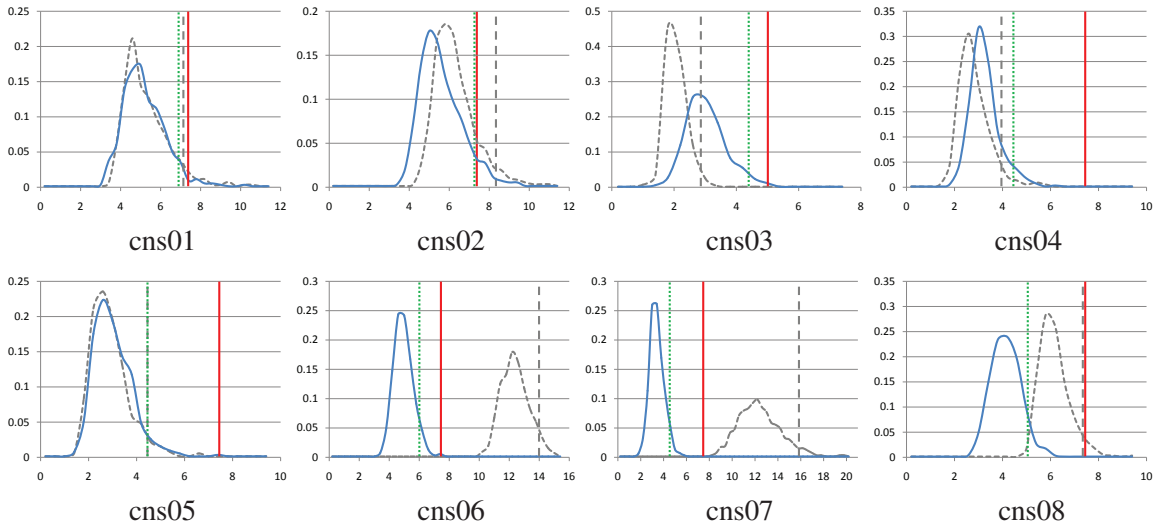


Figure 4.8: Probability density functions for worst local skew of our clock trees (blue line) and meshes produced by CNSrouter (gray dashed line) for the eight ISPD 2010 benchmarks, calculated using 500 independent SPICE runs for each benchmark. The x-axis shows skew in picoseconds. Local skew limits (Ω_{Δ}) are shown with red solid lines, and the 95%-ile of local skew ($\omega_{\Delta,\nu,0.95}$) are shown by dotted green lines (our work) and dashed gray vertical lines (CNSrouter).

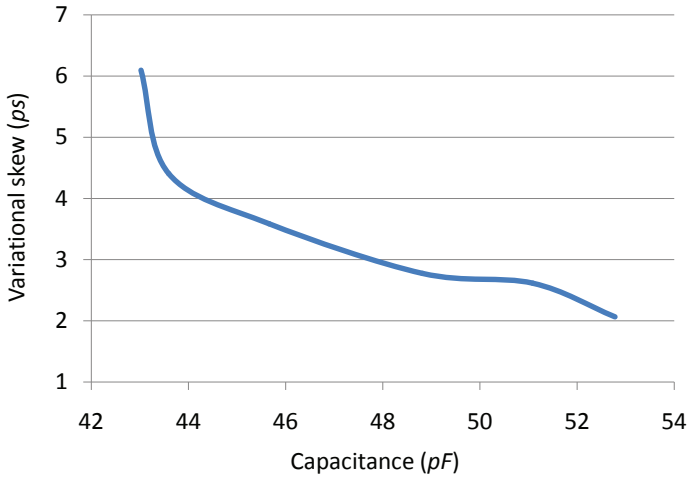


Figure 4.9: Trade-off between capacitance and robustness on *ispd10cns08*. The x-axis represents total capacitance of a tree and y-axis represents the maximal variational skew at 95% yield.

4.7.4 Empirical validation of Chop-SPICE

Runtimes and errors. Table 4.3 and Table 4.4 show Chop-SPICE runtimes and errors as target granularity γ varies from $\alpha(\Psi)$ (full-scale SPICE simulation) to $\beta(\Psi)$ on each circuit. Delay and slew errors at each sink are measured by absolute difference with delay and slew, respectively, from the full-scale SPICE simulation of each circuit. With decreasing granularity and probing points in sub-circuits, delay and slew errors increase while runtimes decrease. We observed runtime improvements of 1.2 - 4.6×, delay errors of 4.99 - 10.73 ps, and slew errors of 2.33 - 4.21 ps per benchmark for the smallest sub-circuits considered in our experiments.

Fidelity of delay estimates. In addition to accuracy, we show that Chop-SPICE delay estimates exhibit good *fidelity*, suggesting that Chop-SPICE is effective as a replacement of full-scale SPICE during optimization. To this end, we analyze intermediate clock trees produced by *Contango2*. During each iteration, we use Chop-SPICE and full-scale SPICE to measure sink delays for clock trees *before* and *after* optimization, and compare the delay differences. Let Ψ^k = clock tree at the k^{th} iteration of *Contango2* optimization. Then, considering clock trees Ψ^k and Ψ^{k+1}, we calculate the delay difference for each sink between Ψ^k and Ψ^{k+1}, as measured by Chop-SPICE and full-scale SPICE. Let delay$^k(s_i)$ be the delay of node s_i on RC clock tree Ψ^k, then $\Delta^k(s_i)$ = delay$^{k+1}(s_i)$ – delay$^k(s_i)$ where $s_i \in \omega^\Psi$, k=$\{0,1,..., K(\Psi)\text{-}1\}$, and $K(\Psi)$ is the total number of optimization iterations. Empirical results are depicted in Figure 4.10. For each datapoint, the x-axis represents the delay difference measured by Chop-SPICE while the y-axis represents the delay difference measured by full-scale SPICE. Five to six optimization iterations of *Contango2*

were used. Most $\Delta^k(s_i)$ values are positive since the *Contango2* optimizations increase sink delays to meet the *local skew limits*.

For datapoints that fall on the straight line, Chop-SPICE and full-scale SPICE agreed on delay differences; datapoints above the straight line indicate that Chop-SPICE underestimates the delay difference while datapoints below the line indicate that Chop-SPICE overestimates the delay difference. As shown in the figure, the majority of datapoints fall slightly above or below the line (a difference of a few picoseconds). Moreover, we have shown in Section 4.7.4 that the delay and slew errors of using Chop-SPICE are within 10 ps. Therefore, Chop-SPICE can be reliably used instead of full-scale SPICE.

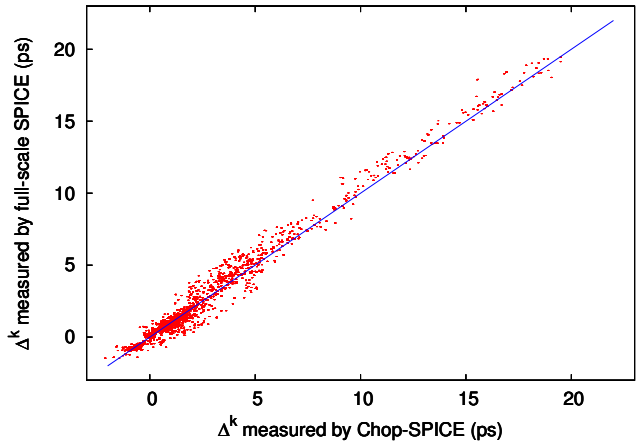


Figure 4.10: Fidelity of Chop-SPICE estimates ($\gamma = \beta(\Psi)$) for CNS02, CNS07, and CNS04 ISPD 2010 benchmarks.

4.7.5 Comparison with a commercial clock-tree synthesis tool

Table 4.5 compares our software to a commercial clock-tree synthesis tool, *Cadence First Encounter*. We worked with a 121 $\mu m \times 121$ μm circuit block based on a 65 nm technology with a 5 ns target clock period. When skew requirement is set to 2 ps, Encounter produces a clock tree with 97.0 ps worst insertion delay and 8.5 ps nominal skew.

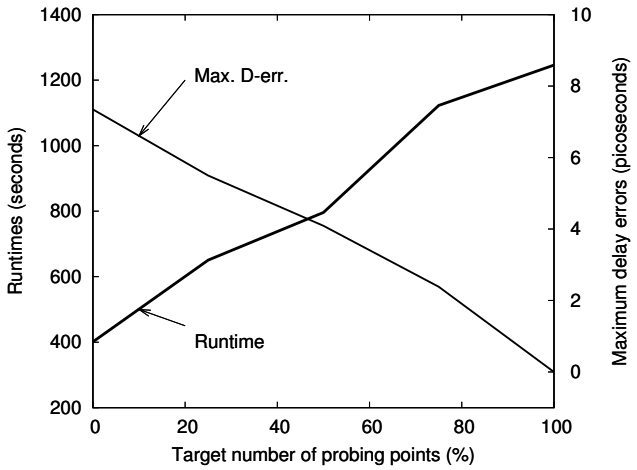


Figure 4.11: Trade-offs between accuracy and runtime.

	$\alpha(\Psi)$	$\beta(\Psi)$+0.75·$\theta(\Psi)$			$\beta(\Psi)$+0.5·$\theta(\Psi)$			$\beta(\Psi)$+0.25·$\theta(\Psi)$			$\beta(\Psi)$		
Bench.	◷	D-err.	S-err.	◷	D-err.	S-err.	◷	D-err.	S-err.	◷	D-err.	S-err.	◷
CNS01	227	0.383	0.804	213	0.728	0.731	148	0.963	1.001	133	2.413	1.225	76.6
CNS02	715	0.315	0.748	638	0.433	0.740	411	0.834	1.168	339	3.584	1.429	157
CNS03	43.9	0.238	0.313	41.3	0.365	0.720	31.0	0.630	0.944	29.6	1.564	0.979	24.8
CNS04	77.5	0.221	0.450	69.2	0.438	0.896	60.4	0.731	0.776	31.4	1.519	1.016	32.9
CNS05	26.1	0.330	0.580	24.1	0.579	0.439	23.5	0.764	0.470	22.7	0.950	0.594	21.6
CNS06	32.3	0.237	0.360	31.8	0.546	1.228	29.1	0.810	1.386	27.9	1.181	1.477	27.0
CNS07	81.7	0.262	0.427	71.7	0.543	0.595	61.9	1.123	0.827	38.9	1.624	0.911	33.6
CNS08	42.5	0.339	0.634	33.4	0.794	0.828	30.9	1.614	1.003	28.1	1.942	1.262	27.8
Ave.	1.0×	0.291	0.540	1.1×	0.553	0.772	1.6×	0.934	0.947	1.9×	1.847	1.112	3.1×

Table 4.3: Averages delay (D-err.) and slew error (S-err.) (in picoseconds) and runtimes (in seconds) with varying the granularity on the ISPD 2010 CNS contest benchmark suite. The clock networks are synthesized by *Contango2*.

We export the locations of the registers and create a 45 nm ISPD-compatible benchmark with chip size 84 μm by 84 μm and the target clock period 2 ns. Our software produces a clock network with 62.6 ps worst insertion delay and 1.6 ps nominal skew. To compensate for the differences in technology parameters, note that skew improvement considerably outpaces insertion-delay improvement. The result shows that our nominal-skew optimization techniques are more precise and reliable than Encounter when a given skew constraint is difficult to satisfy.

Bench.	$\beta(\Psi)$+0.5·$\theta(\Psi)$		$\beta(\Psi)$+0.25·$\theta(\Psi)$		$\beta(\Psi)$	
	D-err.	S-err.	D-err.	S-err.	D-err.	S-err.
CNS01	1.861	2.152	1.882	2.710	4.993	3.737
CNS02	6.775	3.847	6.757	3.839	10.745	4.214
CNS03	1.955	2.551	1.656	2.559	6.764	2.684
CNS04	1.462	2.423	7.623	1.885	7.623	2.609
CNS05	4.880	1.853	4.880	1.853	7.067	2.328
CNS06	7.063	2.860	7.063	3.107	7.063	2.908
CNS07	1.625	2.862	7.046	2.229	7.449	2.740
CNS08	7.107	3.132	7.079	3.637	7.074	3.645
Average	4.091	2.710	5.498	2.727	7.347	3.108

Table 4.4: Maximum delay (D-err.) and slew errors (S-err.) in ps, on the ISPD 2010 CNS contest benchmark suite, as sub-circuit granularity varies.

	Insertion Delay				Nominal Skew	
	rise		fall		rise	fall
	min (ps)	max (ps)	min (ps)	max (ps)	(ps)	(ps)
ENCOUNTER	86.5	95.0	88.7	97.0	8.5	8.3
our work	59.9	61.2	61.0	62.6	1.3	1.6

Table 4.5: Comparison of our software on a design with 309 registers to a commercial clock-tree synthesis tool, *Cadence First Encounter*. Skew limit 2.0 ps is used to produce each clock network.

4.8 Summary

Power-performance-cost trade-offs are becoming a major issue in modern high-performance CPU clock designs. Mesh structures often sacrifice power to improve robustness to variations. We propose a tree solution for CPU clock routing that improves power consumption under tight skew constraints in the presence of variations. To this end, we introduce the notion of local-skew slack for clock trees, a model for variational skew, a path-based technique to enhance robustness, a new time-budgeting algorithm for clock-tree tuning and accurate optimizations that satisfy budgets. We have shown that clock trees can be tuned to have nominal skew below 5 ps and total skew in single picoseconds in the presence of variations. Our optimizations not only satisfy given skew constraints and target yield but

also lead to $4.22\times$ capacitance improvement on average over mesh structures proposed in the ISPD 2010 contest. Furthermore, our clock trees had a higher yield than the meshes because meshes are not as easy to tune for nominal skew. The comparison with a commercial clock-tree synthesis tool shows that our nominal-skew optimization techniques are more precise and reliable than an industrial tool when a given skew constraint is difficult to satisfy. Our analysis does not consider gated clocks, inductive effects and short-circuit power in meshes, but these factors generally favor trees over meshes. Our strong empirical results suggest that clock trees constructed using accurate variational skew modeling and optimizations have distinct advantage in power consumption and similar robustness as meshes. Hence, our techniques may improve power of future CPUs without sacrificing other performance metrics.

PART III

Increasing the Scope of Clock-Network-Synthesis Optimizations

CHAPTER V

Obstacle-aware Clock-tree Shaping during Placement

Traditional IC design flows optimize clock networks after placement and thus limit the quality of clock networks by register locations. Existing publications also reflect this bias and focus mostly on clock routing. The few known techniques for register placement exhibit significant limitations and do not account for recent progress in large-scale placement and obstacle-aware clock-network synthesis.

In this chapter, we integrate clock network synthesis with global placement by optimizing register locations. We propose (1) obstacle-aware virtual clock-tree synthesis; (2) arboreal clock-net contraction force with virtual-node insertion, which can handle multiple clock domains and gated clocks; (3) an obstacle-avoidance force. Our work is validated

on large benchmarks with numerous macro blocks. Experimental results indicate that our software implementation, called Lopper, prunes clock-tree branches to reduce their length by 30.0%∼36.6% and average total dynamic power consumption by 6.8%∼11.6% versus conventional wirelength-driven approaches. SPICE-driven simulations show that our methods improve robustness of clock trees.

5.1 Introduction

Our analysis of prior work reveals serious limitations in published techniques. Some methods coerce the placer into shortening the clock tree by capturing portions of the clock tree with the half-perimeter wirelength (HPWL) objective, which is usually applied only to signal nets [20, 103]. This idea overlooks the fact that low-skew clock trees exhibit much greater wirelength than signal nets with the same bounding box. To make matters worse, the HPWL estimate does not offer much fidelity for clock-tree lengths, as we show in Figure 5.2. Furthermore, a handful of existing publications that optimize clock networks during placement (reviewed in Section 5.2) do not reflect recent progress in large-scale placement and clock-network synthesis, and do not compare their results with best-of-breed software. In most cases, they are evaluated on small benchmarks without routing/buffering obstacles rather than on modern ASIC or SoC designs with many macro blocks. *Our research addresses these gaps in the literature by developing a set of new techniques for clock-net optimization during placement and evaluating these techniques against leading academic software*. We extended the ISPD 2005 benchmark suite toward clock-network synthesis, with the largest benchmark including 2.1M standard cells and 327K registers. The benchmarks include macros, which we interpret as routing obstacles.

To optimize the trade-off between clock network minimization and traditional placement objectives, we propose a new placement methodology based on *obstacle-aware virtual clock-tree synthesis* that extends force-directed placement by adding *an arboreal clock-net force* using virtual nodes. *A key challenge addressed in our work is preserving the quality of global placement when adding clock-net optimizations.* We also accommodate multiple clock domains and gated clocks. Our algorithms are integrated into the SimPL placer [47], which currently produces lowest-wirelength placements on the ISPD'05 benchmarks. The quality of register placement is evaluated by Contango 2.0 [53] – the winner of the ISPD 2010 contest. Experimental results show that our method can reduce clock-network capacitance by 30.0%$\sim$36.6% while reducing the overall dynamic power of the IC by 6.8%$\sim$11.6% compared to conventional approaches.

Modern CPU designs demand low-power clock networks, yet also impose stringent skew limits, especially in the presence of process, voltage and temperature (PVT) variation for sub-45 nm CMOS technologies. Clock networks that are robust to PVT variations are usually not power efficient. To this end, our proposed methodology integrates variation-sensitive virtual clock-tree construction into the primary optimization objective of global placement, and therefore produces more robust clock trees without increasing power. Empirically, we increase clock-tree yield by 24.6% compared to state-of-the-art wirelength-driven optimizations.

The remainder of this chapter is organized as follows. Section 5.2 covers limitations of existing techniques. Section 5.3 reviews the optimization objective for clock-net optimization in placement subject to dynamic-power reduction. Section 5.4 describes our tech-

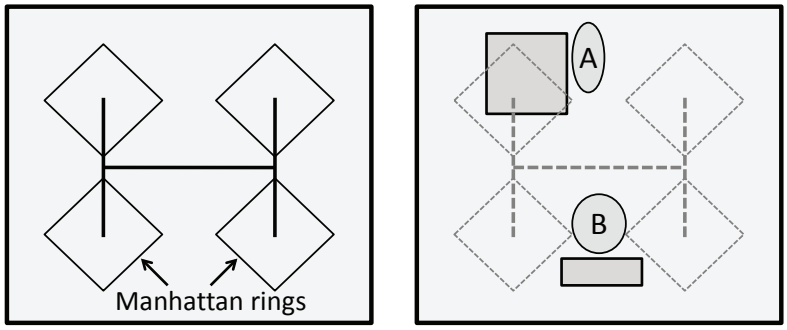
niques for high-quality register placement. Section 5.5 describes our methodology for integrating proposed techniques into a state-of-the-art placer used in industry and academia. Our empirical results are described in Section 5.6. Summary is given in Section 5.7.

5.2 Limitations of existing techniques

Clock-net optimization during placement seeks better register locations but should not harm total wirelength of signal nets. A naive method is to increase the weight of the clock net and pull all registers together. Unfortunately, this method increases routing congestion and hot spots, and also leads to poor signal-net wirelength when dealing with more than several hundred registers [20,103]. To resolve the conflict between clock-net minimization and traditional placement objectives, careful problem formulation is essential.

