Clock Tree Synthesis for Timing Convergence and Timing Yield Improvement in Nanometer Technologies

by

Jeng-Liang Tsai

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

2005

Abstract

Designing high-performance very large-scale integration (VLSI) chips has become more challenging than ever due to nanometer effects and accelerating time-to-market cycles. Due to the interconnect delay dominance, a small routing change in the design can increase coupling capacitances on its neighboring paths and significantly increase their path delays. This can cause new timing violations and result in design iterations. While timing convergence is getting harder and harder to achieve, the accelerating time-to-market cycles further aggravate the problem. Process variations result in interconnect variations, threshold voltage variations, leakage power variations, etc. These effects not only generate reliability issues but also make the circuit performance deviate from the design specification and cause timing yield losses. Due to the increasing process variations in nanometer technologies, timing yield has become an important design concern because it directly affects the manufacturing cost.

Clock designs have significant impacts on both timing convergence and timing yield. A carefully designed clock distribution network can reduce *design-inherited* clock skews, the discrepancies between designer intended clock skews and achieved clock skews under perfect process conditions. This can improve circuit performance and timing convergence. A clock distribution network can also be optimized to reduce *process-induced* clock skews, such that the circuit would require less timing margin to tolerate process related timing variations. This will also improve timing convergence and timing yield. *Clock*

scheduling, the process of assigning the clock arrival times to sequential elements, can greatly improve the timing yield of the manufactured chips by taking into account process-induced path delay and clock skew uncertainties.

This dissertation provides a comprehensive study on four clock design techniques: 1) clock tree optimization, 2) clock scheduling, 3) clock tree topology optimization, and 4) post-silicon clock tuning. The integration of these techniques offerS the maximum benefit on timing convergence and timing yield improvements. A zero-skew clock tree optimization algorithm for clock delay and power optimization is proposed. The optimized clock trees are zero-skew, or free from design-inherited clock skews, and their process-induced clock skews are reduced by clock delay minimization. A false-path-aware statistical timing analysis method using a novel implicit true path enumeration algorithm is proposed. A statistical-timing-driven clock scheduling algorithm then utilizes the statistical timing information for timing yield improvement. False-path-aware gate-sizing methods are also investigated to preserve more timing margin for clock scheduling. A partition-based clock tree topology optimization algorithm that considers the circuit connectivity and timing information is proposed. The clock trees based on the new topologies require less timing margin for process-induced clock skews, which makes the timing convergence easier to achieve. Timing yield can be further improved if clock arrival timing assignment can be performed for each manufactured chip separately. This can be achieved by using a post-silicon-tunable clock tree. Two post-silicon-tunable clock tree synthesis algorithms are proposed to reduce the hardware cost to realize post-silicon clock tuning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In nanometer technologies, timing convergence and timing yield are two of the most critical design issues. Due to the high device density and complicated physical design effects, a small change in the design for timing fixes can cause new timing violations and result in design iterations. With the accelerating time-to-market cycles, timing convergence has become the main focus in the design flow. Processes in nanometer technologies are difficult to control. Process variations cause timing variations in manufactured chips, causing failures and chips not meeting the performance guaranteed by design specifications. Because of the high design and manufacturing costs, a small percentage of the timing yield loss can turn the design from a money-spinner into a profit-loser. Therefore, it is paramount to address the timing convergence and timing yield issues.

Timing convergence and timing yield issues have been aggravated by the following four factors: 1) technology scaling, 2) process variations, 3) deep pipelining, and 4) ever increasing clock frequency. The

reasonings are elaborated as follows:

1. As the feature size continues to shrink, gate delays also decrease substantially. However, interconnect delays do not scale as well as gate delays and have become the dominant delay component [1, 2]. Since a design change for a path usually involves change of routings[1], which in turn affects the parasitic couplings, it can easily cause delay changes on other paths, generating new timing violations and resulting in design iterations. Copper interconnects with sub-100nm widths suffer resistivity increase due to electron-scattering effects [3, 4], which increase the RC interconnect delay even further. It is expected that timing convergence in nanometer technologies will be increasingly more difficult to achieve.

2. Process variations are getting difficult to control due to the limits of physics. For example, a gate oxide in 90nm technologies is only a few atoms thick[2] and a variation as small as a single atom can amount to a change in thickness of several percents and thus the delay of the transistor. According to the 2003 ITRS Roadmap [5], lithography needs to achieve a 3σ gate critical-dimension (CD) variation less than $1.8nm$ for $\geq 50nm$ technologies by 2009. However, there is no known solution to-date. The increasing process-related timing uncertainties need to be accounted for by reserving more timing margins, which slows down timing convergence. Cutting back timing margins to satisfy time-to-market results in low timing yield designs.

[1]Inserting a buffer on a long wire usually cause routing detours because available white spaces for the buffer may not be directly under the wire. Even pure gate sizing can cause routing changes because the pin locations of gates with different drive strength can be different.

[2]The equivalent oxide thickness (EOT) of the $hp90$ technology node is $1.2nm$ [5]. It is about four atom layers if thermal silicon dioxide (S_iO_2, $k = 3.9$) is used, and about seven atom layers if silicon nitride (Si_3N_4, $k = 7$) is used.

3. To allow fine-grained circuit partitioning for performance improvement, high-performance designs such as microprocessors have been increasing their pipeline stages. As a result, the average number of gate levels between a pair of sequential elements decreases. With fewer gate levels, the combinational path delay uncertainties increase substantially relative to the path delays [6]. For example, let the gate delays be independent Gaussian distributions (μ, σ), the delay distribution of a path with N gates is $(N\mu, \sqrt{N}\sigma)$. Therefore, the ratio of path delay uncertainty versus path delay is proportional to $\frac{1}{\sqrt{N}}$, which increases as N decreases. This further aggravates the timing variation problem.

4. Design iterations and timing yield loss can be avoided if there are sufficient timing margins to tolerate timing variations. Nevertheless, the increasing demand for high clock frequency leaves the designers with less timing margins.

From the above observations, a comprehensive solution to reducing timing uncertainties and better utilizing timing margins is in urgent need.

While clock delays increase as the total number of sequential elements increase, combinational path delays are decreasing due to deep pipelining. Therefore, the ratio of clock delay versus combinational path delay keeps increasing. As a consequence, a significant portion of the timing margins is consumed by process-induced clock skews. By optimizing the clock distribution networks, the process-induced clock skews and the demand for timing margins can be reduced. Timing yield is highly correlated to the distribution of timing margins. If all the paths in a circuit except one have sufficient timing margins, the timing yield can still be very low. Therefore, carefully managing timing margins through assigning the clock arrival times to sequential elements can greatly improve the timing yield. In the next section, the related clock design researches that address the timing convergence and timing yield issues are reviewed.

1.2 Literature Review

1.2.1 Clock Tree Optimization

Clock tree is the most commonly used structure for clock distribution networks. Although clock meshes or clock grids are used in some microprocessor designs [7–10], they are less preferred due to the high routing cost and clock power. The recently announced Intel dual-core Itanium® processor has re-adopted clock tree because of the power consideration [11].

A clock tree is synthesized from the positions and capacitance loads of the clock sinks. The first step is topology generation through partitioning the clock sinks. It is then followed by routing and optimization steps. Tsay [12] proposes a zero-skew clock routing algorithm for a given clock tree topology. For a wire connecting two clock sinks, the algorithm finds the tapping point on the wire that has the same clock delay to both clock sinks. The tapping point becomes a new clock sink and the algorithm repeats in a bottom-up fashion until it reaches the clock root. The total wire length of the final clock tree is dependent on the clock tree topology and the routing of each wire that connects two clock sinks. Chao et al. [13] propose a balanced-bipartition (BB) algorithm that generates the clock tree topology and the Deferred-Merge-Embedding (DME) algorithm that determines the routing of each wire. The combined BB+DME algorithm achieves a 10% average wire length savings comparing to [12]. In [14,15], the DME algorithm is extended to handle bounded-skew and useful-skew constraints. The objective of these works aims at minimizing the total wire length of a clock tree, which in turn minimizes the power consumption.

Despite the power advantage of clock trees over clock grids, clock trees are more susceptible to process variations than clock grids. However, process-induced clock skews are not known until the clock tree implementation is done. To reduce the susceptibility of a clock tree to process variations, a metric to predict process-induced clock skews is needed. Empirically, it is observed that process-induced clock

skews can reach 10% of the clock delay [16]. Therefore, minimizing clock delays can reduce process-induced clock skews.

Interconnect delay optimization techniques, which include buffer insertion, buffer sizing and wire sizing, are applicable to clock delay optimizations. The difference between synthesizing a signal net and a clock tree is that for a signal net, the design objective is to minimize the maximum delay from the signal source to any of its signal receivers, whereas a clock signal needs to be distributed to each clock sink at the prescribed time.

Two excellent surveys of interconnect optimization techniques are available in [1, 2]. A significant amount of works are based on dynamic programming. Van Ginneken [17] proposes a dynamic programming algorithm to solve the buffer placement problem. Given a distributed RC-tree, a buffer library, and the legal buffer positions, the problem aims at finding the buffering option that minimizes the maximum delay from the root of the RC-tree to any of its leaf nodes. In [18], discrete wire sizing for general routing trees is assumed and a bottom-up dynamic programming approach is used to propagate the optimal wire sizing options toward the root node. Designers can then choose an option by making a tradeoff between delay and power consumption. In [19, 20], bottom-up dynamic programming algorithms are extended to consider simultaneous buffer insertion and wire sizing techniques. The DME algorithm for zero-skew clock routing is also a dynamic programming algorithm, in which candidate zero-skew tapping points are stored as merging segments. Extended works [21–23] allow a zero-skew clock tree to achieve better clock delay and power by buffer insertion and wire sizing.