Prior approaches to clock-net minimization in placement form two families. *Manhattan-ring guidance methods* commit registers to certain guidance locations and try to pull the registers close to the nearest such locations during placement [64]. However, such methods do poorly in the presence of numerous obstacles, e.g., macro-blocks, or when register locations found by the global placer are not uniformly distributed. In other words, guidance rings cannot accurately predict ideal locations for register clusters. Figure 5.1 illustrates how Manhattan-ring methods fail. In Figure 5.1(b), the sink group A is attracted by the closest Manhattan ring. The sinks in A are erroneously guided toward the obstacle. The sink group B and the related standard cells have heavy connections to the bottom macro block. However, the two bottom Manhattan rings encourage the sinks in B to move away from the center of B, which will likely increase signal-net wirelength significantly.

The second family of approaches performs clock-network synthesis using register lo-



(a) (b)

Figure 5.1: Two examples of Manhattan rings proposed in [64]. (a) Zero-skew Manhattan rings driven by an H-tree. (b) Manhattan rings on the design with obstacles. Obstacles are indicated by darker boxes, two sink groups (A, B) are represented as ellipses.

cations from intermediate placement results. Specific techniques [20, 103] often simplify the structure of the clock network and bias the placement process to optimize such simplified networks. However, clock trees generated by those techniques are not realistic and very different from those generated by leading software. In the DCTB algorithm [103], the essential parameters of clock network synthesis, such as sink capacitance and wire capacitance/resistance, are ignored, and the cost function is derived by only considering Manhattan length between sinks or nodes. The quick CTS algorithm in [20] relies on simple heuristic clustering for topology generation and is more simple-minded than standard DME algorithms, which minimize wirelength with zero or bounded skew based on Elmore delay. Furthermore, all previous work ignores the presence of routing obstacles, common in modern IC designs, and this ignorance can undermine end results (Sections 5.4 and 5.6).

Previous publications that simplify clock-tree synthesis during placement [20, 103] cluster clock trees and represent these clusters with bounding boxes to model clock network reduction by placement objectives. Typically, registers are clustered at one or mul-

tiple levels based on the structure of the reference (simplified) clock tree, and bounding boxes are created for each cluster. The experimental results of [20, 103] show that bounding boxes are helpful for clock-net size reduction. However, we argue below that this method fails to represent clock-net reduction problem in placement.

Bounding boxes are represented by fake nets during placement and are optimized to reduce HPWL [47, 91]. The HPWL objective is relevant to placement because it estimates the lengths of signal routes reasonably well. However, clock routing is very different from signal-net routing and requires longer routes to ensure low skew. Therefore, HPWL does not offer accurate estimates of clock-tree lengths. Figure 5.2 shows that reducing HPWL of the clock net may increase the total length of the clock tree, demonstrating that the HPWL estimates lack not only accuracy, but also fidelity.

The authors of [103] adapted MLAF to compensate for the drawback of MLBB. However, we show in Section 5.4.2 that MLAF offers only a partial solution to this problem.

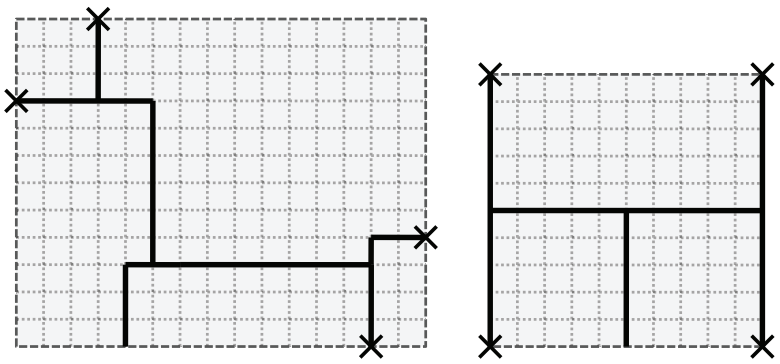


Figure 5.2: Bounding boxes of two partial ZST-DME clock trees. (a) HPWL of the bounding box is (15+12)=27. The total wirelength of the inside clock tree is 32. (b) HPWL is (10+10)=20 and the total wirelength of the clock tree is 35. The clock-net wirelength of (b) is greater than (a) although the bounding-box HPWL of (b) is notably smaller than (a) while the source-to-sink wirelength is 15 for all sinks.

5.3 Optimization objective

Let $\mathcal{N}$ be the set of signal nets, and let $\mathcal{E}$ be the set of clock-net edges. To optimize clock networks in placement, we minimize the total switching power P_{sw}, defined as the sum of $\mathcal{N}$'s switching power $P_{\mathcal{N}}$ and $\mathcal{E}$'s switching power $P_{\mathcal{E}}$

$$P_{sw} = P_{\mathcal{N}} + P_{\mathcal{E}} \tag{V.1}$$

Given activity factors of signal nets and clock-net edges, the total signal-net and clock-net switching power are

$$P_{\mathcal{N}} = \sum_{n_i \in \mathcal{N}} \alpha_{n_i} HPWL_{n_i} C_n V^2 f \tag{V.2}$$

$$P_{\mathcal{E}} = \sum_{e_i \in \mathcal{E}} \alpha_{e_i} L_{e_i} C_e V^2 f \tag{V.3}$$

Here, α_{n_i} and α_{e_i} are the respective signal-net and clock-edge activity factors, C_n and C_e are the respective unit capacitance for signal and clock wires, V is the supply voltage, f is the clock frequency, $HPWL_{n_i}$ is the HPWL of net n_i, and L_{e_i} is the Manhattan length of edge e_i. Activity factors of clock-net edges are required when multiple clock domains or gated clocks are utilized for given designs, otherwise $\alpha_{e_i} = 1$ as clock edges switch every clock cycle. The handling of gated clocks is discussed in Section 5.5 in more detail. If the activity factors of signal nets are not available, the computation of total switching power relies on *clock-power ratio* β, i.e., clock-net switching power divided by total switching power. In this case, the average activity factor of signal-net α_{avg} can be derived as

$$\alpha_{avg} = \frac{(1-\beta)\sum_{e_i \in \mathcal{E}} L_{e_i} C_e}{\beta \sum_{n_i \in \mathcal{N}} HPWL_{n_i} C_n} \tag{V.4}$$

α_{avg} is utilized for the activity factors of all the signal nets.

Compared to the work in [103], where the main objective does not capture the power of signal nets, our primary objective function (Formula V.1) captures both clock-net and signal-net switching power. The objective function in [20] considers clock-net and signal-net power, but their estimation of clock-net switching power relies on bounding boxes that cannot accurately represent clock-net wirelength. However, our analysis of clock-net power in Section 5.4.2 allows us to explicitly represent clock-net power in the primary objective. Thus, the optimization of our primary objective effectively decreases total switching power as described in Section 5.6.

5.4 Proposed techniques

We propose a methodology and several new techniques to overcome limitations of prior work and reliably optimize large IC designs with numerous layout obstacles. Our approach consists of two major phases: (*i*) virtual clock-tree synthesis, (*ii*) arboreal clock-net contraction force, which is corrected by an obstacle-avoidance force.

5.4.1 Obstacle-aware virtual clock trees

Our virtual clock-tree synthesis handles macro blocks as wiring obstacles and produces obstacle-avoiding clock trees. The importance of utilizing obstacle-aware clock trees is illustrated in Figure 5.3 (the contraction forces are described in Section 5.4.2). Clock-net optimizations without obstacle handling pull clock sinks inside obstacles, which undermines global placement.

Experimental results in [53] show that the difference in total capacitance between initial zero-skew DME trees (based on Elmore delay) and the final SPICE-optimized trees

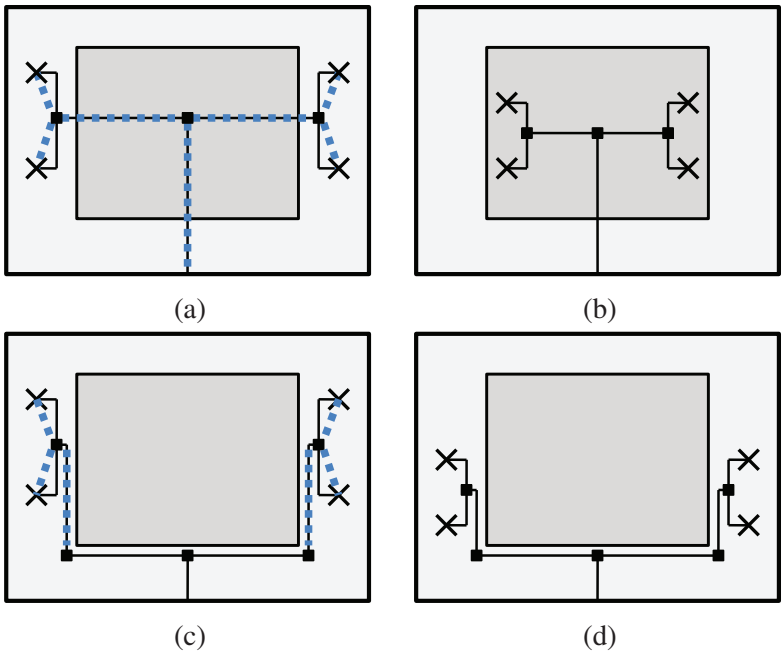


Figure 5.3: An example of clock-net optimization with an obstacle. (a) The virtual clock tree and corresponding contraction forces are created without considering the obstacle. (b) The result of a placement iteration with the forces in (a). (c) The obstacle is accounted during virtual clock-tree generation and when establishing additional forces. (d) The result of (c).

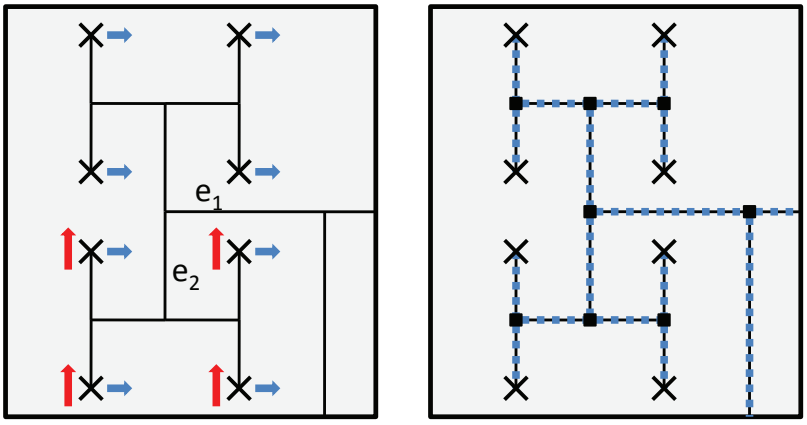
is only 2.2% on average. Hence, initial trees produced by leading clock-network synthesis tools offer reasonably accurate capacitance estimates. To quickly construct a *virtual clock-tree* during placement, our methodology first performs traditional DME-based zero-skew clock-tree synthesis with Elmore delay model, subject to obstacle avoidance. Several techniques are known for this problem, including direct obstacle-avoiding clock-tree construction [46] and incremental repair of obstacle-unaware trees [51]. Each approach can be used in our methodology, but we found that incremental-repair techniques are simpler and yet produce high-quality trees.[1] Our clock trees target the 45 nm technology used at the ISPD 2010 clock network synthesis contest [95].

[1]Extensive empirical studies and the experience of ISPD clock-network synthesis contests suggest that when clock sinks are placed outside the obstacles, the overlaps caused by obstacle-unaware trees can often be fixed with minimal impact on skew and total capacitance, compared to obstacle-aware trees.

5.4.2 Arboreal clock-net contraction force

If the virtual clock network connecting to current register locations faithfully represents a realistic clock network, then optimizing it directly should improve the final clock network produced by a specialized CTS tool after placement is complete. To this end, we extend force-directed placement with new, structurally-defined forces that seek to reduce individual edges of the virtual clock network. This technique communicates current clock-tree structure to the placement algorithm, and also allows the structure to change with placement.

Figure 5.4(a) illustrates a sample virtual clock tree. To reduce the length of e_1 directly, all sinks downstream from e_1 can be moved in the direction of reducing the length of e_1. For each downstream sink of e_1, a force vector needs to be assigned. The force vectors created for e_1 should not affect other tree edges.



(a) (b)

Figure 5.4: Two types of forces for clock-net optimization. Registers are indicated by crosses. (a) For each edge, the corresponding downstream registers are given force vectors. Right arrows are the force vectors for reducing e_1, and up arrows are the force vectors for reducing e_2. (b) Virtual nodes are inserted (squares), and forces are created between each pair of connected nodes (dotted lines).

The sum of magnitudes of force vectors induced by e_1 ($F_{e_i}^{sum}$) needs to be carefully controlled to avoid excessive increase in signal-net wirelength. $F_{e_i}^{sum}$ may vary when the activity factors of clock edges differ (e.g., in gated clocks). Figure 5.4(a) illustrates force vectors. The force from e_1 is weaker than the force from e_2, $F_{e_1} < F_{e_2}$ since the sum of magnitudes should be same.

The main problem with this method is that the relative locations of branching nodes from sinks are assumed to be same when the force vectors are created. However, optimal relative locations of the branching nodes change during the optimization. Therefore, placement iterations with fixed force vectors for sinks do not produce optimal locations.

To shorten clock wires, we propose *an arboreal clock-net contraction force with virtual-node insertion*. Our approach creates forces between clock-tree nodes and structurally transfer the forces down to registers. Virtual nodes represent branching nodes in the clock tree and split the clock tree into individual edges, seen as different nets by the placement algorithm. The virtual nodes have zero area and do not create overlap with real cells, so they do not affect the spreading process in force-directed placers. Zero-area nodes may or may not be allowed to overlap with obstacles (if such a node is placed over an obstacle, its overlap has zero area). In our case, virtual nodes should not be placed over obstacles to avoid routing over obstacles.

Compared to the fixed force vectors applied exclusively to sinks, our technique creates forces between flexible nodes and each force seeks to reduce the length of the corresponding clock edge. Unlike in the bounding-box based method, each force is integrated into the placement instance as a two-pin pseudo net, as shown in Figure 5.4(b).

To reduce dynamic power consumption of the IC, contraction forces are calculated based on the activity factors of the signal nets. When activity factors of signal nets are available, the average activity factor α_{avg} over all nets is

$$\alpha_{avg} = \frac{\sum_{n_i \in \mathcal{N}} \alpha_{n_i} HPWL_{n_i}}{\sum_{n_i \in \mathcal{N}} HPWL_{n_i}} \tag{V.5}$$

and the weight of signal net n_i is defined as

$$w_{n_i} = \frac{\alpha_{n_i}}{\alpha_{avg}} \tag{V.6}$$

When activity factors of signal nets are not available, Equation V.4 is utilized to compute α_{avg} and $w_{n_i} = 1$ for all signal nets. A two-pin net representing clock-net contraction forces for clock edge e_i is given a weight

$$w_{e_i} = \frac{C_e \alpha_{e_i}}{C_n \alpha_{avg}} \tag{V.7}$$

and the HPWL of a two-pin net from e_i is equal to the Manhattan length of e_i,

$$L_{e_i} = HPWL_{e_i} \tag{V.8}$$

Note that our primary objective function is a sum of signal-net and clock-net switching power (Formula V.1). By combining Formulas V.2, V.3 and V.8, the total switching power is expressed as

$$\Big(\sum_{n_i \in \mathcal{N}} \alpha_{n_i} HPWL_{n_i} C_n + \sum_{e_i \in \mathcal{E}} \alpha_{e_i} HPWL_{e_i} C_e \Big) V^2 f \tag{V.9}$$

By substituting α_{n_i} and α_{e_i} in terms of w_{n_i} and w_{e_i} (Equations V.5, V.7), Equation V.9 can be rewritten as

$$\alpha_{avg} \Big(\sum_{n_i \in \mathcal{N}} w_{n_i} HPWL_{n_i} C_n + \sum_{e_i \in \mathcal{E}} w_{e_i} HPWL_{e_i} C_n \Big) V^2 f \tag{V.10}$$

Let K be $\alpha_{avg} C_n V^2 f$, $\mathcal{M} = \mathcal{N} \cup \mathcal{E}$. Then the total switching power of signal nets and current clock nets is,

$$P_{sw} = P_{\mathcal{N}} + P_{\mathcal{E}} = K \sum_{m_i \in \mathcal{M}} w_{m_i} HPWL_{m_i} \tag{V.11}$$

In other words, our techniques capture the switching-power minimization problem, which can be solved by any high-quality wirelength-driven placer capable of net weighting. Figure 5.5 compares our technique and MLAF from [103]. MLAF is ineffective in shortening clock nets that significantly differ from H-trees. Additionally, MLAF does not establish exclusive forces that represent the edges between parents and children nodes. Instead, bounding boxes (MLBB) are used with MLAF in [103]. We show in Section 5.2 that bounding boxes cannot offer accurate estimates of clock-tree lengths. Also, the authors of [103] did not explain how they assigned weights to MLAF but only hinted that the magnitude of MLAF was similar to the magnitude of forces for signal nets. Detailed comparison between our technique and MLAF is discussed in Section 5.6.2 (Table 5.5).

5.4.3 Obstacle-avoidance force

Given an obstacle-avoiding tree, we modify arboreal clock-net contraction forces to promote obstacle avoidance. Contraction forces based on an obstacle-avoiding clock tree do not necessarily improve every tree edge, as shown in Figure 6. In Figure 5.6(a), five edges are derived from a virtual obstacle-aware tree built as in Section 5.4.1. If we create forces for all the edges, subsequent optimization will produce the tree in Figure 5.6(b). The force f_4 associated with edge e_4 is rendered ineffective by the obstacle. Our force-modification algorithm for obstacle avoidance detects these obstacle-detouring edges and

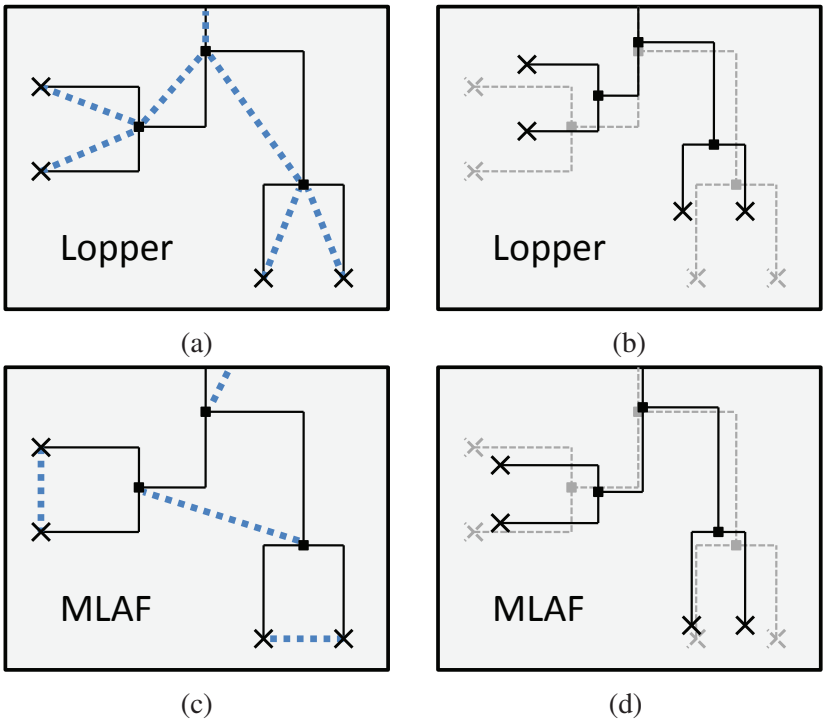


Figure 5.5: Comparison between our arboreal clock-net contraction force and MLAF of [103]. (a) Arboreal clock-net contraction forces are generated. (b) The modified register and virtual clock-node locations when forces in (a) are utilized. (c) The forces created by the MLAF algorithm. (d) The modified register and virtual clock-node locations when forces in (c) are utilized. We can observe that the edges between parents and children nodes are poorly handled for the force creation in (c), and our method is more efficient on non H-tree structures (which is common in modern designs).