In [24], a sensitivity-based iterative algorithm performs wire sizing one segment at a time and about 1.5X to 3X improvements on minimum delay are observed. In [25–27], the simultaneous buffer insertion, buffer sizing and wire sizing problems are formulated as optimization problems, in which the maximum

delay of each sink node is constrained. Iterative and Lagrangian Relaxation methods are then used to solve the problems. In [28], iterative algorithms are used to reduce clock delay and clock skew through wire sizing based on sensitivity information. Zeng et al. [29] propose a three-stage optimization algorithm, i.e., buffer insertion, delay optimization and skew optimization, to minimize the delay and skew of a clock tree. Compared with the initial unbuffered clock trees, a maximum of 27X delay improvement is achieved by buffer insertion and sizing. Chen et al. [30] formulate the clock delay/power/area minimization problem for buffered clock trees as a geometric programming problem and solve it using Lagrangian Relaxation methods.

The existing clock tree optimization works, either dynamic-programming-based or mathematical-programming-based, either only utilize one or two available optimization techniques, i.e., buffer insertion, buffer sizing and wire sizing, or consider them in separate stages. This limits the full advantage of the available techniques. There is a need to develop an optimization algorithm that considers buffer insertion, buffer sizing and wire sizing simultaneously.

1.2.2 Clock Scheduling

In a zero-skew design, the clock period is determined by the longest path delay of the circuit. To increase the operation frequency, several techniques such as circuit retiming and pipelining are usually adopted to balance path delays at different parts of the circuit [31, 32]. Since path delays usually cannot be perfectly balanced, clock scheduling is applied to further optimize the clock period [33–40].

As shown in [33], the timing constraints for flip-flop-based circuits are linear constraints and the clock period optimization problem can be solved by linear programming solvers. Deokar et al. [34] use a graph-theoretic approach to solve the optimization problem. First, the timing graph of a circuit is

constructed and the timing constraints are modeled as the parametric edge costs, which change according to the clock period. A clock period is feasible if the timing graph contains no negative cost cycle. The optimal clock period can be obtained through a binary search between $[0, D_{max}]$, where D_{max} is the maximum combinational path delay, by applying the Bellman-Ford algorithm [41, 42] on the timing graph. An alternative is to solve the parametric shortest path problem by a path-pivoting algorithm [43].

The optimal clock period and the clock schedule found by solving the linear program is not directly applicable to real designs since there are some paths with zero slack [3]. To make the design more robust, a slightly larger clock period than the optimal value is chosen and the slacks on every path are then optimized by clock scheduling, introducing useful clock skews to the sequential elements. Neves et al. [35] and Kourtev et al. [36] formulate the clock skew optimization problem as a least square error problem where the error is defined as the difference between the skew and the middle point of the permissible range. Albrecht et al. [39] adopt the minimum balance algorithm [43] to distribute the timing margins so that it yields a *lexicographically maximum* slack vector when the slacks are sorted in nondecreasing order. However, these approaches do not take into consideration the statistical behavior of process variations and also their results are not verified with any timing yield model.

It is observed that many structural paths in a circuit are functionally un-sensitizable paths, or false paths [44–47]. Traditionally, the clock scheduling algorithms obtain the longest and shortest path delays by using static timing analysis tools that usually do not consider false paths. If all the structural longest (shortest) paths between some pairs of sequential elements are false paths, the clock schedule obtained based on this pessimistic expectation of path delays will be sub-optimal. Furthermore, false paths can affect the statistical path delay distributions significantly [48], thus affecting the decision for slack allo-

[3] Slack and timing margin are interchangeable.

cation. Therefore, it is necessary to perform false-path-aware statistical timing analysis as the first step of clock scheduling. Although statistical timing analysis techniques have been studied for more than a decade [49, 50], only a few works take false paths into consideration [48, 51] and they are not suitable for clock scheduling applications. There is a need to develop an efficient false-path-aware statistical timing analysis algorithm and a statistical-timing-driven clock scheduling algorithm.

1.2.3 Clock Tree Topology Optimization

Traditional clock tree topology generation algorithms aim at finding a clock tree topology such that zero-skew can be achieved with minimum wire length. Kahng et al. [52] and Edahiro [53] propose clustering-based algorithms that pair up clock sinks recursively to form the clock tree topology. In both works nearest-neighbor matching heuristics are used for clustering. Chao et al. [13] propose a balanced-bipartition (BB) algorithm that generates a balanced clock tree topology through recursive bipartitioning the set of clock sinks into balanced subsets. Chou and Cheng [54] use simulated annealing to explore different clock tree topologies and select the topology with the best cost; the weighted sum of wire length and clock delay. These works do not consider the effects of clock buffers.

Ellis et al. [55] propose a simulated-annealing-based clock tree topology selection algorithm. For local clock distribution, clock sinks are clustered based on the near neighbor information obtained from a Delaunay triangulation of the clock sinks. Simulated annealing is used to find the clustering that minimizes the wire length. For global clock distribution, the binary clock tree topology is perturbed and a new topology is accepted or rejected based on the annealing temperature and the figure of merit from a fast buffer insertion and wire sizing heuristic.

Liu et al. [56] point out that the interconnect variation effect on clock skews cannot be captured by

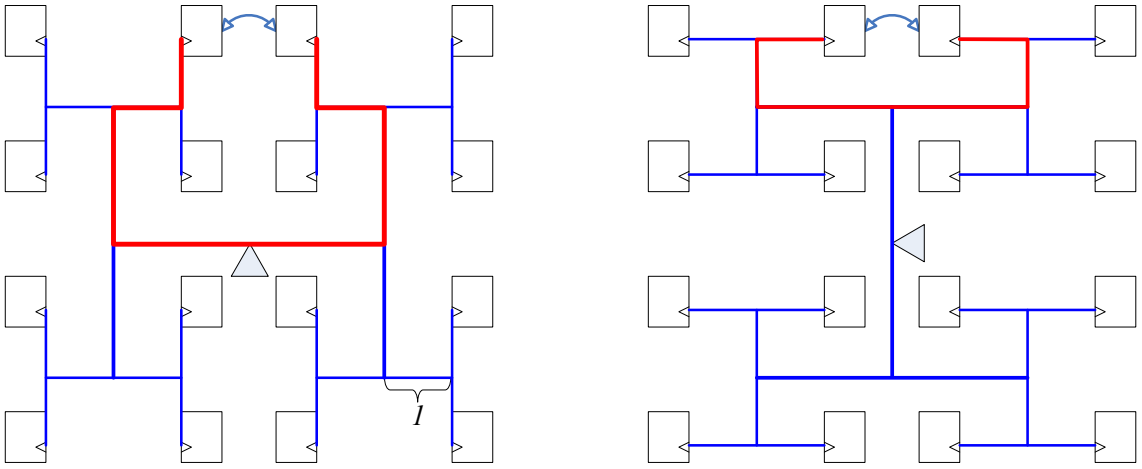


Figure 1.1: Two clock tree topologies for the same set of clock sinks.

worst/best case corner point methods because it is dependent on the physical design of the clock tree. Since clock trees are interconnect-dominated circuit structures, process-induced clock skews caused by interconnect variations are significant. It is observed that the maximum process-induced clock skew of a clock tree is positively correlated to its clock delay. Previous works on clock delay minimization reduce the maximum process-induced clock skew using combinations of routing, buffer insertion, buffer sizing and wire sizing techniques. Although these works reduce the maximum process-induced clock skew, they do not necessarily minimize the total clock skew uncertainty on timing-critical paths and the total slack requirement of a design. This is because these works all start from a given clock tree topology that is generated without considering the locations of timing-critical paths.

Figure 1.1 shows two clock trees for the same set of clock sinks from two different clock tree topologies. The arc in Figure 1.1 indicates the most timing-critical path of the circuit. Although the two clock trees have the same maximum process-induced clock skew (assuming clock trees are symmetric and

process variations are uniform), the clock tree in Figure 1.1(a) requires more timing margin to achieve timing convergence because the long non-common clock distribution path can cause a large process-induced clock skew on the already timingly critical path. Hence the clock tree topology in Figure 1.1(b) is preferred.

One of the methods to reduce the variation impacts to clock is link insertion after clock tree construction [57]. By connecting two nodes in a clock tree with an additional wire, the process-induced clock skew between them may be reduced. However, this method still relies on a good clock tree topology as a good topology can reduce the number of required links. Velenis et al. [58, 59] propose a greedy clustering-based algorithm to reduce process-induced clock skews on timing-critical paths. The algorithm creates the clock tree topology by first clustering the source and target clock sinks of the most timing-critical path together, then clustering the two clock sinks of the second most timing-critical paths, and so on so forth. However, the clustering-based algorithm lacks the global view and has practical limitations. For example, the algorithm will cluster the two clock sinks of the most timing-critical path together even if the two clock sinks are located at the opposite corners of the chip.

To reduce the timing margin requirement and improve the timing convergence, a clock tree topology optimization algorithm that takes timing information as well as physical location of the clock sinks needs to be developed.