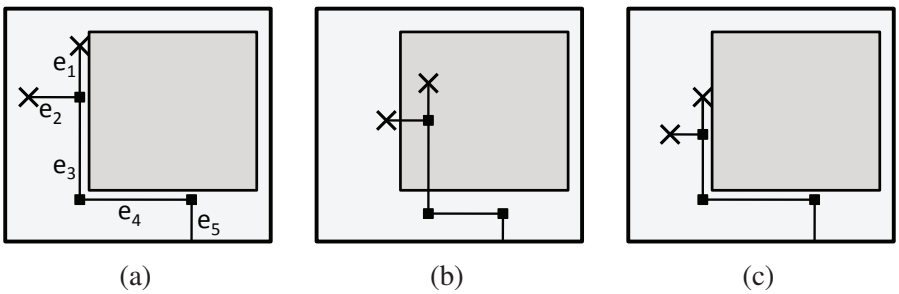


Figure 5.6: Obstacle-avoidance force. (a) Five edges of an obstacle-aware virtual clock tree. (b) The result when all the edges are utilized for contraction forces. (c) The result when e_4 and e_5 are excluded from force construction.

eliminates the contraction forces for them.[2] In this example, e_4 and e_5 are excluded from force construction, and the result is illustrated in Figure 5.6(c).

5.5 Proposed methodology

We integrate our techniques into SimPL, a flat, force-directed quadratic placer [47]. Recall that analytic placers first minimize a function of interconnect length, neglecting overlaps between standard cells and macros. This initial step places many cells in densely populated regions. Clock-net contraction forces are ineffective at this step for two reasons: (*i*) the current virtual clock network may differ greatly from the final clock network, (*ii*) the contraction forces may restrict the spreading of the registers at the center of the design due to their high net weight. Therefore, our techniques are invoked between signal-net wirelength-driven global placement and detailed placement. Our clock-net optimization during placement is referred to as *Lopper*, and described in Figure 5.7.

5.5.1 The Lopper flow

At each iteration of Lopper, a new virtual clock tree is generated based on current register locations. We discard the previous virtual clock tree based on the following observation. The topology of a clock tree and the embedding of its wires minimize (*i*) skew as the primary objective, (*ii*) total wirelength as the secondary objective. When an iteration of Lopper is performed, the locations of the registers are modified in order to reduce the total wirelength of the given virtual clock tree. Since registers are displaced by different amounts (due to different connectivities), keeping the previous clock-tree structure would

[2]Consider a clock-tree edge that does not cross a given obstacle. The edge *detours* the obstacle if the straight line connecting the ends of the edge crosses the obstacle.

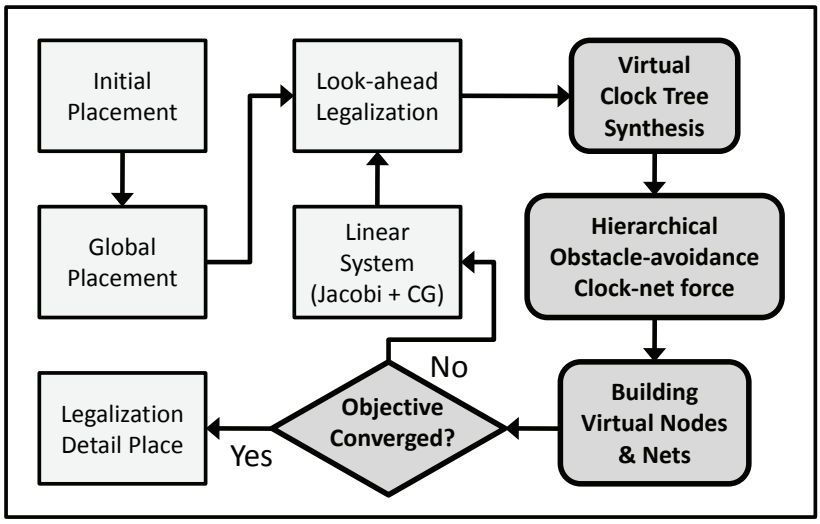


Figure 5.7: Key steps of Lopper integrated into the SimPL placer, as indicated with darker rounded boxes and a lozenge. Plain boxes represent the SimPL steps.

risk a large increase in skew. Therefore we regenerate the virtual clock tree for each iteration to obtain an optimal virtual clock tree with the current register locations. The tree topology typically undergoes only moderate changes, while branching nodes relocate to reduce skew.

Early placement iterations may greatly displace the registers, suggesting that effective clock-net wirelength reduction requires moving registers over the obstacles. In this case, obstacle-aware virtual CTS and obstacle-avoidance force may undermine the potential improvement. Therefore, Lopper ignores obstacles until average displacement of registers becomes small.

Global placement typically continues while HPWL continues improving, but clock-tree reduction in Lopper requires a different convergence criterion. After each iteration, total switching power is calculated and compared to previous values. Lopper is invoked repeatedly until total switching power (Equation V.1) stops reducing.

Legalization and detailed placement are applied after Lopper is complete. It is important to preserve the virtual nodes and two-pin nets that represent the clock-net contraction forces during detailed placement because detailed placement algorithms usually optimize wirelength and would not have preserved clock-optimized register locations if guided only by signal nets.

5.5.2 Trade-offs and additional features

Quality control. Our techniques reduce the size of clock networks, but are likely to increase signal-net wirelength. The activity factor of each signal-net α_{n_i} or clock-power ratio β are required for Lopper to reduce total switching power. However, even clock-power ratio β is hard to estimate before the design is completed and can vary with various applications running on a CPU. Therefore, in our implementation the trade-off between clock-net and signal-net switching power can be easily controlled with a single parameter β. This simple quality control allows an IC designer to achieve intended total switching power of a chip without changing the algorithm or its internal parameters. Relevant trade-offs are illustrated in Table 5.3.

Gated clocks and multiple clock domains are well-known and often the most effective techniques to reduce clock network power dissipation [73]. To extend our techniques to gated clocks and multiple clock domains, each register s_i is given an activity factor α_{s_i} and the activity factors are propagated through the tree. The activity factor of an edge is the highest activity factor of its child edge or register (see Figure 5.8). Without clock gating, all registers are given activity factors 1.0, which are propagated to all tree edges.

Once activity factors are propagated to tree edges in each clock tree, they are used to

calculate net weights that represent clock-net contraction forces in Equation V.7. Registers that switch less frequently due to clock gating will be more affected by signal nets than normal registers without clock gating. Our technique does not track the locations of gaters assuming that the final clock tree and the gaters are constructed after register placement. While we have not experimented with gater placement, we do not believe that it will affect results reported in our work.

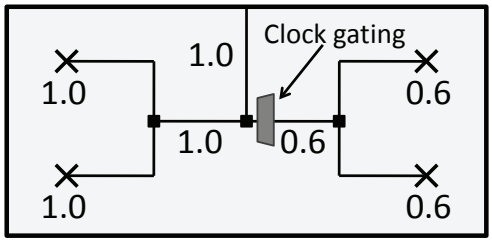


Figure 5.8: Activity-factor propagation for gated clocks. Registers are indicated with crosses. Tree edges and registers are labeled with activity factors.

Flexible integration. Through the Lopper flow, forces for clock-net optimization are represented in placement instances by virtual nodes and nets. No support for clock-net optimization is required in the placement algorithm. Therefore, Lopper can integrate any fast obstacle-aware clock-tree synthesis technique into any iterative high-performance wirelength-driven placer capable of net weighting.

Integration into timing-driven placement. Timing-driven placement optimizes cell locations to satisfy timing constraints, while minimizing interconnect [43, Chapter 8]. During timing-driven placement, delays of timing-critical nets are carefully controlled to prevent timing violations. Lopper can be integrated into timing-driven placers thanks to the flexibility of proposed integration. For example, a common approach to timing-driven

placement increases weights of critical nets.[3] To this end, Formula V.6 can be extended to include the *criticality* c_{n_i} of net n_i which represents how important this net is to satisfying timing constraints.

$$w_{n_i} = \frac{\alpha_{n_i} c_{n_i}}{\alpha_{avg}} \tag{V.12}$$

If weights of critical nets exceed clock-net weights (i.e., $w_{n_i} > w_{e_i}$), the wirelength of critical nets is affected less when reducing clock-net wirelength. During the Lopper flow, only the wirelengths of less critical nets increase to reduce clock-net switching power. Therefore, optimization for total switching-power reduction can be performed without violating timing constraints.

High-quality register placement for robustness to variations. In practice, not only low nominal skew but also robustness to PVT variations are essential for building high-quality clock networks. When making a clock network more robust, one uses large buffers and/or redundant wiring, which increases total capacitance and dynamic power. Because clock networks consume a large portion of total power, it is important to limit the maximum clock-network power. *When clock-network power is limited by design, one of the most effective techniques to improve robustness to PVT variations is to optimize register placement.* Since our virtual clock-tree construction algorithm is variation sensitive[4] and integrated into our primary optimization objective during placement, our register placement is sensitive to PVT variations. Extensive SPICE simulations confirm that the Lopper

[3] This approach is used in our work to illustrate the compatibility of Lopper with timing-driven placement. Many other approaches exist [43, Chapter 8].

[4] When clock-network power is limited, the DME-based initial-tree construction algorithm in [53] is variation sensitive in the sense that it generate low-skew trees with optimizing wirelength. When initial wirelength of a clock tree is small, more optimizations for enhancing robustness are possible within given power limit. Our virtual clock-tree construction algorithm is adapted from this variation-sensitive initial-tree construction algorithm.

flow significantly improves robustness of clock trees when clock-network power is limited (see Section 5.6.3).

5.6 Empirical validation: Lopper

The benchmarks used in prior publications on clock-tree optimization during placement exhibit the following problems: (1) Empirical validation of each existing publication relies on one benchmark suite which is not utilized by any other work. Most of the benchmarks are inaccessible to public, therefore comparisons to new techniques are impossible. (2) The benchmark designs are based on unrealistically small placement instances. None of the prior publications provide results on a design with more than 1M standard cells, which is common in modern modern ASIC designs. (3) Macro blocks became essential components, and many IC designs include more than hundreds of macros with fixed locations after floorplanning [6,43]. However, prior publications used the benchmarks without macro blocks or ignored macro blocks present in the benchmarks [103]. (4) Reference placement tools used for comparison are often outdated [64] or self-implemented [103]. Such comparisons risk not being representative of state-of-the-art EDA tools.

In this section, we propose a new benchmark set that addresses the above pitfalls. Our experimental results offer full comparisons with leading academic wirelength-driven placers and a known technique for register placement (MLAF). The quality of register locations is validated by a leading academic clock-network synthesis tool.

5.6.1 Experimental setup

The ISPD 2005 placement contest benchmark suite is being used extensively in placement research, and the academic community consistently advanced physical design techniques using the ISPD'05 benchmarks. These benchmarks are directly derived from industrial ASIC designs, with circuit sizes ranging from 210K to 2.1M placeable objects. We adapted eight designs from the ISPD'05 benchmarks and created register lists in which 15% of standard cells are selected to be registers. We selected the number 15% based on the industrial designs introduced in [20], where the average 14.65% of cells are registers. The largest benchmark has 327K registers. Fixed macro blocks are viewed as routing and placement blockages during clock-network synthesis.[5] Some macro blocks that create routing dead space are slightly resized and/or repositioned to eliminate dead space (see Figure 5.9). This modification is so small that the impact on density and timing is negligible. The benchmarks are mapped to the Nangate 45 nm open cell library [70] to facilitate clock-network synthesis with parameters from ISPD 2010 CNS contest. The standard-cell height (or row height) is set to 1.4 μm according to the 45 nm library.[6] Clock-power ratio β is set to 0.3 for clock network optimization during placement based on the industrial circuits from [20], where clock power is responsible for 31.9% of total power on average. For each circuit, the average activity factor of signal nets is calculated based on the signal-net and clock-net wirelength of the placement produced by SimPL [47] using Formula V.4. The unit-wire capacitances for signal-net and clock-net (C_n, C_e) are set to 0.1 $fF/\mu m$, 0.2

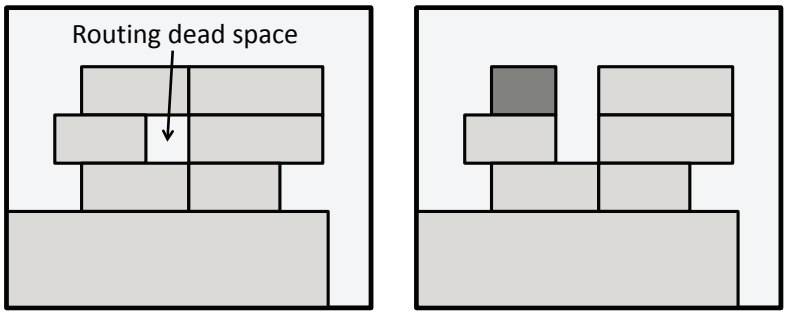
[5] When macro blocks act as placement blockages but routing is allowed above them, the load-capacitance-aware obstacle-avoidance algorithm in [51] can be utilized to detour the clock-tree wires that (i) cross macro blocks, (ii) but cannot be driven by the buffers outside macro blocks.

[6] Unit length in the ISPD'05 benchmark corresponds to approximately 117 nm in our benchmark set.

Name	Cells	Regs	Macros	CoreX (mm)	CoreY (mm)	Area (mm^2)
clkad1	210K	32K	56	1.247	1.246	1.554
clkad2	255K	38K	177	1.640	1.638	2.686
clkad3	451K	68K	721	2.706	2.722	7.363
clkad4	494K	74K	1329	2.706	2.722	7.363
clkbb1	278K	42K	30	1.247	1.246	1.554
clkbb2	535K	84K	923	2.181	2.192	4.781
clkbb3	1095K	165K	666	3.231	3.242	10.47
clkbb4	2169K	327K	639	3.756	3.772	14.16

Table 5.1: The new CLKISPD'05 benchmarks.

$fF/\mu m$ respectively based on the 45 nm technology model from the ISPD'10 contest [95] and the Nangate open-cell library [70]. Supply voltage and clock frequency are set to $1.0V$ and $2GHz$. The coordinate of clock source is set to the bottom left corner of core area except when it is blocked by macros. When the desired location is blocked, we move the clock source to the closest unblocked coordinate. Since many academic placers handle the ISPD'05 benchmarks, a direct comparison of clock-network quality and signal-net wirelength is possible. The new benchmarks (referred to as *CLKISPD'05*, downloadable from http://vlsicad.eecs.umich.edu/BK/CLKISPD05bench [57]) are described in Table 5.1.



(a) (b)

Figure 5.9: An example of routing dead space that can be found in the ISPD'05 benchmarks. (a) Routing dead space is created by enclosing macro blocks. (b) One macro block is modified to open the space.

		FastPlace3			mPL6			SimPL 101			SimPL+Lopper			
Bench	α_{avg}	ClkWL (mm)	HPWL (m)	Pwr (mW)	ClkWL (mm)	HPWL (m)	Pwr (mW)	ClkWL (mm)	HPWL (m)	Pwr (mW)	ClkWL (mm)	HPWL (m)	Pwr (mW)	⏲ (min)
clkad1	0.11	214.7	9.12	285.5	248.2	9.09	298.3	209.1	8.97	279.9	**152.3**	9.23	**263.0**	3.6
clkad2	0.10	236.2	10.9	310.1	267.0	10.7	318.9	223.1	10.5	297.6	**161.0**	10.8	**278.4**	5.5
clkad3	0.09	469.3	25.0	640.8	467.6	25.0	640.8	468.5	24.0	624.7	**326.9**	24.9	**583.0**	11
clkad4	0.11	540.9	23.1	732.9	615.6	22.6	751.6	519.4	21.7	692.6	**354.4**	22.3	**640.4**	12
clkbb1	0.10	250.5	11.2	323.6	245.1	11.3	322.5	238.2	11.2	317.6	**166.3**	11.5	**295.7**	5.2
clkbb2	0.15	539.2	18.1	752.6	514.1	17.8	733.6	533.2	16.8	710.9	**371.2**	17.3	**661.4**	13
clkbb3	0.10	892.6	42.7	1236	1032	40.2	1240	866.3	39.2	1155	**602.2**	41.0	**1085**	30
clkbb4	0.09	1907	97.3	2575	2119	96.8	2650	1855	93.0	2473	**1266**	95.2	**2279**	87
Avg		1.03×	1.05×	1.04×	1.11×	1.03×	1.06×	1.00×	1.00×	1.00×	**0.70**×	1.03×	**0.93**×	1.8×

Table 5.2: Results on the CLKISPD'05 benchmark suite. ClkWL represents total wirelength of a clock network synthesized by the initial phase of Contango 2.0 [53]. HPWL is total HPWL of signal nets. Pwr is total net-switching power. SimPL+Lopper is 4.16× faster than mPL6 and 1.51×, 1.81× slower than FastPlace3, SimPL respectively.

The quality of clock networks based on the final register locations of each placer is evaluated by Contango 2.0 [53]. Contango 2.0 is the winner of the ISPD 2009 and 2010 Clock Network Synthesis (CNS) contests and produces clock trees with less than 7.5 ps skew in the presence of variation on the ISPD'10 CNS benchmarks. During our experiments in Section 5.6.2 , we exclude SPICE-accurate tuning in Contango 2.0 for two reasons: (1) the designs from the ISPD'05 benchmarks are too large to run SPICE simulations, (2) the average added capacitance during the SPICE-driven optimization on the ISPD'10 CNS benchmarks is 2.2% of total clock-net capacitance (including sink, wire and buffer capacitance), suggesting that the initial trees optimized for Elmore delay provide good estimates of power consumption. In Section 5.6.3, we present experimental results for clock trees with SPICE-driven optimizations on the modified benchmark set. Insertion delay and skew from SPICE simulations are reported as well as total capacitance of optimized clock trees with driving buffers. We also present robustness analysis of different register placements when total capacitance is limited for clock networks in the presence of variations.

5.6.2 Empirical results

Table 5.2 compares results of our methodology to the leading academic placers on the CLKISPD'05 benchmarks. The results of SimPL [47] are used as reference for comparison. α_{avg} is computed for each benchmark based on the given $\beta = 0.3$ as described in Section 5.6.1, and total wire-switching power is calculated based on α_{avg}. Power consumed inside macro blocks is ignored since it cannot be optimized during placement and is not available in original ISPD benchmark data. On average, the combination of SimPL and Lopper reduces total clock-tree length by 30.0%, total wire-switching power by 6.8% while the total HPWL of the signal nets only increases by 3.1% compared to SimPL. Compared to FastPlace3 [100] and mPL6 [15], our methodology reduces the total clock-net wirelength by 32.1%, 36.6%, total wire-switching power by 10.5%, 11.6% respectively, while the total signal-net HPWL is smaller than that produced by FastPlace3 by 1.4% and very similar to that produced by mPL6. Our methodology shows consistent improvement for the benchmarks considered, with various configurations of macro blocks. Figure 5.10 compares two clock trees based on register placements from SimPL and our method.