1.2.4 Post-Silicon Clock Tuning

Process related timing variation is a serious problem in nanometer designs. Simulation results based on a $180nm$ technology show that gate CD variation can result in path delay and clock skew variations as high as 16% and 8% of the clock period [60]. Existing design methodologies that guarantee a satisfactory

timing yield by reserving more timing margins for larger timing uncertainties will become impractical in new technologies due to performance and time-to-market requirements. However, insufficient timing margins can cause significant timing yield losses. Although timing yield loss is *recoverable* by reducing the sensitivity of the circuit to process variations, the growing intensity of process variations in nanometer technologies, stringent time-to-market requirements, and limits on non-recurring-engineering (NRE) cost have made it difficult to add timing yield as part of the design objectives during circuit optimization steps. It is favorable to have generic design-for-yield techniques that can be applied to different designs and have the least impact on the current design flow.

Rajaram et al. [57] propose to reduce clock skew uncertainties by inserting *cross links* in a given clock tree. However, this technique cannot take path delay uncertainties into account. Another promising design-for-yield technique is to use post-silicon-tunable (PST) clock trees [9, 11, 61–65]. By inserting tunable clock buffers into the clock tree, slacks can be redistributed among adjacent timing paths and timing failures may be corrected through *post-silicon clock tuning*.

As shown in Figure 1.2, the clock distribution network of Intel's recently announced dual-core Itanium® processor uses two levels of PST clock buffers to counter clock skews caused by process variations and improve the timing yield [11]. The tunable second level clock buffers (SLCBs) at the terminals of L1 route can be dynamically adjusted by on-chip clock phase detection hardware to cancel clock skew variations. They can also be programmed from the test access port (TAP) for timing optimization. There is also a second level of PST clock buffers at every terminal of the L2 route. This level consists $\sim 15K$ clock vernier devices (CVDs) for clock fine-tuning through scan.

Takahashi et al. [64] propose a post-silicon clock timing adjustment method that adjusts the clock arrival times to compensate for path delay variations. The idea is similar to that in [9, 11, 61–63], except

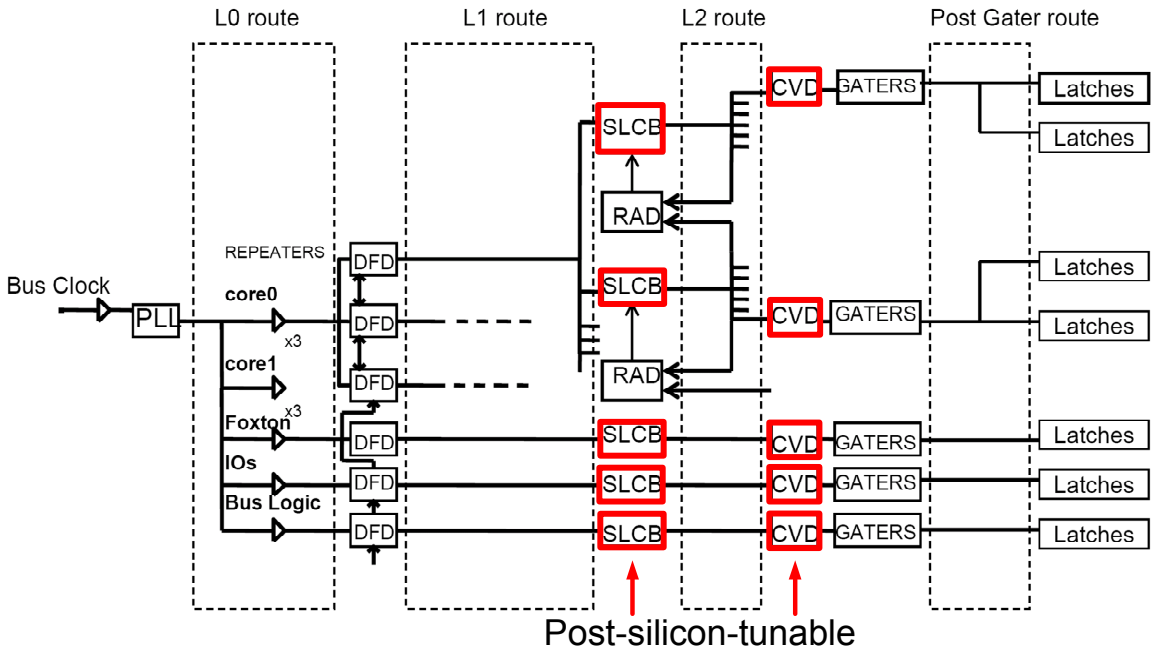


Figure 1.2: Clock distribution network of a dual-core Intel® Itanium® processor (Source: Intel).

that the authors propose to put PST clock buffers at the clock input pins of selected sequential elements or design modules instead of a brute-force design method that inserts a PST clock buffer for each sequential element or at every clock tree terminals at certain levels.

Post-silicon clock tuning not only improves the timing yield but also reduces clock power by avoiding the use of grid-based clock distribution networks. However, there is no systematic way to construct a PST clock tree that *provides the maximum tuning capability for timing yield improvements with minimum hardware cost*. Therefore, the hardware overhead has limited the use of PST clock trees in high end microprocessors. To make PST clock tree an attractive design-for-yield choice, it is essential to develop PST clock tree synthesis algorithms for hardware cost reduction.

1.3 Organization of the Dissertation

This dissertation provides new solutions to the timing convergence and timing yield issues through advancements on four clock design techniques: clock tree optimization, clock scheduling, clock tree topology optimization and post-silicon clock tuning. First, improvements are proposed on current clock design flows by considering timing uncertainties in the early design stage. Studies on post-silicon clock tuning, an emerging technique to tackle timing convergence and timing yield issues, are then presented.

Chapter 2 begins with clock tree optimization, the core of current clock design flows. A novel zero-skew clock tree optimization algorithm [66, 67] for clock delay minimization is proposed. The optimized clock trees are free from design-inherited clock skews and allow faster timing convergence due to the reduced process-induced clock skews.

The clock arrival time constraints are generated by clock scheduling. Although clock scheduling has been well-studied in the context of performance improvement, little research has been done on applying clock scheduling for timing yield improvement. Chapter 3 presents a new clock scheduling scheme that combines a novel false-path-aware statistical timing analysis method with a fast clock scheduling algorithm [68, 69]. A false-path-aware gate-sizing method, which preserves more timing margins for clock scheduling, is also investigated [70]. The new clock scheduling scheme achieves significantly better timing yields through better timing margin utilization.

Clock tree optimization algorithms usually start from a given clock tree topology. Therefore, timing convergence is affected by the choice of clock tree topology. In Chapter 4, a new clock tree topology optimization algorithm is proposed. The new algorithm takes into consideration both the estimations of path delay and clock skew uncertainties, and sequential element locations. The clock trees based on the optimized clock tree topologies require significantly less timing margins to tolerate process related

timing variations, which speeds up the timing convergence.

The current correct-by-construction clock design philosophy will eventually make the design process both difficult and overlong in nanometer technologies due to the increasing timing uncertainties. The PST clock tree, a clock structure that can adapt itself in response to timing variations, is a versatile solution to the timing challenges in new technologies. In Chapter 5, two PST clock tree synthesis algorithms are proposed. The algorithms insert PST clock buffers only at the critical locations in a clock tree and greatly reduce the hardware cost compared to current design methods.

Figure 1.3 demonstrates the integration of proposed clock design techniques in this dissertation. Chapter 6 summarizes the contributions of the dissertation and presents future research directions.