To further study the relative significance of clock-power ratio β, we show in Table 5.3 the impact of varying β on the benchmark *clkad1*. The average activity factor of signal nets α_{avg} is computed based on the reference layout and utilized for computing the total wire-switching power. The performance of Lopper is improved when clock networks consume a greater portion of total power. Table 5.3 also shows that reducing clock networks does not necessarily reduce the total switching power. For example, the result for $\beta = 0.6$ consumes 109.6 mW for total wire-switching power, but if the same circuit is used for the

β	α_{avg}	Orig. P (mW)	ClkWL (mm)	HPWL (m)	Pwr	
					(mW)	(Rel)
Orig	-	-	209.1	8.968	-	-
0.1	0.420	837.0	184.2	9.073	**835.8**	**0.999**
0.15	0.264	557.2	173.5	9.128	**551.3**	**0.990**
0.2	0.187	419.1	165.7	9.188	**409.9**	**0.978**
0.25	0.140	334.8	158.0	9.225	**321.5**	**0.960**
0.3	0.109	279.9	152.3	9.233	**262.2**	**0.939**
0.35	0.087	239.7	151.0	9.280	**221.9**	**0.925**
0.4	0.070	209.2	144.8	9.305	**188.2**	**0.900**
0.45	0.057	185.9	144.5	9.316	**164.0**	**0.882**
0.5	0.047	168.0	139.5	9.342	**143.6**	**0.854**
0.55	0.038	151.8	135.7	9.343	**125.3**	**0.826**
0.6	0.031	139.3	128.0	9.425	**109.6**	**0.787**

Table 5.3: The results on *clkad1* with various clock power ratios β. The specifications of the reference placement produced by SimPL are in the row *Orig*. α_{avg} is calculated based on β and reference placement produced by SimPL. Total wire-switching power values of the reference placement with the corresponding β are represented in the column *Orig. P*. The relative power ratios are indicated with *Rel*.

applications with $\beta = 0.1$, the total wire-switching power computed by Equations V.1 - V.3 is 842.9 mW, which is greater than the switching power of the reference placement 837.0 mW. This implies that clock-net optimization must utilize activity factors of signal nets or clock-power ratios to reduce total switching power.

Table 5.4 shows the impact of obstacle-aware virtual clock trees (OAVCT) and obstacle avoidance forces (OAF). When OAVCT is excluded, DME trees without obstacle handling are utilized for the remaining flow. The results indicate that 9.5% of clock-net wirelength can be reduced on average by utilizing obstacle-aware trees. The advantage of OAVCT is reduced on benchmarks with very few obstacles such as *clkbb1* where a few obstacles exist at the top left corner of the chip. Obstacle-avoidance forces reduce clock-net length by 4.1% and total switching power by 0.7%.

Table 5.5 compares results of our technique to the technique called MLAF on MLBB [103]. We re-implemented their MLAF algorithm and integrated it into the SimPL placer

[47] instead of the FDP framework [101] they utilized. Since their DCTB algorithm cannot process obstacles, our obstacle-aware virtual clock-tree generation algorithm in Section 5.4.1 is utilized for the MLAF algorithm. In terms of clock-net wirelength and net-switching power, the average gain from the MLAF technique is limited by 43.5%, 30.6% of the improvement of our technique respectively, which means that our arboreal clock-net contraction force is 3.3× more effective for switching-power reduction than MLAF. Our comparison to MLAF concludes that explicit and structural representation of clock-net force and an accurate weighting function are important to achieve competitive register placement.

	Orig. Flow		w/o OAVCT		w/o OAF	
Bench	ClkWL (mm)	Pwr (mW)	ClkWL (mm)	Pwr (mW)	ClkWL (mm)	Pwr (mW)
clkad1	152.3	263.0	165.7	267.8	158.5	265.3
clkad2	161.0	278.4	170.9	285.5	163.7	278.7
clkad3	326.9	583.0	362.1	595.1	340.8	587.4
clkad4	354.4	640.4	403.1	657.2	379.8	649.4
clkbb1	166.3	295.7	172.6	297.4	169.1	296.4
clkbb2	371.2	661.4	411.2	673.8	389.9	666.7
clkbb3	602.2	1085	663.1	1104	627.2	1093
clkbb4	1266	2279	1412	2331	1328	2102
Avg	**1.0**×	**1.0**×	+**9.5**%	+**1.8**%	+**4.1**%	+**0.7**%

Table 5.4: Impact of excluding obstacle-aware virtual clock trees (OAVCT), obstacle avoidance forces (OAF). OAVCT and OAF are excluded in the columns under "w/o OAVCT". Only OAF is removed in "w/o OAF"

5.6.3 SPICE-driven validation

Insertion delay and skew are important metrics when evaluating the quality of clock trees. Accurate analysis of these metrics requires SPICE simulations. The CLKISPD'05 benchmark set has up to 327K registers in one benchmark and it is impractical to perform SPICE-driven optimizations introduced in [53]. To construct high-quality SPICE-accurate clock trees, we decrease the number of registers in the CLKISPD'05 benchmarks

	SIMPL+MLAF		
Bench	ClkWL (mm)	HPWL (m)	Pwr (mW)
clkad1	182.4 (46.9%)	9.194 (85.3%)	274.2 (33.7%)
clkad2	200.9 (35.8%)	10.76 (76.2%)	293.0 (24.0%)
clkad3	402.5 (46.6%)	24.71 (76.9%)	609.8 (35.7%)
clkad4	449.5 (42.4%)	22.24 (86.9%)	676.6 (30.7%)
clkbb1	203.8 (47.9%)	11.48 (84.9%)	309.7 (36.1%)
clkbb2	473.8 (36.7%)	17.16 (80.0%)	699.3 (23.4%)
clkbb3	743.5 (46.5%)	40.81 (91.0%)	1139 (22.9%)
clkbb4	1587 (45.5%)	94.77 (80.2%)	2399 (38.1%)
Avg	**0.87× (43.5%)**	**1.03× (82.7%)**	**0.98× (30.6%)**

Table 5.5: Results of the MLAF technique integrated into SimPL with comparison to our technique. Average results are compared to the results for SimPL in Table 5.2. The numbers in parentheses represent the amount of reduction(ClkWL, Pwr) [increase(HPWL)] assuming 100% reduction [increase] for our technique. For example, [209.1(SimPL) - 182.4(MLAF)] / [209.1(SimPL) - 152.3(Lopper)] = 46.9%.

		FASTPLACE3			MPL6			SIMPL 101			SIMPL+LOPPER		
Bench	Regs	Ins. D. (ps)	Skew (ps)	Cap. (pF)	Ins. D. (ps)	Skew (ps)	Cap (pF)	Ins. D. (ps)	Skew (ps)	Cap. (pF)	Ins. D. (ps)	Skew (ps)	Cap. (pF)
clkad1_s	2114	386.3	2.702	29.90	388.2	2.629	34.09	381.7	3.730	26.61	369.0	0.962	**16.70**
clkad2_s	2550	406.8	3.329	35.95	414.2	2.471	36.24	405.8	3.152	31.52	402.9	2.163	**20.53**
clkad3_s	4516	453.7	3.921	68.08	468.0	3.642	77.36	453.4	3.770	65.95	452.3	2.208	**45.00**
clkad4_s	4960	455.4	3.502	77.84	470.0	2.065	86.28	454.8	3.568	71.66	446.6	2.216	**44.44**
clkbb1_s	2781	385.0	2.711	29.86	387.0	2.139	35.61	386.1	5.166	29.10	385.9	1.754	**18.35**
clkbb2_s	5578	445.1	8.153	78.34	444.4	5.984	83.77	444.4	1.876	75.18	431.4	2.537	**47.18**
clkbb3_s	10968	489.8	3.118	125.7	513.0	7.140	150.2	497.6	2.796	120.7	494.2	2.286	**77.75**
clkbb4_s	21773	523.9	2.318	268.6	522.7	2.956	284.0	523.5	3.511	250.8	510.9	2.852	**156.0**
Avg		1.02×	1.83×	1.68×	1.03×	1.77×	1.87×	1.02×	1.85×	1.57×	1.00×	1.00×	**1.00×**

Table 5.6: Results of SPICE-driven optimizations on the modified CLKISPD'05 benchmark suite. Regs represents the number of registers in each benchmark. Ins. D. is insertion delay and Skew is nominal local skew defined in [53] with local skew distance limit 600μm. Cap. represents total capacitance of the clock tree including driving buffers.

and invoke Contango 2.0 on various register placements. The experimental results with SPICE-driven optimizations are described in Table 5.6.

We adapted wire and buffer libraries from the ISPD 2010 CNS benchmarks. The concept of local skew defined in [53] is utilized to calculate exact skew with the local skew distance limit 600μm. Unlike Table 5.2 that reports dynamic power based on only wire capacitance, the total capacitance including driving buffers is reported in Table 5.6. Same

buffering scheme is utilized for all clock trees in this table. The Lopper flow produces high-quality register placement and it is already shown in Table 5.2 that the clock-network wirelength is significantly smaller than other methods. Due to compact clock-net wirelength, fewer driving buffers are required for clock-tree synthesis on our register placement, hence the total clock-network capacitance including buffers is 57%$\sim$87% less than other register placements.

Table 5.6 also shows that insertion delay based on our method is 1.6%$\sim$3.3% smaller than other methods. Since insertion delay depends on the path length from the clock source to sinks, the size of layout area is closely related to insertion delay. As shown in Figure 5.10, the wirelength of clock trees is reduced mainly near leaves, hence the improvement of insertion delay is not proportional to power improvement.[7] In practice, clock trees spend a lot of power near the leaves. The results show that Contango 2.0 can reduce nominal skew down to 10 ps for any register placement. The quality of nominal skew is not highly related to register placement but it depends on optimization performed during CNS. However, compact clock trees are easier to tune therefore average nominal skew reported in Table 5.6 is 77%$\sim$87% smaller on our design.

Table 5.7 shows how the quality of register placement affects robustness of clock networks in the presence of variations. Since robustness analysis requires extensive SPICE simulations, we rebuilt the CLKISPD'05 benchmarks with fewer registers, so that we can run hundreds of Monte-Carlo SPICE simulations. We imposed a capacitance limit when running Contango 2.0, and clock trees are optimized to be as robust as possible within this

[7]When the clock source is at a corner of chip area, often 30%$\sim$50% of insertion delay is due to the tree trunk (the wire that connects the clock source and the root node of the clock tree [51]) and the length of the trunk is largely unaffected by register placement *in practice*.

			SIMPL 101			SIMPL+LOPPER		
Bench	Regs	Cap. (pF)	Nom. (ps)	Mean (ps)	Yield (%)	Nom. (ps)	Mean (ps)	Yield (%)
clkad1_v	1057	23	1.855	6.634	72.4	2.177	4.298	**99.2**
clkad2_v	1275	28	1.113	5.771	**90.4**	3.265	5.291	89.8
clkad3_v	1354	46	1.132	7.281	60.4	2.673	6.615	**72.4**
clkad4_v	1488	50	2.010	8.007	43.2	1.610	6.094	**82.0**
clkbb1_v	1390	26	1.014	6.342	78.8	0.994	5.104	**94.6**
clkbb2_v	1115	39	2.208	7.674	49.0	1.338	5.912	**86.6**
clkbb3_v	1096	49	1.532	7.246	58.4	1.379	6.071	**82.4**
clkbb4_v	1088	67	5.226	9.607	15.4	4.207	7.342	**57.8**
Avg			2.011	7.320	58.5	2.205	5.841	**83.1**

Table 5.7: Results of SPICE simulations in the presence of variations. Regs represents the number of registers in each benchmark. Cap. represents the capacitance limit for clock networks. Nom. represents nominal skew without variation and Mean is average skew with variation. Yield represents the percentage of acceptable results with given skew limit 7.5 ps.

limit. After building clock trees, we first measure the nominal skew of clock trees without variations. Then we run extensive Monte-Carlo simulations with variations to estimate the impact of variations on clock-tree circuits. The variation model from the ISPD 2010 CNS benchmarks is utilized in the experiments.

Our register placement leads to more compact clock trees than other methods, and robustness is further enhanced by Contango 2.0. The results show that the clock trees based on our techniques offer 24.6% greater yield than the clock trees based on simPL alone when the skew limit is set to 7.5 ps.

5.7 Summary

Despite the increasing significance of power optimization in VLSI, state-of-the-art placement algorithms only optimize signal-net switching power and ignore clock-network switching responsible for over 30% of total power. We propose new techniques and a methodology to optimize total dynamic power during placement for large IC designs with

macro blocks. To this end, we advocate obstacle-aware virtual clock-tree synthesis, an arboreal clock-net contraction force with virtual nodes that can handle gated clocks, and an obstacle-avoidance force for clock edges. Our methodology is integrated into the SimPL placer [47], and the total switching power is measured by utilizing Contango 2.0 [53] — both programs are leading academic software. A new set of 45 nm benchmarks is proposed to better represent modern IC designs. Experimental results show that our method lowers the overall dynamic power by significantly reducing clock-net switching power. Other benefits of our optimizations include smaller insertion delay in clock trees, diminished sensitivity to process variations, and reduced supply voltage noise.

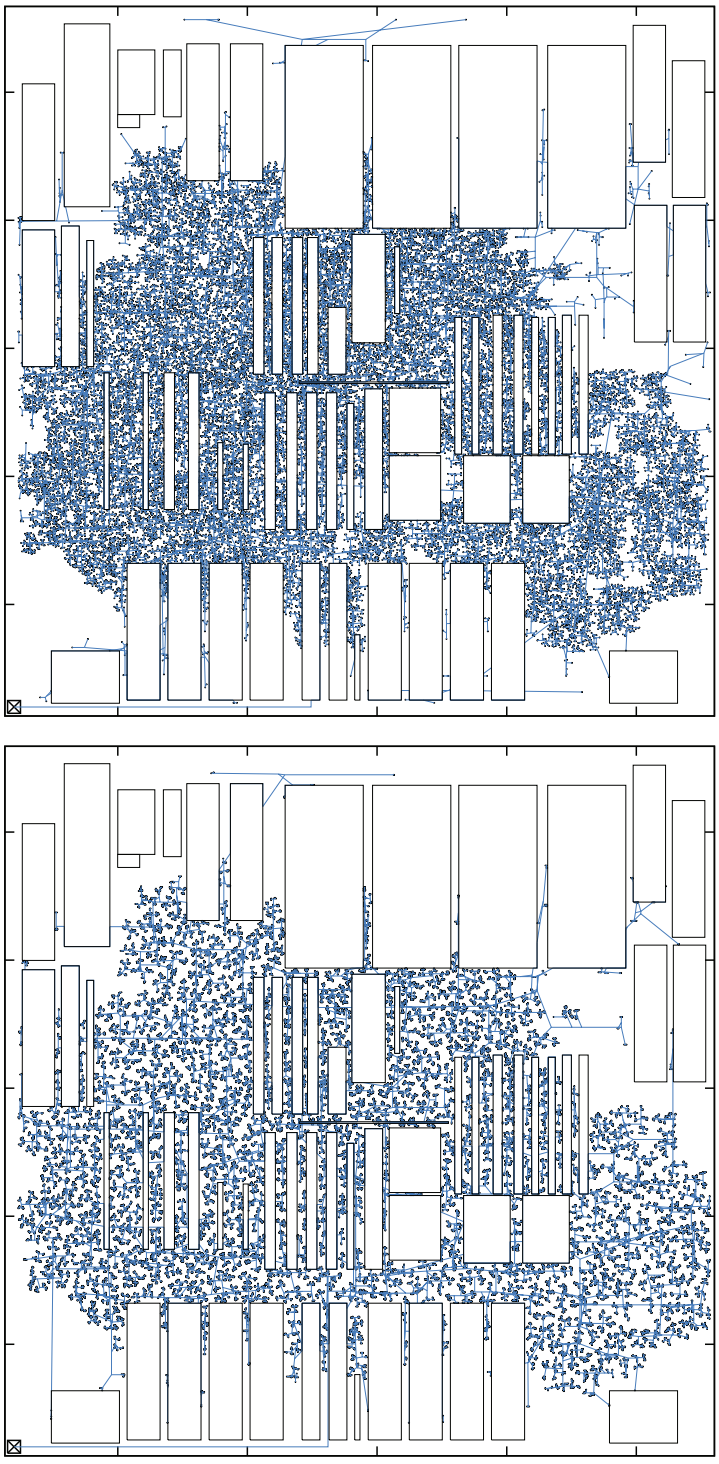


Figure 5.10: Clock trees for clkad1, based on a SimPL register placement (top) and produced by proposed techniques (bottom). The respective clock-tree wirelengths based on SimPL and our method are 209.13 mm and 152.27 mm. The total switching power of SimPL and our method are 279.9 mW and 263.0 mW respectively.

CHAPTER VI

Multilevel Tree Fusion for Robust Clock Networks

Recent improvements in clock-tree and mesh-based topologies maintain an on-going competition between the two. Trees require much smaller capacitance, but meshes are naturally robust against process variation and can accommodate late design changes. Cross-link insertion has been advocated to make trees more robust, but is limited in practice to short distances. In this chapter we develop a novel non-tree topology that fuses several clock trees to create large-scale redundancy in a clock network. Empirical validation shows that this clock-network structure incrementally enhances robustness to satisfy given variation constraints. Our implementation called Contango 3.0 produces robust clock networks even for challenging skew limits, without parallel buffering used by other implementations. It also offers a fine trade-off between power and robustness, increasing the capacitance of the initial tree by less than 60%, which results in 2.3× greater power efficiency than mesh structures.

6.1 Variation modeling for buffered paths

In this section, we develop statistical models for delay and skew in RC-buffered clock networks, including proposed clock-network topologies.

6.1.1 Impact of variation on delay

In the presence of PVT variations, the delay of a buffered path p can be treated as a random variable D_p whose mean d_p is the nominal delay. Given that tree-like clock networks entail long paths without significant reconvergence, path delay can be modeled by Gaussian variables:[1]

$$D_p = N(d_p, \sigma_p^2) \tag{VI.1}$$

The delays of serially connected paths p_1 and p_2 add up.

$$D_{p_1p_2} = D_{p_1} + D_{p_2},\ \mathrm{E}[D_{p_1p_2}] = \mathrm{E}[D_{p_1}] + \mathrm{E}[D_{p_2}] \tag{VI.2}$$

$$\sigma_{p_1p_2}^2 = \sigma_{p_1}^2 + \sigma_{p_2}^2 + 2\sigma_{p_1}\sigma_{p_2}\rho_{(p_1,p_2)} \tag{VI.3}$$

where $0 \leq \rho_{(p_1,p_2)} \leq 1$ is the correlation between D_{p_1}, D_{p_2}.

Given n parallel paths from a to b, we tune nominal path delays using existing methods [51, 53] to bring their difference under 10 ps. We also size the drivers, that jointly drive the sink, to have similar strength. Under these circumstances, the random variable of path delay and its expectation (nominal delay) are

$$D_{p(a,b)} = \sum_{i=1}^{n} D_{p_i}/n,\ d_{p(a,b)} = \sum_{i=1}^{n} d_{p_i}/n \tag{VI.4}$$

This averaging is illustrated in Figure 6.10. Detailed analysis is given in Section 6.4.2 and Table 6.6.

Then, the variance of $D_{p(a,b)}$ is

$$\sigma_{p(a,b)}^2 = \sum_{i=1}^{n} \sigma_{p_i}^2/n^2 + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \sigma_{p_i}\sigma_{p_j}\rho_{(p_i,p_j)}/n^2 \tag{VI.5}$$

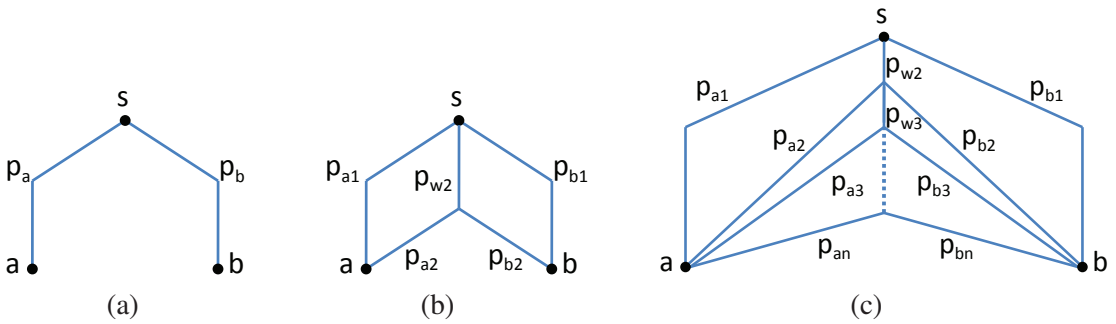


Figure 6.1: Simple clock networks with source node s, two sink nodes a and b. All paths are considered buffered. (a) a tree, (b) redundant paths. (c) n multilevel paths for each sink. Each i-th ($2 \leq i$) new root-to-sink path consists of a shared p_{w_i} section and a p_{a_i} or p_{b_i} section that is not shared.

Example III.1 Consider the case $n = 2$, $\sigma^2_{p_1} = \sigma^2_{p_2} = 10$ and $\rho_{(p_1,p_2)}$=0.1. Then $\sigma^2_{p(a,b)} = 5.5$, which reduces standard deviation by about 26% compared to a single path. Thus, having multiple paths reduces the impact of PVT variation compared to a single path.

6.1.2 Impact of variation on skew

Let s be a source node and a, b be two sink nodes. *Nominal skew* (without variation) is defined as

$$\text{skew}_{(a,b)} = |d_{p(s,a)} - d_{p(s,b)}| \tag{VI.6}$$

We define *total signed skew* (with variation), *mean signed skew*, *variational signed skew*, *signed skew variance*, *total absolute skew* and *variational absolute skew* as follows.

$$S_{(a,b)} = D_{p(s,a)} - D_{p(s,b)}, \quad \bar{S}_{(a,b)} = \text{E}[S_{(a,b)}] \tag{VI.7}$$

$$S^*_{(a,b)} = S_{(a,b)} - \bar{S}_{(a,b)}, \quad \sigma^2_{s(a,b)} = \text{E}[{S^*_{(a,b)}}^2] \tag{VI.8}$$

$$\text{skew}_{(a,b)} = |S_{(a,b)}|, \quad \text{skew}^*_{(a,b)} = |S^*_{(a,b)}| \tag{VI.9}$$

[1]While specific sources of variation and the probability distributions of device parameters can be complicated, the Central Limit Theorem suggests that path delay distributions are close to normal. Empirical data in Figure 6.9 confirm that the delay of a buffered path is normally distributed.

Since $S_{(a,b)}$ has a normal distribution, $\mathrm{skew}_{(a,b)}$ has a folded normal distribution.[2] Figure 6.2 shows empirical distributions for signed and absolute skew of two example sink pairs from the experimental results in Section 6.4.2. The mean and variance of $\mathrm{skew}_{(a,b)}$ can be derived from the mean μ and variance σ of $S_{(a,b)}$ as

$$\mathrm{E}[\mathrm{skew}_{(a,b)}] = \sigma\sqrt{2/\pi}\exp(-\mu^2/2\sigma + \mu\big(1 - 2\Phi(-\mu/\sigma)\big) \tag{VI.10}$$

$$\begin{aligned}\mathrm{var}[\mathrm{skew}_{(a,b)}] = \mu^2 + \sigma^2 - \{\sigma\sqrt{2/\pi}\exp(-\mu^2/2\sigma^2) \\ + \mu\big(1 - 2\Phi(-\mu/\sigma)\big)\}^2\end{aligned} \tag{VI.11}$$

where $\Phi(\cdot)$ denotes the cumulative distribution function of a standard normal distribution. When nominal skew is zero,

$$\text{Expected skew} : \mathrm{E}[\mathrm{skew}_{(a,b)}] = \sigma_{s(a,b)}\sqrt{2/\pi} \tag{VI.12}$$

$$\text{Skew variance} : \mathrm{var}[\mathrm{skew}_{(a,b)}] = \sigma^2_{s(a,b)}(1 - 2/\pi) \tag{VI.13}$$

Note that mean absolute skew can be positive with zero nominal skew.

For yield analysis, given a variation bound $x > 0$,

$$\mathrm{P}[\mathrm{skew}_{(a,b)} < x] \equiv \mathrm{P}[-x < S_{(a,b)} < x] \tag{VI.14}$$

This suggests that we can use signed skew as a proxy for the analysis of absolute skew. In other words, we can obtain the yield of skew ($\mathrm{P}[\mathrm{skew}_{(a,b)} < x]$) by examining the yield of signed skew ($\mathrm{P}[-x < S_{(a,b)} < x]$). In this section, we analyze the impact of variation on signed skew because of mathematically simpler analysis. Figure 6.1a illustrates a simple

[2] Given a normally distributed random variable $X = N(\mu, \sigma)$, the variable $|X|$ follows the *folded normal* distribution [58].

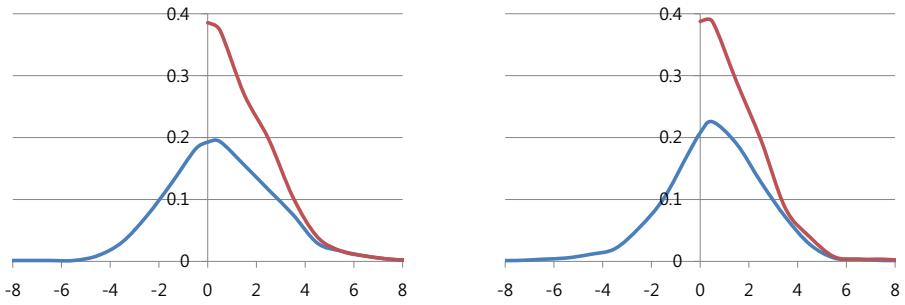


Figure 6.2: Empirical distributions of signed and absolute skew of two example sink pairs. The data are collected from Monte-Carlo simulations with variations. (a) Sink pair with nominal skew 0.3 ps. (b) Sink pair with nominal skew 1.2 ps.

clock tree with one path per sink. In this case, the skew variance is

$$\sigma^2_{s(a,b)} = \mathrm{var}(S_{(a,b)}) = \sigma^2_{p_a} + \sigma^2_{p_b} - 2\sigma_{p_a}\sigma_{p_b}\rho_{(p_a,p_b)} \tag{VI.15}$$

We extend this analysis to clock networks with multiple paths for each sink node, as illustrated in Figure 6.1b. p_{w_2} is the shared path and p_{a_2}, p_{b_2} connect the shared path to the sinks a and b. From the multiple-path delay variation model from Section 6.1.1, we obtain

$$D_{p(s,a)} = \big(D_{p_{a1}} + (D_{p_{w2}} + D_{p_{a2}})\big)/2 \tag{VI.16}$$

$$D_{p(s,b)} = \big(D_{p_{b1}} + (D_{p_{w2}} + D_{p_{b2}})\big)/2 \tag{VI.17}$$

Skew between a and b, and its variance can be expressed as

$$S_{(a,b)} = \big((D_{p_{a1}} + D_{p_{a2}}) - (D_{p_{b1}} + D_{p_{b2}})\big)/2 \tag{VI.18}$$

$$\begin{aligned}\sigma^2_{s(a,b)} = {} & (\sigma^2_{p_{a1}} + \sigma^2_{p_{a2}} + \sigma^2_{p_{b1}} + \sigma^2_{p_{b2}})/4 \\ & + (\sigma_{p_{a1}}\sigma_{p_{a2}}\rho_{(p_{a1},p_{a2})} + \sigma_{p_{b1}}\sigma_{p_{b2}}\rho_{(p_{b1},p_{b2})})/2 \\ & - \sum_{i=1}^{2}\sum_{j=1}^{2}(\sigma_{p_{ai}}\sigma_{p_{bj}}\rho_{(p_{ai},p_{bj})})/2\end{aligned} \tag{VI.19}$$

Example III.2 Consider the clock tree in Figure 6.1a with $\sigma^2_{p_a} = \sigma^2_{p_b} = 50$ and $\rho = 0$. Then from Formula VI.15, $\sigma^2_{s(a,b)} = 50 + 50 = 100$. We assume the variation constraint to be 15 ps with yield 95% (i.e., $\mathrm{P}[-15 < S_{(a,b)} < 15] > 95\%$, Figure 6.3). However, with $\sigma_{s(a,b)} = 10$, the probability is

$$\mathrm{P}[-15 < S_{(a,b)} < 15] = 86.64\% \tag{VI.20}$$

The current tree structure does not satisfy the given variation constraint. In this case, we

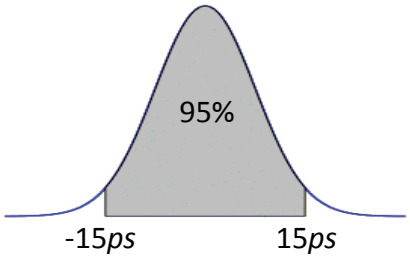


Figure 6.3: Skew limit 15 ps with yield 95%.

can insert a new subtree and fuse it to the original tree to enhance robustness.

Example III.3 Consider adding a subtree with three paths (p_{w2}, p_{a2}, p_{b2}) to Figure 6.1a and build a fusion topology as in Figure 6.1b with $\sigma^2_{p_w 2} = \sigma^2_{p_{a2}} = \sigma^2_{p_{b2}} = 25$. From Formula VI.19, $\sigma^2_{s(a,b)}$ reduces down to 37.5. Now the probability becomes

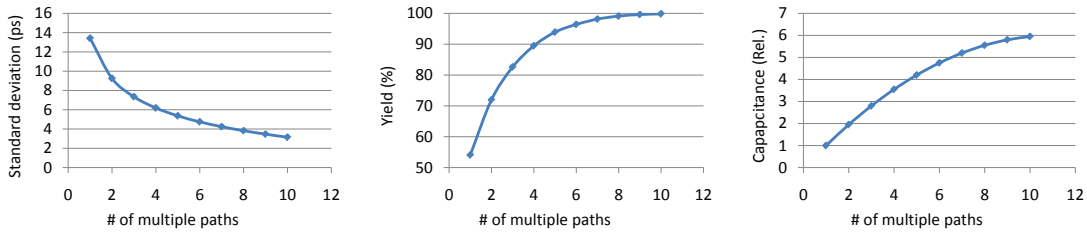
$$\mathrm{P}[-15 < S_{(a,b)} < 15] = 98.5\% \tag{VI.21}$$

which satisfies the given variation constraint.

6.1.3 Multiple redundant paths

We generalize the above analysis to clock networks with $n > 1$ redundant paths per sink as illustrated in Figure 6.1c.

$$D_{p(s,a)} = \big(D_{p_{a1}} + \sum_{i=2}^{n} (D_{p_{wi}} + D_{p_{ai}})\big)/n \tag{VI.22}$$



(a) (b) (c)

Figure 6.4: The impact of redundant paths for a pair of critical sinks (Figure 6.1c) on clock-network parameters, based on Formulas VI.25, VI.27 and VI.28. The skew constraint and ρ are set to 10 ps and 0.1 respectively. (a) Standard deviation. (b) Yield. (c) Relative total capacitance of each clock network compared to the total capacitance of the clock tree without redundant paths ($n = 1$).

$$D_{p(s,b)} = \big(D_{p_{b1}} + \sum_{i=2}^{n}(D_{p_{wi}} + D_{p_{bi}})\big)/n \tag{VI.23}$$

$$S_{(a,b)} = \big(\sum_{i=1}^{n}(D_{p_{ai}} - D_{p_{bi}})\big)/n \tag{VI.24}$$

$$\begin{aligned}\sigma^2_{s(a,b)} = &\sum_{i=1}^{n}(\sigma^2_{p_{ai}} + \sigma^2_{p_{bi}})/n^2 \\ &+2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\sigma_{p_{ai}}\sigma_{p_{aj}}\rho_{(p_{ai},p_{aj})} + \sigma_{p_{bi}}\sigma_{p_{bj}}\rho_{(p_{bi},p_{bj})})/n^2 \\ &-2\sum_{i=1}^{n}\sum_{j=1}^{n}(\sigma_{p_{ai}}\sigma_{p_{bj}}\rho_{(p_{ai},p_{bj})})/n^2\end{aligned} \tag{VI.25}$$

In the case when $\sigma_{p_{ai}} = \sigma_{p_{bi}} = \sigma$ and all ρ values are equal,

$$\sigma^2_{s(a,b)} = 2\sigma^2(1-\rho)/n \tag{VI.26}$$

Just as in Formula VI.15, highly correlated path delays lead to small skew variance.

Example III.4 Figure 6.4 illustrates how n redundant paths for each sink (as in Figure

6.1c) reduce $\sigma_{s(a,b)}$ and increase yield (based on Formula VI.25). Here we assume

$$\sigma^2_{p_{ai}} = \sigma^2_{p_{bi}} = 100 - 10(i-1),\ 1 \le i \le 10 \tag{VI.27}$$

$$\begin{aligned} \mathrm{cap}(p_{a_i}) = \mathrm{cap}(p_{b_i}) = 100 - 10(i-1),\ 1 \le i \le 10 \\ \mathrm{cap}(p_{w_1}) = 0,\ \mathrm{cap}(p_{w_i}) = 10,\ 2 \le i \le 10 \end{aligned} \tag{VI.28}$$

where $\mathrm{cap}(p)$ represents the capacitance of the path p.

In practice, we select *eligible* sinks a and b (see Section 2.1) that maximize initial $\sigma^2_{s(a,b)}$. Thus $\rho_{(p_{a1},p_{b1})}$ will be small, but, for additional redundant paths, $\rho_{(p_{ai},p_{bi})}$ will be greater, especially when a and b are located close to each other. These paths are added so that $\rho_{(p_{ai},p_{aj})}$ and $\rho_{(p_{bi},p_{bj})}$ remain small. The same statistical analysis applies to process, voltage and temperature (PVT) variations.

6.1.4 Skew of a clock network

Given a clock network Ψ and a skew limit x, let $\mathcal{E}$ be a set of eligible sink pairs of Ψ. We define the following parameters (skew, nominal skew, mean skew and yield) for the entire clock network:

$$\mathrm{skew}_\Psi = \max_{(a,b)\in\mathcal{E}} \mathrm{skew}_{(a,b)} \tag{VI.29}$$

$$\mathrm{skew}_\Psi = \max_{(a,b)\in\mathcal{E}} \mathrm{skew}_{(a,b)} \tag{VI.30}$$

$$\overline{\mathrm{skew}}_\Psi = \mathrm{E}[\max_{(a,b)\in\mathcal{E}} \mathrm{skew}_{(a,b)}] \tag{VI.31}$$

$$\begin{aligned} \mathrm{yield}_\Psi &= \mathrm{P}[\max_{(a,b)\in\mathcal{E}} \mathrm{skew}_{(a,b)} < x] \\ &= \mathrm{P}[-x < S_{(a,b)} < x\ ,\ \forall (a,b) \in \mathcal{E}] \end{aligned} \tag{VI.32}$$

Let σ be the standard deviation of *the most critical sink pair* in Ψ (i.e., $\sigma = \max_{(a,b)\in\mathcal{E}} \sigma_{s(a,b)}$). If $\mathrm{skew}_\Psi \gg \sigma$, then $\overline{\mathrm{skew}}_\Psi$ and yield_Ψ are significantly affected by skew_Ψ (nominal skew of a clock network). However, when $\mathrm{skew}_\Psi \ll \sigma$, the clock-network's yield is closely related to the yields of *critical sink pairs* (see Section 6.2.1). Our methodology invokes nominal skew optimizations to satisfy $\mathrm{skew}_\Psi \ll \sigma$ (see Section 6.3). Therefore our proposed methods in Sections 6.2 and 6.3 for enhancing robustness of critical sink pairs effectively increase the yield of Ψ.

6.2 Multilevel tree fusion

Analysis in Section 6.1 suggests that one can reduce the impact of variation on clock skew by driving critical sinks through multiple redundant paths. To generalize, we propose a novel family of clock-network structures, called *fused multilevel trees*, which maintains advantages of tree structures and incrementally enhances robustness to variation by trading-off power and robustness.

6.2.1 Critical sink pairs

After performing initial-tree construction according to [53], we analyze the impact of variation on skew between eligible sink pairs. Using models from Section 6.1.2, we can determine the variance and standard deviation of skew between each sink pair and detect critical sink pairs that are not robust enough with respect to given timing constraints. Eligible sink pairs are often geometrically close (or placed within the local skew distance limit in ISPD10 CNS benchmarks). However, they can be distant in the tree, i.e., the shortest tree-path connecting them can traverse many tree edges. These sinks are included

in the set of critical sink pairs after variational analysis because the impact of variations accumulates on long paths, resulting in significant skew variance.

6.2.2 Construction of auxiliary trees and their fusion

Once we find all critical sink pairs, we cluster them based on their least common ancestors (LCA) in the tree. The pairs that share LCA are clustered, and a set of sinks is formed as the union of the sink pairs in the cluster. The LCA plays the role of the clock source for a new auxiliary tree that connects to the sinks in a given set. Here we use the same tree-construction algorithm that we used for initial tree construction.

The nominal delays of multiple redundant paths from the clock source to each critical sink must be carefully synchronized in order to reduce nominal skew in the fused topology. This process is discussed in detail in Section 6.3.2. Figure 6.5 illustrates detection of critical sink pairs and the addition of auxiliary trees to enhance robustness.

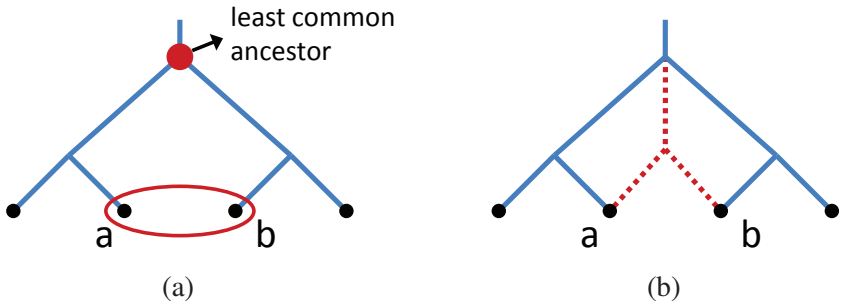


Figure 6.5: (a) A critical sink pair is indicated by a red oval and the LCA of two sinks is shown. (b) Corresponding subtree for the sink cluster in (a).

After auxiliary trees are constructed and fused, we analyze the impact of variation on skew of eligible sink pairs again. Since there are multiple paths to some sinks, we utilize variation modeling from Section 6.1.3. If some critical sinks remain, we construct another round of auxiliary trees and fuse them into the main network to enhance robust-

ness. This robustness evaluation and tree construction/fusion process is repeated until we cannot find a critical sink pair anymore. The success of our iterative fusion process critically depends on the precision of delay synchronization of redundant paths by clock-tree tuning. If implemented correctly, every fusion iteration significantly reduces the number of critical sinks (Tables 6.4 and 6.5), but if path synchronization fails, this improvement is not guaranteed. Figure 6.6 illustrates the proposed methodology including initial tree construction, detection of critical sink pairs and multilevel tree fusion.