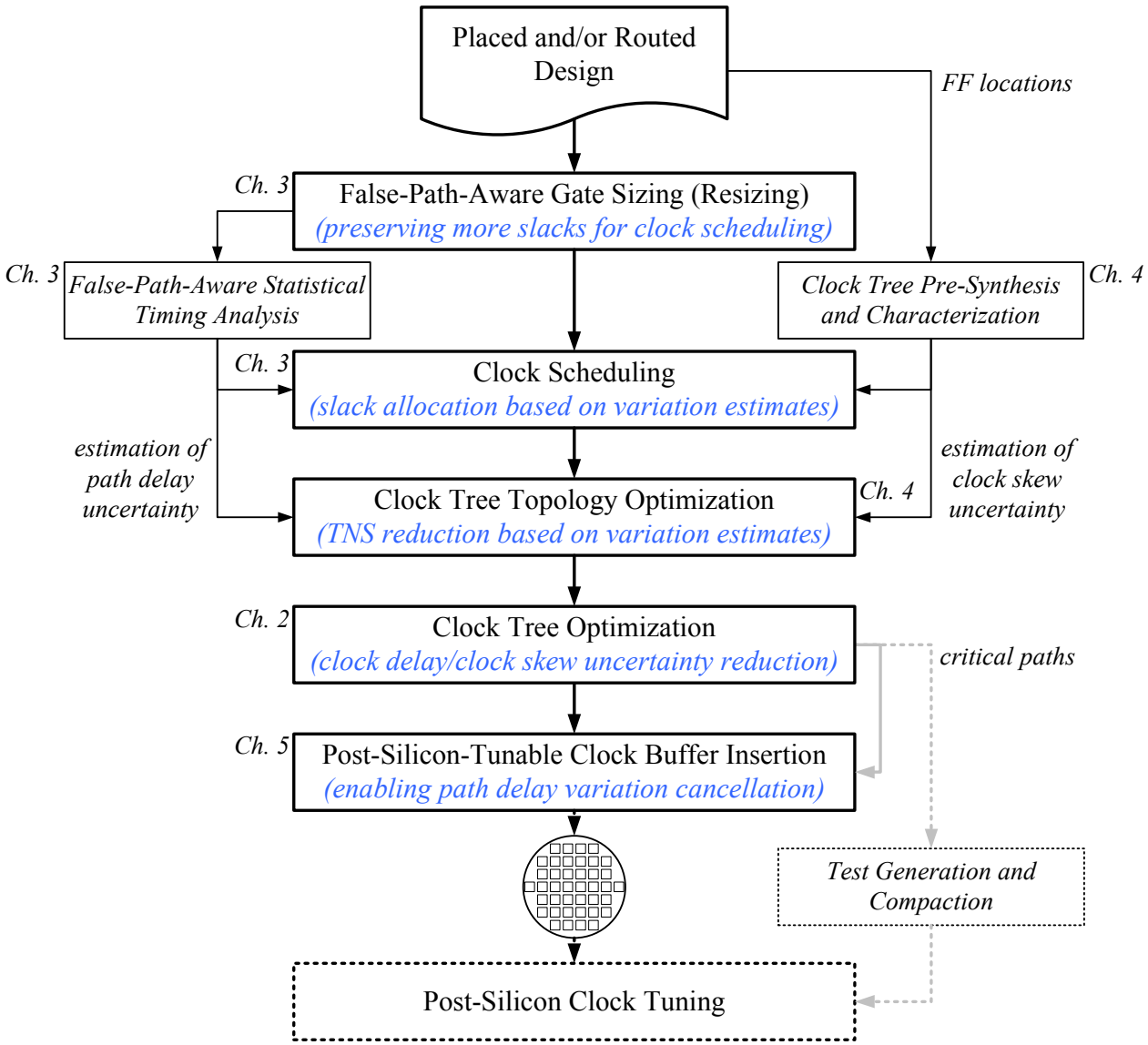


Figure 1.3: The integration of proposed clock design techniques.

Chapter 2

Zero-Skew Clock Tree Optimization

Clock delay and clock power are two of the most important clock tree optimization objectives. It is observed in [16] that process-induced clock skews can reach 10% of the clock delay. By reducing the clock delay, process-induced clock skews in the manufactured chips can also be minimized. Clock trees can easily consume over 30% of the total power due to their high switching activities [8,63]. Therefore, it is also very important to reduce the total capacitance of a clock tree in order to control the chip power. The techniques for clock delay and power minimization include buffer insertion, buffer sizing and wire sizing. In this chapter, a novel zero-skew clock tree optimization algorithm that applies all three techniques simultaneously is proposed. The algorithm is based on a two-phase optimization framework adopted by van Ginneken's algorithm [17]. First, the delay and power models and van Ginneken's algorithm are reviewed. A novel design space representation approach is presented and illustrated with a simplified clock tree optimization problem considering only wire sizing. Details on extending the algorithm for simultaneous buffer insertion, buffer sizing and wire sizing are then presented. Finally, the effectiveness of the algorithm is demonstrated by the experimental results.

2.1 Preliminaries

2.1.1 Delay and Power Models

Interconnect delay and buffer delay are the two delay components in a clock tree. Interconnects and buffers are modeled with the resistance-capacitance (RC) model and their delays with the Elmore delay model [71]. A clock tree with a given routing rooted at node v is denoted by T_v. For a wire with length l and width w, the wire resistance is $\frac{lr_0}{w}$ and the wire capacitance is lc_0w, where r_0 and c_0 are the resistance and capacitance of a $1\mu m^2$ wire. The wire capacitance is modeled as two equal capacitors attached to both ends of the wire. For a buffer with gate width w_b, the gate capacitance at its input is w_bc_b and its effective output resistance is $\frac{r_b}{w_b}$, where c_b and r_b are the unit-width gate capacitance and resistance. The buffer is modeled as a ramp voltage source with an intrinsic delay of t_c.

The clock tree power consumption is generally modeled as $P = fCV^2 + P_s + P_l$, where f is the switching frequency, V is the voltage swing, C is the sum of all interconnect capacitance, buffer gate capacitance, and sink loads. P_s accounts for the buffer short-circuit power and P_l accounts for the leakage power. It is observed that P_s is less than 10% in proper designs [72–74]. Leakage power is a growing component in nanometer designs and consumes about 50% of the total chip power in $65nm$ technologies [75]. Leakage power reduction is usually done by system-wide techniques such as using high-k dielectric for gate oxide, multiple-threshold CMOS, sleep transistor insertion, etc. [76]. Moreover, the switching factor of the clock tree is much higher than the rest of the chip, making P_l over P in a clock tree much smaller than 50%. Therefore, the switching power $P = fCV^2$ alone is a reasonable estimate of the total clock tree power.