6.2.3 Advantages of the multilevel tree fusion topology

The new clock-network structure is a joint of several trees that provides multiple redundant paths, helping to improve network robustness and satisfy skew constraints. Such a clock network exhibits the redundancy and robustness of a mesh but is easier to analyze and optimize. Our results in Section 6.4.2 shows that fusion topologies can be essentially as robust as meshes, at a fraction of capacitance budget.

Multilevel tree fusion topology is technically not a tree structure because of interconnect loops. However, those loops always close at the sink nodes, which makes it easy to reduce not only the complexity of variational analysis but also nominal skew by various tree-based skew optimization techniques. Section 6.3.2 outlines the use of tree optimization techniques in this context.

6.3 Implementation insights

Figure 6.7 shows our methodology for multilevel tree fusion.

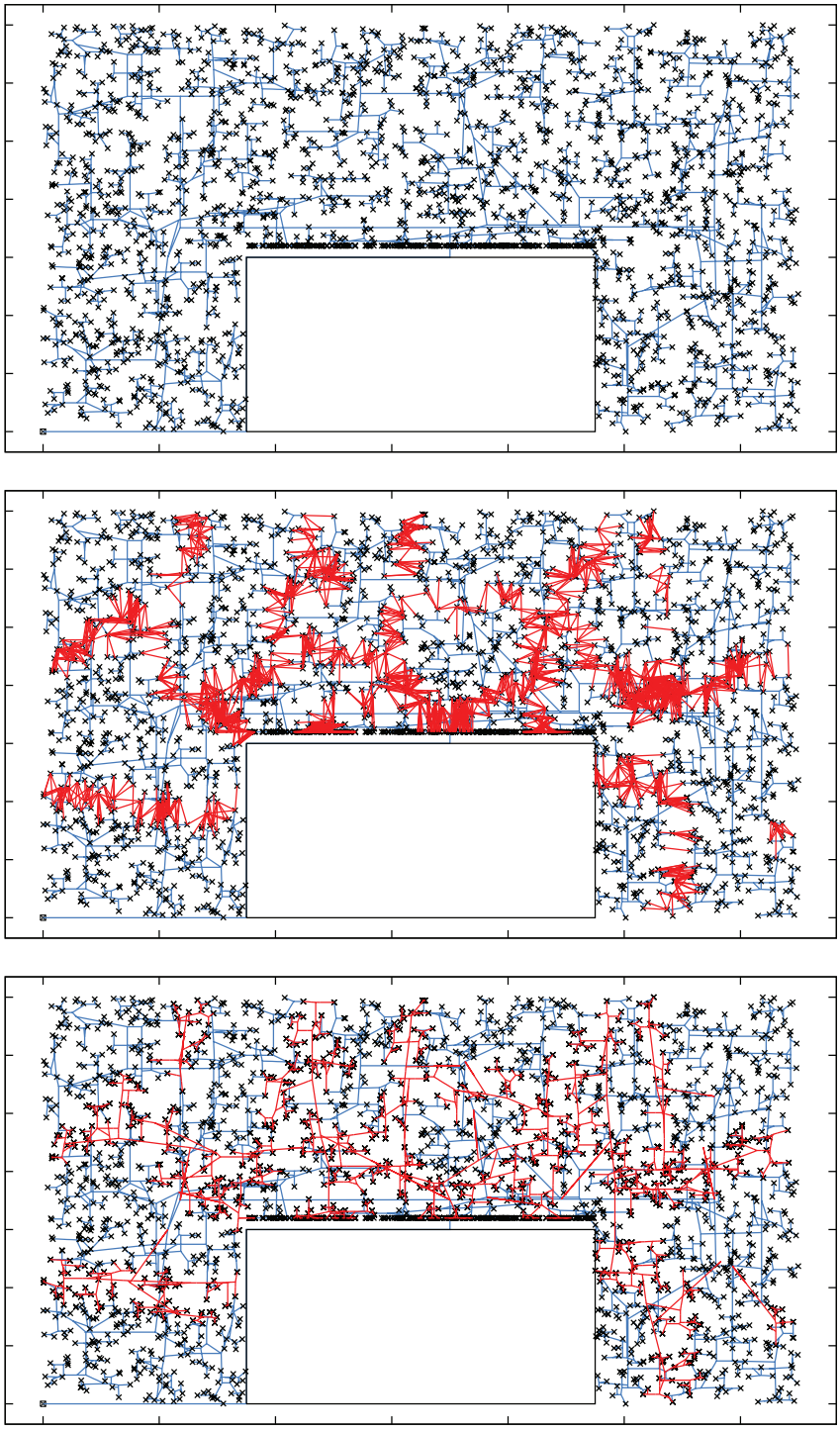


Figure 6.6: Illustration of multilevel tree fusion on *ispd10cns02*. (a) Initial tree construction. (b) Critical sink pairs are connected by red lines. (c) Auxiliary trees are fused in to enhance robustness.

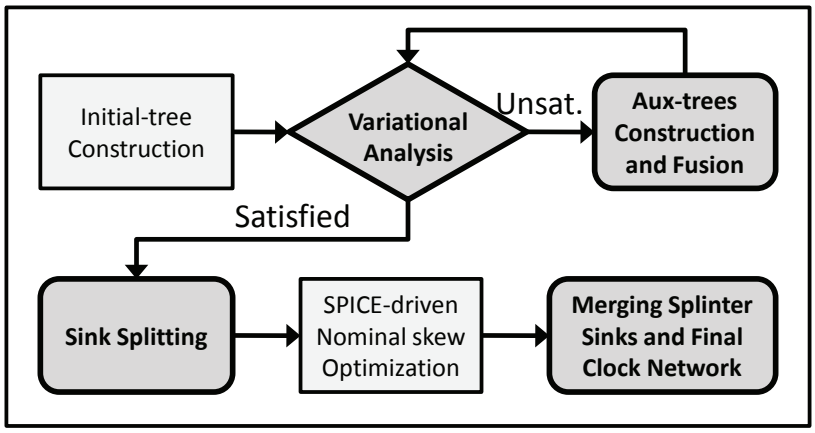


Figure 6.7: Key steps of multilevel tree fusion. Proposed techniques are indicated with darker rounded boxes and a lozenge. Plain boxes represent techniques adapted from earlier publications.

6.3.1 Estimating variation on a buffered path

After initial-tree construction [51, 53], we perform variational analysis based on the methods in Section 6.1.2 and build fusion topology to enhance robustness. For precise variational analysis, it is important to estimate Gaussian random variables for each buffered path. For accurate estimation of random variables, we build various test trees for given technology node, buffer and wire library and variation environment. Then we perform Monte-Carlo simulations with variation and record the variance of each buffered path in a look-up table. It is not necessary to record the mean of each random variable because our experimental results show that $\mathrm{E}[X]$ is nearly zero for all cases. The table is accessed by wirelength w and buffer count b to estimate the impact of variation on a buffered path with wirelength w and b buffers. Finally, the table is used to produce a least-squares fit F.

For a buffered path p of length w_p with b_p buffers,

$$\sigma_p^2 = \mathrm{F}(w_p, b_p) \tag{VI.33}$$

With a Gaussian estimate of path delay, we analyze the impact of variation on eligible sink pairs and perform multilevel tree fusion as described in Section 6.2.

6.3.2 Splinter sinks

Since the initial and auxiliary trees are built using Elmore delay, they need to be tuned using more accurate delay calculations. Therefore we reduce skew by a SPICE-driven optimization process. Our novel clock-network structure is similar to traditional trees except for loops that close at critical sinks. To leverage the efficiency of existing tree-optimization techniques, we propose to split (clone) each critical sink and distribute its input capacitance among the resulting splinter sinks, as illustrated in Figure 6.8. Once splinter sinks are generated, there is no metal loop and our clock network becomes a tree, amenable to existing tree-optimization techniques. A key challenge is to correctly model nominal delays of multiple paths ending at the same sink, and then equalize them using tree-tuning techniques.

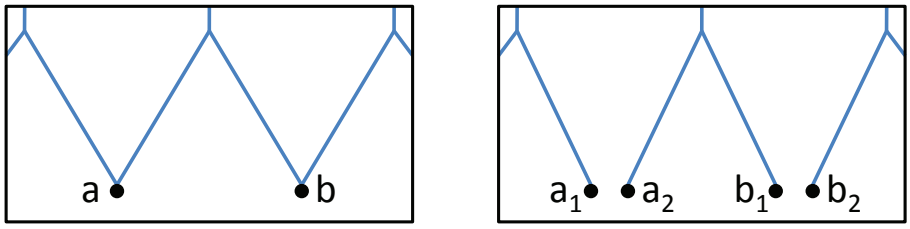


Figure 6.8: (a) Multiple paths from clock source to sinks a and b. (b) Splinter sinks are generated to utilize tree optimization algorithms.

We adopted the slack computation and wiresnaking techniques described in [53] to reduce nominal skew measured by SPICE simulations. During SPICE-driven skew optimization, our goal is to make nominal skew as small as possible.

After nominal skew optimization, in the context of splinter sinks, the average nominal skew drops below 4 ps on the ISPD 2010 CNS benchmarks. We merge splinter sinks to recover the fusion topology structure, at which point sink latencies may change and nominal skew may worsen. However, our experiments show that this deterioration can be limited to 2 ps in the worst case.[3] The average nominal skew of fusion topologies on the ISPD 2010 CNS benchmarks is 2.55 ps.

6.4 Empirical validation: Contango 3.0

Our empirical evaluation of multilevel tree fusion focuses on total capacitance and robustness to variations. We use ISPD 2010 CNS benchmarks but enhance their buffer library and variation setup to perform more realistic experiments.

6.4.1 Experiment design

ISPD 2010 CNS benchmarks are based on microprocessor designs from IBM and Intel and use a 45 nm technology library. Each benchmark is given a local-skew limit and local skew distance bound. Result are evaluated by 500 Monte-Carlo simulations with a given variation model, with respect to a given yield constraint. ISPD 2010 benchmarks suffer from a recognized deficiency in the modeling of numerous parallel buffers (that may or may not appear in the clock network), which underestimates electrical parasitics and power overhead. Process variations are not spatially correlated, making parallel buffers completely independent and underestimating the impact of process variations. These deficiencies encourage unrealistic clock-network configurations. To this end, the best pub-

[3]It is important to note that the number of splinter sinks for a given sink may increase by at most one during each fusion iteration. This significantly simplifies delay synchronization for redundant paths.

lished results for the ISPD 2010 benchmarks [67] seem to require the stacking of numerous inverters in a unique 10+40 configuration. The authors attribute the quality of results to a new cross-link insertion technique, but do not report results without cross-link insertion to substantiate this claim. Results in [53] report even smaller skews but greater capacitance, but the authors also stack numerous (32) small inverters in parallel.

num. par. buf.	nominal skew (ps)	total skew				cap. (fF)
		mean (ps)	σ (ps)	yield ($\%$)	95$\%$ (ps)	
8	2.082	5.81	1.18	92.4	7.75	26647
16	0.929	3.49	0.88	99.9	5.23	28093
24	1.843	3.16	0.80	99.9	4.70	32619

Table 6.1: Results of clock trees on *ispd10cns05* with parallel buffering. Local skew limit is 7.5 ps as in the ISPD 2010 benchmarks. The statistics of nominal skew, total skew are reported based on Monte-Carlo simulations. For each tree, we report its mean, standard deviation (σ), as well as yield for a given skew limit. '95$\%$' column represents the worst local skew for 95$\%$ yield.

Table 6.1 illustrates how one can reduce the impact of process variation by only using excessive parallel buffers without any structural modification. It shows that competitive results on the ISPD 2010 benchmarks can be easily achieved by stacking only 16 small inverters in parallel.

We now propose a different experimental configuration to avoid major shortcomings of the ISPD 2010 benchmarks. First, instead of the ISPD 2010 buffer library that exhibits uniformly-distributed variation, we use a buffer type with Gaussian variation. Table 6.2 compares buffers used in the ISPD 2010 benchmarks and in this work.

By essentially clustering a reasonable number of small ISPD buffers into one large buffer we deliberately avoid parallel buffer stacking to prevent unrealistic modeling of constituent buffers as experiencing independent process variations. Unlike many previous publications, we limit our empirical validation to a single wire type to illustrate that

buffer type	in cap (fF)	out cap (fF)	out res Ω	distribut'n of proc. variation	σ (V)	parallel buffers allowed
ispd10b1	35	80	61.2	uniform	0.043	yes
ispd10b2	4.2	6.1	440	uniform	0.043	yes
our work	33.6	48.8	55	Gaussian	0.015	no

Table 6.2: Comparison of buffer types. *ispd10b1* and *ispd10b2* are two buffer types in ISPD 2010 CNS benchmarks. The large buffer utilized in this work has Gaussian variation and parallel buffering is not allowed. The buffer type in this work is intended to represent a composite buffer made from 8 *ispd10b2* buffers, but in a way that would prevent modeling constituent buffers as experiencing independent PVT variation.

proposed multilevel tree fusion can still produce high-quality clock networks. We also note that spatially-correlated variation is only responsible for a fraction of total variation, whereas random variation also makes a significant contribution. Thus, our experimental setup is pessimistic and serves to show that our proposed technique can achieve strong results even in adverse circumstances. Using one buffer type for clock-network synthesis also restricts the flexibility to allocate driver strength throughout the clock network. We use this limitation as a handicap in our experiments to highlight the strength of multilevel tree fusion.

Figure 6.9a shows the impact of variations on a buffered path with our buffer type. The path is buffered by four inverters. Monte-Carlo simulations based on the variation model from the ISPD 2010 benchmarks are performed. The experiment validates our variation modeling in Section 6.1, in which the delay of a buffered path is modeled by a Gaussian variable. We perform the same experiment based on an industrial 45 nm technology (Figure 6.9b). The 45 nm low-power process from STMicro is utilized for the experiment. The result shows that the delay of a buffered path on an industrial technology also can be modeled by a Gaussian variable.

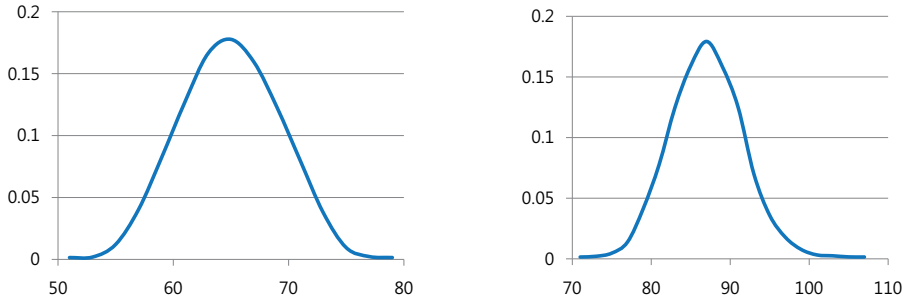


Figure 6.9: Impact of variations on a buffered path. The path is 2 mm long with 30 fF load capacitance at the end and buffered by 4 inverters. (a) The 45 nm technology, variation model from the ISPD 2010 benchmarks and a buffer type used in our work are utilized. (b) The 45 nm low-power technology, buffer library and variation model from STMicro are utilized.

6.4.2 Empirical results

Table 6.4 shows empirical results on the *ispd10cns08* benchmark. We vary the local skew limit for the benchmarks to evaluate the flexibility of our novel clock-network structure. Once again, we use only one Gaussian buffer type without parallel stacking. When there is no local skew limit, the initial clock tree is left unchanged. To satisfy increasingly difficult skew constraints, additional auxiliary trees are generated and fused to enhance robustness of clock networks. Total clock-network capacitance increases as local skew limit decreases because the tree must become more robust. The statistics of variational skew are also shown in the table. Since nominal skew varies for each fusion topology, variational skew more correctly represents the impact of variations on skew. As shown in the table, variational skew consistently decreases as the robustness of fusion topologies is improved. The results show that the multilevel fusion topology exhibits sufficient flexibility to incrementally improve robustness based on variational analysis with given local skew limit. Compared to traditional tree structures, clock-network capacitance is increased by 59.5%

to satisfy the difficult skew constraints with 4.5 ps skew limit. The probability density functions for skew of each clock network in Table 6.4 are illustrated in Figure 6.11.

method	total skew			
	mean (ps)	95% (ps)	cap. (fF)	cap. Ratio
CNSRouter [105]	3.58	5.37	97421	2.30
[86]	-	5.47	228243	5.38
our work	2.17	5.06	42414	1.00

Table 6.3: Comparison of results on *ispd10cns08* to published data for meshes. Local skew limit 6.0 ps is used to produce a clock network with better robustness than meshes. Our clock network is more robust than meshes but also 2.30× greater power efficient than CNSRouter [105].

Table 6.3 compares our clock network with those produced by CNSRouter [105] and by techniques in [86]. Our clock network is more robust than meshes, and exhibits significantly smaller total capacitance.

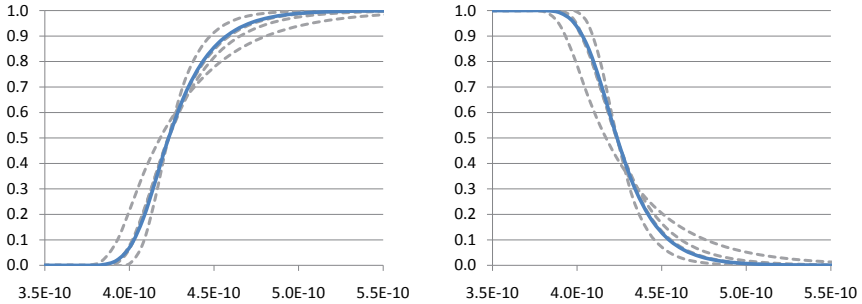
skew limit (ps)	nominal skew (ps)	total skew				variational skew			cap. (fF)	◷ (s)
		mean (ps)	σ (ps)	yield (%)	95% (ps)	mean (ps)	σ (ps)	95% (ps)		
-	1.713	5.471	1.116	-	7.563	5.380	1.107	7.405	32580.4	781.0
7.5	2.673	5.295	0.991	97.6	7.159	5.048	1.032	6.945	37279.4	1100.9
7.0	2.294	4.788	0.931	97.8	6.325	4.456	0.952	6.046	40393.7	1721.8
6.5	1.967	4.275	0.883	98.4	5.822	3.884	0.890	5.533	41641.6	1787.6
6.0	2.171	3.740	0.757	99.0	5.06	3.423	0.820	4.918	42414.1	2192.8
5.5	2.639	3.851	0.754	97.2	5.29	3.411	0.793	4.834	44053.4	2204.5
5.0	2.020	3.211	0.673	99.0	4.508	2.723	0.751	4.220	48440.9	1913.1
4.5	2.115	2.993	0.647	97.0	4.125	2.485	0.655	3.711	51955.5	3900.9

Table 6.4: Results on *ispd10cns08* with different local skew limits. The statistics of nominal skew, total skew and variational skew are reported based on Monte-Carlo simulations. For each tree, we report its mean, standard deviation (σ), as well as yield for a given skew limit. the worst local skew when yield is 95%. All the results satisfy slew constraints.

The reported nominal skews in Table 6.4 confirm that our strategy of utilizing tree-optimization techniques by generating splinter sinks is effective in reducing nominal skews of fused clock networks. To more explicitly validate our splinter-sink technique, Figure 6.10 illustrates SPICE waveforms at a reconvergent sink and its splinter sinks. The worst-case reconvergent sink (Sink 680) in the clock network with skew limit 4.5 ps on

skew limit (ps)	nominal skew (ps)	total skew				variational skew			cap. (fF)	◷ (s)
		mean (ps)	σ (ps)	yield (%)	95% (ps)	mean (ps)	σ (ps)	95% (ps)		
-	0.980	16.47	2.619	-	22.086	16.46	2.595	21.88	37704.8	293.1
22	2.333	15.32	3.137	98.0	21.08	15.21	3.035	20.83	44093.6	339.6
20	4.081	14.39	2.545	97.4	19.03	14.14	2.619	19.09	46373.9	328.6
18	1.845	12.07	2.446	98.8	16.89	11.90	2.569	17.01	48153.0	469.9
16	3.317	10.84	2.068	99.2	14.07	10.62	2.093	14.15	49918.3	509.3
14	2.412	9.359	2.068	98.4	12.72	9.103	2.201	12.82	56374.6	746.6

Table 6.5: Results on *ispd10cns08* with the buffer type *ispd10b1* in Table 6.2 without parallel buffering. The statistics of nominal skew, total skew and variational skew are reported based on Monte-Carlo simulations. Mean, standard deviation (σ) and yield for given local skew limit are reported for each tree. '95%' column represents the worst local skew when yield is 95%. All the results satisfy slew constraints.



(a) (b)

Figure 6.10: SPICE waveforms for a reconvergent sink (Sink 680) with largest temporal displacement of split sinks in a fused clock network with skew limit 4.5 ps on *ispd10cns08*. Among the four splinter sinks, the maximum rising-delay displacement before merging is 5.31 ps. The maximum rising-delay is 423.58 ps and the minimum rising-delay is 418.27 ps. The delay of the sink after merging is measure as 423.22 ps. The gray dashed lines represent the waveforms at splinter sinks before merging. The blue solid lines represent the waveforms at the sink after merging. (a) rising edge. (b) falling edge.

ispd10cns08 is shown. Since there are four different paths from the clock source to this sink, four splinter sinks are generated before optimizing nominal skew. After nominal-skew optimization with tree-optimization techniques, the maximum rising-delay difference between splinter sinks is measured as 5.31 ps. However, after merging splinter sinks, the waveform of the sink is close to the average of the waveforms of splinter sinks before merging. This reinforces our modeling of delay and variation based on Formula VI.4. Ta-

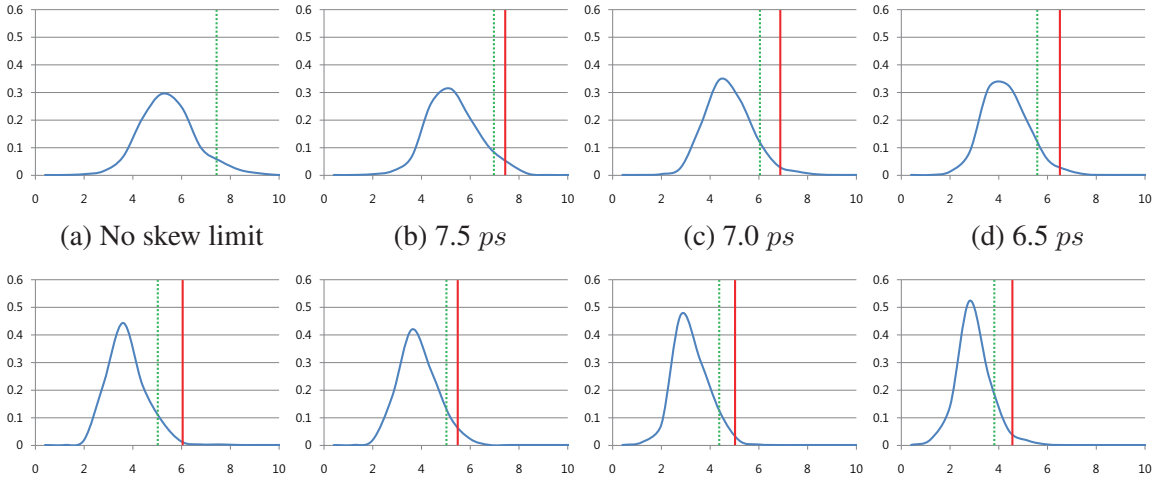
skew limit (ps)	rising edge				falling edge			
	max dSS (ps)	avg dSS (ps)	max err. (ps)	avg err. (ps)	max dSS (ps)	avg dSS (ps)	max err. (ps)	avg err. (ps)
7.5	1.50	0.42	0.89	0.31	1.65	0.52	0.98	0.34
7.0	0.90	0.22	1.15	0.40	1.73	0.52	1.08	0.45
6.5	1.02	0.25	0.93	0.30	1.64	0.47	1.77	0.45
6.0	0.89	0.36	0.95	0.34	1.04	0.24	0.86	0.30
5.5	1.99	0.21	1.16	0.29	1.53	0.53	0.97	0.26
5.0	2.40	0.33	0.91	0.29	2.17	0.67	0.85	0.26
4.5	5.31	1.07	3.60	0.31	6.75	0.38	4.30	0.31

Table 6.6: Delay analysis of splinter sinks before/after merging on *ispd10cns08*. dSS represents displacement of splinter-sink delay before merging. err. represents difference between average splinter-sink delay (before merging) and actual delay (after merging).

ble 6.6 shows the overall delay analysis of splinter sinks before and after merging. Since we utilize SPICE-driven tree-optimization techniques for nominal skew reduction under the splinter-sink condition, the average displacement in delay of splinter sinks is small. Also the error between the actual delay after merging and the average delay of splinter sinks is small due to our input-capacitance distribution during splinter-sink generation. This analysis shows that nominal skew optimization with the splinter-sink technique is reliable and efficient for fusion topologies.

In Table 6.5, we present our experimental results on *ispd10cns08* with more pessimistic modeling of process variations. In this experiment, the buffer type *ispd10b1* in Table 6.2 is utilized without parallel stacking. The purpose of this experiment is to verify how robust fusion topology is when the impact of variation is more significant than normal condition. Given that buffer delays are particularly affected by variation, the skew induced by variation is significant in the tree structure. However the results show that we can decidedly reduce the impact of variation by constructing additional auxiliary trees and

fusing them into the main network.



(a) No skew limit (b) 7.5 ps (c) 7.0 ps (d) 6.5 ps

(e) 6.0 ps (f) 5.5 ps (g) 5.0 ps (h) 4.5 ps

Figure 6.11: Skew distributions in our clock networks for *ispd10cns08*, calculated using 500 independent SPICE runs with variations (Table 6.4). The x-axis shows skew in ps, skew limits are shown with red solid lines, and the 95%-ile of skew are shown by dotted green lines.

6.5 Summary

Clock network topologies described in the literature fall into several categories: (i) trees, (ii) meshes, (iii) trees with incrementally added cross-links, (iv) combinations of trees and meshes. The gap between tree-like and mesh-like topologies remains significant, and cross-links have not been convincingly shown to improve upon pure trees, due to known shortcomings of adding one cross-link at a time. In this chapter we propose, develop and empirically evaluate a fundamentally new family of clock-network topologies derived from trees by adding auxiliary trees and iteratively fusing them into the main network. Each fusion iteration balances a large subset of skew-critical clock sinks, but as auxiliary trees are much smaller than the initial tree, the added capacitance is also small.

The accuracy of fusion iterations rests on the variational skew analysis techniques we proposed. The final clock-network topology averages out source-to-sink delay and cancels out some of the correlations induced by process, voltage and temperature (PVT) variations. Empirical evaluation shows strong results even with exceptionally pessimistic modeling of process variations, a single wire width and a single allowed buffer configuration without parallel stacking.

CHAPTER VII

Conclusions and Future Work

Synthesis of high-quality clock networks in modern synchronous VLSI designs is growing in importance as it significantly affects power-performance trade-offs. Due to semiconductor scaling, the impact of process, voltage and temperature (PVT) variation complicates the design of reliable clock networks. Therefore, multi-objective optimizations with difficult constraints are often required to produce reliable and power-efficient clock networks. We observe that published clock-network synthesis techniques often ignore intradie-process variation and overlook possible synergies with global placement. Given a clear dichotomy between trees and meshes in clock-network design, clock-network structures that can incrementally improve robustness based on given constraints are missing in prior work. Such a limited view of clock-network optimization leaves room for improvement. In this dissertation, we have investigated new objectives, constraints and concerns in clock-network synthesis, and developed new optimization techniques to address them. We propose optimization algorithms for SoC and microprocessor clock trees, register placement and novel fused multilevel trees. Below we summarize our contributions and discuss directions for future research.

7.1 Summary of contributions

We have found that a significant source of suboptimality in both academic and industry clock-network synthesis tools today is the fact that they are limited by inaccurate timing analysis, traditional physical-design flows and clock-network structures optimized for objectives that are not fully relevant. In this dissertation, we make several contributions that advance the strength and capabilities of clock-network synthesis tools for large-scale 45 nm designs, with the ultimate goal to improve the quality of leading-edge semiconductor products. Our major contributions are summarized below:

SPICE-accurate SoC clock network synthesis.

Most published algorithms and techniques establish fundamental methodologies for clock network synthesis, but perform large-scale optimization using analytical models that lose accuracy at recent technology nodes, and are not always validated by realistic SPICE simulations on large industry designs. In Chapter III we propose a methodology for SPICE-accurate optimization of clock networks, coordinated to satisfy slew constraints and achieve best trade-offs between skew, insertion delay, power, as well as tolerance to variations [51, 52]. Our implementation, called *Contango*, is evaluated on 45 nm benchmarks from IBM Research and Texas Instruments with up to 50K sinks. Contango shared the first place at the ISPD 2009 Clock-Network Synthesis Contest with two other teams. Improved experimental results after the contest show that our methodology outperforms all published results in terms of skew and shows superior scalability.

Optimization of clock trees for microprocessors.

Clock networks account for a significant fraction of system power dissipation while limiting CPU performance. Therefore, power-performance-cost trade-offs are becoming a major issue in modern high-performance CPU clock design. On the other hand, the increasing impact of process variation makes clock network synthesis particularly challenging. Mesh structures are often utilized to improve robustness to variations, but significant additional power consumption is unavoidable. In Chapter IV we propose a tree solution for CPU clock routing that improves power consumption under tight skew constraints in the presence of variations [53]. Our key contributions include a new time-budgeting step for clock-tree tuning, accurate optimizations that satisfy budgets, modeling and optimization of variational skew. Our software implementation, *Contango 2.0*, won the first place at the ISPD 2010 Clock-Network Synthesis Contest. We have shown that clock trees can be tuned to have nominal skew below 5 ps and total skew in single picoseconds in the presence of variations. Our optimizations not only satisfy given skew constraints and target yield but also lead to $4.22\times$ capacitance improvement on average over mesh structures proposed at the ISPD 2010 contest, with better yield. The comparison with a commercial clock-tree synthesis tool shows that our nominal-skew optimization techniques are more precise and reliable than an industrial tool when a given skew constraint is difficult to satisfy. Our strong empirical results suggest that clock trees constructed using accurate variational skew modeling and optimizations have distinct advantage in power consumption and competitive robustness.

Clock network optimization during placement.

Most of the existing literature on clock network synthesis assumes that register locations are given and cannot be changed. While clock networks can be improved by finding better register locations during placement, few publications develop such optimizations, hence the quality of resulting clock networks is limited by un-optimized locations of the clocked elements. In Chapter V, we propose new techniques and a methodology to optimize total dynamic power during placement for large IC designs with macro blocks [55, 56]. To this end, we advocate obstacle-aware virtual clock-tree synthesis, an arboreal clock-net contraction force with virtual nodes that can handle gated clocks, and an obstacle-avoidance force for clock edges. Our methodology is integrated into the state-of-the-art SimPL placer [47], and the total switching power is measured by utilizing Contango 2.0 [53]. A new set of 45 nm benchmarks is proposed to better represent modern IC designs. Experimental results indicate that our software implementation, Lopper, prunes clock-tree branches to reduce their length by 30.0%$\sim$36.6% and average total dynamic power consumption by 6.8%$\sim$11.6% versus conventional wirelength-driven approaches. SPICE-driven simulations show that our methods improve robustness of clock trees.

Closing the gap between tree and mesh structures.

Commonly used structures for clock networks can be categorized into two major types: trees and meshes. While tree structures were popular for clock network synthesis in older chips, mesh structures were utilized to satisfy tight variation-related constraints in high-performance microprocessor designs where performance is emphasized over power consumption. However, implementation of mesh-type clock networks requires a substantial

amount of total wire/buffer capacitance, which leads to a significant increase in total power dissipation of the design. In Chapter VI, we propose, develop and empirically evaluate a fundamentally new family of clock-network topologies derived from trees by adding auxiliary trees and iteratively fusing them into the main network [54]. Specific innovations include: *(i)* Statistical models for delay and skew in buffered clock networks. *(ii)* A technique to identify critical sink pairs based on robustness analysis. *(iii)* A novel clock-network structure (*fused multilevel trees*) based on auxiliary-tree construction and fusing to enhance robustness. *(iv)* A *sink-splitting* technique for fusion topologies to leverage the efficiency of tree optimization algorithms. *(v)* An experimental configuration with monolithic wires that remedies known deficiencies in ISPD 2010 benchmarks. Each fusion iteration balances a large subset of skew-critical clock sinks, but as auxiliary trees are much smaller than the initial tree, the added capacitance is also small. The accuracy of fusion iterations rests on the variational skew analysis techniques we proposed. The final clock-network topology averages out source-to-sink delay and cancels out some of the correlations induced by process, voltage and temperature (PVT) variations. Our implementation called *Contango3.0* produces robust clock networks even for challenging skew limits, without parallel buffering used by other implementations. It also offers a fine trade-off between power and robustness, increasing the capacitance of the initial tree by less than 60%, which results in 2.3× greater power efficiency than mesh structures.

7.2 Directions for future research

The results developed during the course of our research suggest several directions for future exploration. In Chapter VI, we propose a novel non-tree topology that fuses several

auxiliary clock trees *at sink nodes* to create redundant paths from a clock source to sinks. In this context, recall that high-quality register placement proposed in Chapter V tends to locally cluster registers to reduce leaf-level clock-net wirelength. When clock sinks are locally clustered, *tree fusion at internal nodes* may produce better clock networks in terms of power-performance trade-off. However, several difficult problems that are not explored in the existing literature need to be resolved to make this technique practical. First, for tree fusion at sink nodes, target latencies for leaf nodes of an auxiliary tree are uniform, therefore traditional clock-tree generation algorithms can be directly utilized without significant modifications. However, target latencies of internal nodes vary even for the nodes at same tree-level, hence auxiliary-tree generation algorithms must consider the uneven target delays of leaf nodes. DME-based tree-generation algorithms can be extended to address this problem but prior work has not addressed this problem with proper analysis tools. Second, the novel non-tree topology has metal loops only at sink nodes. In Chapter VI, we present the splinter-sink technique that duplicates reconvergent sinks and disconnect metal loops. This technique temporarily transforms a non-tree topology to a tree topology, then various tree-based nominal-skew optimization techniques can be adapted to reduce skew of a non-tree structure. However, since the splinter-sink technique cannot be applied to internal nodes, delay synchronization of redundant paths to an internal node is significantly more difficult compared to the case in which we can use the splinter-sink technique. Resolving these problems in future work is not going to be easy, but could help improve robustness of clock networks with superior power-performance trade-off.

Large-scale chip design may soon have hundreds of millions of standard cells including

millions of registers. Hierarchical clock-network synthesis is suggested for these large-scale designs mainly in order to reduce turn-around-time of design flows. We believe that our clock-network synthesis flows in Chapter III, IV and VI can be extended for hierarchical clock-network synthesis by applying our flow to disjoint clusters at different levels. Especially when different parts of a design have different constraints (i.e., some parts of a chip are designed for high performance, and other parts are designed for low power), our tree-fusion clock-network-synthesis flow in Chapter VI are powerful enough to generate optimal clock networks for this kind of designs. Consider a two-level clock network, with the top level driving two disjoint clusters of the bottom level. One cluster of the bottom level may consists of registers for high-performance logics and the other cluster has registers for low-power logics. We can utilize the tree-fusion flow with skew-emphasized constraints for the top-level and high-performance clock networks to generate more redundant paths, and power-emphasized constraints for the low-power clock network to build a compact clock network. We expect the results of hierarchical clock-network synthesis utilizing our flexible non-tree topology to show superior quality.

Design methodologies for TSV-based 3D-ICs are currently pursued by many developers of EDA software tools. Techniques proposed in this dissertation lend themselves naturally to such extensions. In particular, skew optimization techniques in Chapter III and IV are not limited by TSVs if clock networks of 3D-ICs maintain tree structures. Arboreal clock-net contraction force in Chapter V can be utilized for clock networks of 3D-ICs if a TSV-aware virtual clock-tree generation algorithm is adapted. In this case, TSVs will be represented by virtual nodes during placement and the optimal locations of TSVs for

minimizing switching power can be hinted by placement results. The non-tree topology in Chapter VI can be utilized for 3D-ICs to improve robustness if the characteristics and cost of TSV is properly considered during variation analysis and auxiliary-tree generation.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A. I. Abou-Seio, B. Nowak and C. Chu, "Fitted Elmore Delay: A Simple and Accurate Interconnect Delay Model," *IEEE Transactions on VLSI*, vol 12, pp. 691-696, 2004.

[2] K. Agarwal, D. Sylvester and D. Blaauw, "Simple Metrics for Slew Rate of RC Circuits Based on Two Circuit Moments," *ACM/IEEE Design Automation Conference*, 2003, pp. 950-953.

[3] C. Albrecht, A. B. Kahng, B. Liu, I. I. Mandoiu and A. Z. Zelikovsky, "On the Skew-Bounded Minimum-Buffer Routing Tree Problem," *IEEE Transactions on CAD* 22(7), pp. 937-945, 2003.

[4] C. J. Alpert, A. Devgan and C. Kashyap, "RC delay metric for performance optimization,"Simple Metrics for Slew Rate of RC Circuits Based on Two Circuit Moments," *IEEE Transactions on CAD*, vol. 20, pp. 571-582, 2004.

[5] C. J. Alpert, F. Liu, C. Kashyap and A. Devgan, "Closed-Form Delay and Slew Metrics Made Easy," *IEEE Transactions on CAD*, vol. 23, pp. 1661-1669, 2004.

[6] C. J. Alpert, D. P. Mehta and S. S. Sapatnekar, "Handbook of Algorithms for Physical Design Automation," *CRC Press*, 2009.

[7] C. J. Alpert, A. B. Kahng, B. Liu, I. I. Mandoiu and A. Z. Zelikovsky, "Minimum Buffered Routing with Bounded Capacitive Load for Slew Rate and Reliability Control," *IEEE Transactions on CAD*, vol. 22(3), pp. 241-253, 2003.

[8] C. J. Alpert, S. K. Karandikar, L. Zhuo, G.-J Nam, S. T. Quay, H. Ren , C. N. Sze, P. G. Villarrubia and M. C. Yildiz, "Techniques for Fast Physical Synthesis," *Proceedings of IEEE*, vol. 95, no. 3, pp. 573-599, 2007.

[9] C. J. Alpert, C. Chu and P. G. Villarrubia, "The Coming of Age of Physical Synthesis," *IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 246-249.

[10] H. B. Bakoglu, "Interconnections and Packaging for VLSI. Reading," *Addison-Wesley*, 1990.

[11] H. Bakoglu, J. Walker and J. Meindl, "A symmetric clock distribution tree and optimized high-speed interconnects for reduced clock skew in ULSI and WSI circuits," *IEEE International Conference on Computer Design*, 1986, pp. 118-122.

[12] K. D. Boese and A. B. Kahng, "Zero-Skew Clock Routing Trees with Minimum Wirelength," *ASIC Conference and Exhibit*, 1992, pp.111-115.

[13] S. Bujimalla and C.-K. Koh, "Synthesis of Low Power Clock Trees for Handling Power-supply Variations," *ACM/IEEE International Symposium on Physical Design* 2011, pp. 37-44.

[14] P. Chan and K. Karplus, "Computing Signal Delay in General RC Tree/Link Partitioning," *IEEE Transactions on CAD*, vol. 9, pp. 898-902, 1990.

[15] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze and M. Xie "mPL6: Enhanced Multilevel Mixed- Size Placement," *ACM/IEEE International Symposium on Physical Design*, 2006, pp. 212-214.

[16] Y.-T. Chang, C.-C Hsu, M. P.-H. Lin, Y.-W. Tsai and S.-F. Chen, "Post-Placement Power Optimization with Multi-Bit Flip-Flops," *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 218-223.

[17] T.-H. Chao, Y.-C. Hsu and J.-M. Ho, "Zero Skew Clock Net Routing," *ACM/IEEE Design Automation Conference*, 1992, pp. 518-523.

[18] T.-H. Chao, Y.-C. Hsu, J.-M. Ho and A. B. Kahng, "Zero Skew Clock Routing with Minimum Wirelength," *IEEE Transactions on Circuits and Systems*, vol. 39(11), pp. 799-814, 1992.

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

[20] Y. Cheon, P.-H. Ho, A. B. Kahng, S. Reda and Q. Wang, "Power-aware placement," *ACM/IEEE Design Automation Conference*, 2005, pp. 795-800.

[21] M. Cho, D. Z. Pan and R. Puri, "Novel Binary Linear Programming for High Performance Clock Mesh Synthesis," *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 438-443.

[22] J. Cong, A. B. Kahng and G. Robins, "Matching-based Methods for High-performance Clock Routing," *IEEE Transactions on CAD*, vol. 12(8), pp.1157-1169, 1993.

[23] J. Cong, A. B. Kahng, C.-K. Koh and C.-W. A. Tsao, "Bounded-Skew Clock and Steiner Routing," *ACM Transactions on Design Automation of Electronic Systems*, 1998, pp. 341-388.

[24] L. Conway and C. Mead, "Introduction to VLSI Systems. Reading," *Addison-Wesley*, 1980

[25] R. Davidson and J. Mackinnon, "Estimation and inference in econometrics," *Oxford University Press*, 1993.

[26] M. Donno, E. Macci and L. Mazzoni, "Power-Aware Clock Tree Planning," *ACM/IEEE International Symposium on Physical Design*, 2004, pp. 138-147.

[27] D. E. Duate, N. Vijaykrishnan and M. J. Irwin, "A clock power model to evaluate impact of architectural and technology optimization," *IEEE Transactions on VLSI*, vol. 10(6), pp. 844-855, 2002.

[28] M. Edahiro, "A Clustering-Based Optimization Algorithm in Zero-Skew Routings," *ACM/IEEE Design Automation Conference*, 1993, pp. 612-616.

[29] W. C. Elmore, "The transient response of damped linear network with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, pp. 66-63, 1948.

[30] L. v. Ginneken, "Buffer Placement in Distributed RC-tree Networks For Minimal Elmore Delay," *IEEE International Symposium on Circuits and Systems*, 1990, pp. 865-868.

[31] P. E. Gronowski, W. J. Bowhill, R. P. Preston and M. K. Gowan, and R. L. Allmon, "High-Performance Microproccesor Design," *IEEE Journal of Solid-State Circuits*, vol. 33(5), pp. 676-686, 1998.

[32] D. Harris, M. Horowitz and D. Liu, "Timing Analysis Including Clock Skew," *IEEE Transactions on CAD*, vol. 18(11), pp. 1608-1618, 1999.

[33] P.-H. Ho, "Industrial Clock Design," *ACM/IEEE International Symposium on Physical Design*, 2009, pp. 139-140.

[34] R. Ho, K. Mai and M. Horowitz, "The Future of Wires," *Proceedings of IEEE*, vol. 89(4), pp. 490-504, 2001.

[35] W. Hou, X. Hong, W. Wu and Y. Cai, "A path-based timingdriven quadratic placement algorithm," *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2003, pp. 745-748.

[36] J. Hu, A. B. Kahng, B. Liu, G. Venkataraman and X. Xu, "A Global Minimum Clock Distribution Network Augmentation Algorithm for Guaranteed Clock Skew Yield," *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2007, pp. 24-31.

[37] S. Hu, Q. Li, J. Hu and P. Li,"Utilizing Redundancy for Timing Critical Interconnect," *IEEE Transactions on VLSI*, vol. 15(10), pp. 1067-1080, 2007.

[38] J.-H. Huang, A. B. Kahng and C.-W. Tsao, "On Bounded-Skew Routing Tree Problem," *ACM/IEEE Design Automation Conference*, 1995, pp. 508-513.

[39] F. Huebbers, A. Dasdan and Y. Ismail, "Multi-Layer Interconnect Performance Corners for Variation-Aware Timing Analysis," *IEEE/ACM International Conference on Computer-Aided Design* , 2007, pp. 713-718.

[40] M. A. B. Jackson, A. Srinivasan and E. S. Kuh, "Clock Routing for High-performance ICs," *ACM/IEEE Design Automation Conference*, 1990, pp. 573-579.

[41] X. D. Jia, R. M. M. Chen and A. M. Layfield, "Circuit Partitioning for Multiprocessor SPICE," *IEEE TENCON*, 1993.

[42] A. Kahng, J. Cong and G. Robins, "High-performance clock routing based on recursive geometric matching," *ACM/IEEE Design Automation Conference*, 1991, pp. 322-327.

[43] A. B. Kahng, J. Leinig, I. L. Markov and J. Hu, "VLSI Physical Design: From Graph Partitioning to Timing Closure", *Springer*, 2011.

[44] A. B. Kahng and C.-W. Tsao, "Practical Bounded-Skew Clock Routing," *Journal of VLSI Signal Processing*, vol. 16, pp.199-215, 1997.

[45] A. B. Kahng, S. Muddu, E. Sarto and R. Sharma, "Interconnect Tuning Strategies for High-Performance ICs," *IEEE/ACM Design, Automation, and Test in Europe*, 1998, pp. 471-478.

[46] A. B. Kahng and C.-W. Tsao, "Practical Bounded-Skew Clock Routing," *Journal of VLSI Signal Processing*, vol. 16, pp.199-215, 1997.

[47] M.-C. Kim, D.-J. Lee and I. L. Markov, "SimPL: An Effective Placement Algorithm," *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 649-656.

[48] J. M. Kleinhans, G. Sigl, F. M. Johannes and K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Transactions on CAD*, vol. 10(3), pp. 356-365, 1991.

[49] T. Krazit, "ARMed for the Living Room," *CNET news*, Web. 3 Apr. 2006.

[50] W.-C. D. Lam, J. Jam, C.-K. Koh, V. Balakrishnan and Y. Chen, "Statistical Based Link Insertion for Robust Clock Network Design," *IEEE/ACM International Conference on Computer-Aided Design* , 2005, pp. 588-591.

[51] D.-J. Lee and I. L. Markov, "Contango: Integrated Optimization of SoC Clock Networks," *IEEE/ACM Design, Automation, and Test in Europe*, 2010, pp. 1468-1473.

[52] D.-J. Lee and I. L. Markov, "Contango: Integrated Optimizations for SoC Clock Networks", *VLSI Design*, vol. 2011, no. 407507, 12 pp., 2011.

[53] D.-J. Lee, M.-C. Kim and I. L. Markov, "Low-Power Clock Trees for CPUs," *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 444-451.

[54] D.-J. Lee and I. L. Markov, "Multilevel Tree Fusion for Robust Clock Networks," *IEEE/ACM International Conference on Computer-Aided Design*, 2011.

[55] D.-J. Lee and I. L.Markov, "Obstacle-aware Clock-tree Shaping during Placement," *ACM/IEEE International Symposium on Physical Design*, 2011, pp. 123-130.

[56] D.-J. Lee and I. L.Markov, "Obstacle-aware Clock-tree Shaping during Placement," *IEEE Transactions on CAD*, 2011.

[57] D.-J. Lee and I. L. Markov, The CLKISPD'05 Benchmark Suite, 2011. Downloadable from http://vlsicad.eecs.umich.edu/BK/CLKISPD05bench

[58] F. C. Leone, R. B. Nottingham and L. S. Nelson "The Folded Normal Distribution," *Technometrics*, vol. 3(4), pp. 543-550, 1961.

[59] J. Long, H. Zhou and S. O. Memik, "An O(nlogn) Edge-Based Algorithm for Obstacle-Avoiding Rectilinear Steiner Tree Construction," *ACM/IEEE International Symposium on Physical Design*, 2008, pp. 126-133.

[60] D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI circuits," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 663-670, 1994.

[61] W.-H Liu, Y.-L Li and H.-C. Chen, "Minimizing Clock Latency Range in Robust Clock Tree Synthesis," *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2010, pp. 389-394.

[62] Y. Liu, X. Hong, Y. Cai and W. Wu, "CEP: A clock-driven ECO placement algorithm for standard-cell layout," *IEEE International Conference on ASIC*, 2001, pp. 118-121.

[63] J. Lu, W.-K. Chow, C.-W. Sham and E. F. Y. Young, "A Dual-MST Approach for Clock Network Synthesis," *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2010, pp. 467-473.

[64] Y. Lu, C. N. Sze, X. Hong, Q. Zhou, Y. Cai, L. Huang and J. Hu, "Navigating Registers in Placement for Clock Network Minimization," *ACM/IEEE Design Automation Conference*, 2005, pp. 176-181.

[65] C.-L. Lung, Z.-Y. Zeng, C.-H. Chou and S.-C. Chang "Clock Skew Optimization Considering Compilicated Power Modes," *IEEE/ACM Design, Automation, and Test in Europe*, 2010, pp. 1474-1479.

[66] N. Magen, A. Kolodny, U. Weiser and N. Shamir, "Interconnect-power Dissipation in a Microprocessor," *Proceedings of Workshop on System Level Interconnect Prediction*, 2004, pp. 7-13.

[67] T. Mittal and C.-K. Koh, "Cross Link Insertion for Improving Tolerance to Variations in Clock Network Synthesis," *ACM/IEEE International Symposium on Physical Design*, 2011, pp. 29-36.

[68] L. W. Nagel and D. O. Pederson, "SPICE (Simulation Program with Integrated Circuit Emphasis)," *Memorandum No. ERL-M382, University of California, Berkeley*, 1973.

[69] G. J. Nam, C. J. Alpert, P. Villarrubia, B. Winter and M. Yildiz, "The ISPD2005 Placement Contest and Benchmark Suite," *ACM/IEEE International Symposium on Physical Design*, 2005, pp. 216-220.

[70] Nangate Inc. Open Cell Library v2009 07, 2009. Downloadable from http://www.nangate.com/openlibrary

[71] Nenzi P, "NG-SPICE: The Free Circuit Simulator," http://ngspice.sourceforge.net/.

[72] A. Odabasioglu, M. Celik and L. Pileggi, "PRIMA: Passive Reduced-Order Interconnect Macromodeling Algorithm," *IEEE Transactions on CAD*, vol. 17, 1998.

[73] J. Oh and M. Pedram, "Gated Clock Routing for Low-Power Microprocessor Design", *IEEE Transactions on CAD*, Vol. 20(6), pp. 715-722, 2001.

[74] L. T. Pillage and R. A. Rohrer, "Asymptotic Waveform Evaluation for Timing Analysis," *IEEE Transactions on CAD*, vol. 9, pp. 352-366, 1990.

[75] R.P. Pokala, R.A. Feretich and R.W. McGuffin, "Physical Synthesis for Performance Optimization", *ASIC Conference and Exhibit*, 1992, pp. 34-37.

[76] J. M. Rabaey, A. Chandrakasan and B. Nikolic, "Digital Integrated Circuits: A Design Perspective," *Prentice Hall*, Second Edition, 2003.

[77] A. Rajaram, J. Hu and R. Mahapatra, "Reducing Clock Skew Variability via Cross Links," *ACM/IEEE Design Automation Conference*, 2004, pp. 18-23.

[78] A. Rajaram, D.Z. Pan and J. Hu, "Improved Algorithms for Link-Based Non-Tree Clock Networks for Skew Variability," *ACM/IEEE International Symposium on Physical Design*, 2005, pp. 55-62.

[79] A. Rajaram and D. Z. Pan, "Variation Tolerant Buffered Clock Network Synthesis with Cross Links," *ACM/IEEE International Symposium on Physical Design*, 2006, pp. 157-164.

[80] S. M. Reddy, G. R. Wilke and R. Murgai, "Analyzing Timing Uncertainty in Mesh-based Clock Architectures," *IEEE/ACM Design, Automation, and Test in Europe*, 2006 , pp. 1-6.

[81] P. J. Restle, T. G. McNamara, D. A. Webber, P. J. Camporese, K. F. Eng, K. A. Jenkins, D. H. Allen, M. J. Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovick, B. L. Krauter and B. D. McCredie, "A Clock Distribution Network for Microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 36(5), pp. 792-799, 2001.

[82] R. S. Shelar, "An Algorithm for Routing with Capacitance/Distance Constraints for Clock Distribution in Microprocessors," *ACM/IEEE International Symposium on Physical Design D*, 2009, pp. 141-148.

[83] R. S. Shelar and M. Patyra, "Impact of Local Interconnects on Timing and Power in a High Performance Microprocessor," *ACM/IEEE International Symposium on Physical Design*, 2010, pp. 145-152.

[84] W. Shi and Z. Li, "A Fast Algorithm for Optimal Buffer Insertion," *IEEE Transactions on CAD*, vol. 24(6), pp. 879-891, 2005.

[85] X.-W. Shih and Y.-W. Chang, "Fast Timing-Model Independent Buffered Clock-Tree Synthesis," *ACM/IEEE Design Automation Conference*, 2010, pp. 80-85.

[86] X.-W. Shih, H.-C. Lee, K.-H. Ho and Y.-W. Chang, "High Variation-Tolerant Obstacle-Avoiding Clock Mesh Synthesis with Symmetrical Driving Trees," *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 452-457.

[87] X.-W. Shih, C.-C. Cheng, Y.-K. Ho and Y.-W. Chang, "Blockage-Avoiding Buffered Clock-Tree Synthesis for Clock Latency-Range and Skew Minimization," *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2010, pp. 395-400.

[88] X.-W. Shih and Y.-W. Chang, "Fast Timing-Model Independent Buffered Clock-Tree Synthesis," *ACM/IEEE Design Automation Conference*, 2011, pp. 80-85.

[89] G. Sigl, K. Doll and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?" *ACM/IEEE Design Automation Conference*, 1991, pp. 427-431.

[90] V. Smith, "The Coming War: ARM versus x86," *Bright Side of News*, Web. 7 Apr. 2010.

[91] P. Spindler, U. Schlichtmann and F. M. Johannes, "Kraftwerk2 - A Fast Force-Directed Quadratic Placement Approach Using an Accurate Net Model," *IEEE Transactions on CAD*, vol. 27(8), pp. 1398-1411, 2008.

[92] V. Khandelwal and A. Srivastava, "Variability-Driven Formulation for Simultaneous Gate Sizing and Post-silicon Tunability Allocation," *IEEE Transactions on CAD*, vol. 27(4), pp. 610-620, 2008.

[93] Synopsys, "HSPICE: Simulation and Analysis User Guide," *Synopsys*, Inc., Mountain View, CA, 2003. Release U-2003.03-PA.

[94] C. Sze, P. Restle, G.-J. Nam and C. J. Alpert, "ISPD 2009 Clock-network Synthesis Contest," *ISPD*'09, pp. 149-150. `http://www.sigda.org/ispd/contests/ispd09cts.html`.

[95] C. N. Sze, "ISPD 2010 High-Performance Clock Network Synthesis Contest: Benchmark Suite and Results," *ACM/IEEE International Symposium on Physical Design*, 2010, pp. 143-143.

[96] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel and F. Baez, "Reducing Power in High-Performance Microprocessors," *ACM/IEEE Design Automation Conference*, 1998, pp. 732-737.

[97] R. S. Tsay, "An Exact Zero-Skew Clock Routing Algorithm," *IEEE Transactions on CAD*, vol. 12(2), pp.242-249, 1993.

[98] G. Venkataraman, Z. Feng, J. Hu and P. Li, "Combinatorial Algorithms for Fast Clock Mesh Optimization," *IEEE/ACM International Conference on Computer-Aided Design*, 2006, pp. 563-567.

[99] G. Venkataraman, N. Jayakumar, J. Hu, P. Li, S. Khatri, A. Rajaram, P. McGuinness and C. Alpert, "Practical Techniques to Reduce Skew and its Variations in Buffered Clock Networks," *IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 592-596.

[100] N. Viswanathan, M.Pan and C.Chu, "FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion Control," *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2007, pp. 135-140.

[101] K. P. Vorwerk, A. Kennings and A. Vannelli, "Engineering details of a stable force-directed placer," *IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 795-800.

[102] M. Wang and M. Sarrafzadeh, "Congestion minimization during placement," *ACM/IEEE International Symposium on Physical Design*, 1999, pp. 145- 150.

[103] Y. Wang, Q. Zhou, X. Hong and Y. Cai, "Clock-Tree Aware Placement Based on Dynamic Clock-Tree Building," *IEEE International Symposium on Circuits and Systems*, 2007, pp. 2040-2043.

[104] J. L. Wyatt. Jr., "Signal delay in RC mesh networks," *IEEE Transactions on Circuits and Systems*, vol. 32(5), pp. 507-510, 1985.

[105] L. Xiao, Z. Xiao, Z. Qian, Y. Jiang, T. Huang, H. Tian and E. F. Y. Young, "Local Clock Skew Minimization Using Blockage-aware Mixed Tree-Mesh Clock Network," *IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 458-462.

[106] X. Ye, P. Li, M. Zhao, R. Panda and J. Hu, "Analysis of large clock meshes via harmonic-weighted model order reduction and port sliding," *IEEE/ACM International Conference on Computer-Aided Design*, 2007.

[107] C. Yeh, G. Wilke, H. Chen, S. Reddy, H. Nguyen, T. Miyoshi, W. Walker and R. Murgai, "Clock Distribution Architectures: A Comparative Study," *IEEE International Symposium on Quality Electronic Design*, 2006, pp. 85-91.

[108] J. Yuan and C. Svensson, "High-speed CMOS circuit technique," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 62-70, 1989.

[109] W. Zhao and Y. Cao, "New Generation of Predictive Technology Model for Sub-45 nm Early Design Exploration," *IEEE Transactions on Electron Devices*, vol. 53(11), pp. 2816-2823, 2006.