2.1.2 Van Ginneken's Algorithm

Van Ginneken's algorithm is proposed to solve the buffer placement problem. Given a distributed RC-tree, a buffer library, and a set of legal buffer positions, the goal is to find a buffered RC-tree that minimizes the maximum delay from the root node to all leaf nodes. The algorithm consists of a bottom-up and a top-down phase. During the bottom-up phase, the combinations of total downstream capacitance (c_v) and longest delay (D_v) from the root (v) of a tree branch to its leaf nodes are recorded in 2-tuples, (c_v, D_v). When two branches merge together, the cross-products of the 2-tuple sets from both sides are generated. The total downstream capacitance in a new 2-tuple is the sum of the capacitances from both branches. The longest delay is the maximum delay of both branches. For each delay value, only the tuple that has the smallest capacitance value is kept. For a buffer library with B buffer types, there are B buffering options at each legal buffer position. Each option is represented by a 3-tuple, (b, C_b, $D_v + R_b \times c_v$), in which R_b is the buffer resistance and C_b is the input capacitance of a type b buffer. For each buffer type, (c_v, D_v) is chosen such that $D_v + R_b \times c_v$ yields the smallest delay. Non-buffered options are represented by (ϕ, c_v, D_v).

Figure 2.1 shows the bottom-up phase of van Ginneken's algorithm. To simplify the example, it is assumed that there is only one buffer type and legal buffer positions are right before the branching points of the tree. All values are equal to one unless otherwise noted. Figure 2.2 shows the top-down phase of van Ginneken's algorithm.

There are two major limitations of van Ginneken's algorithm when applied to clock tree optimizations. First, the algorithm only minimizes the maximum clock delay and it may result in large clock skews. Second, if the van Ginneken's algorithm is modified such that clock skews are propagated along with clock delays, multiple buffered options for each buffer type need to be kept at each legal buffer

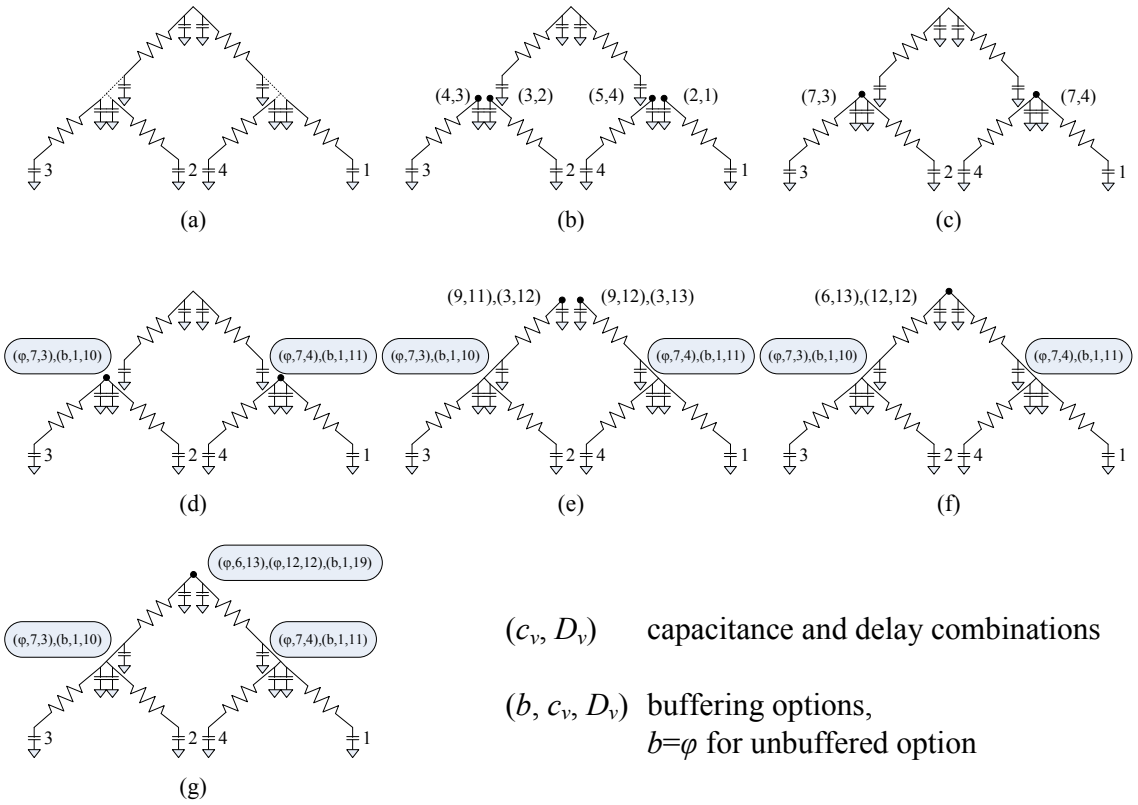


Figure 2.1: The bottom-up phase of van Ginneken's algorithm.

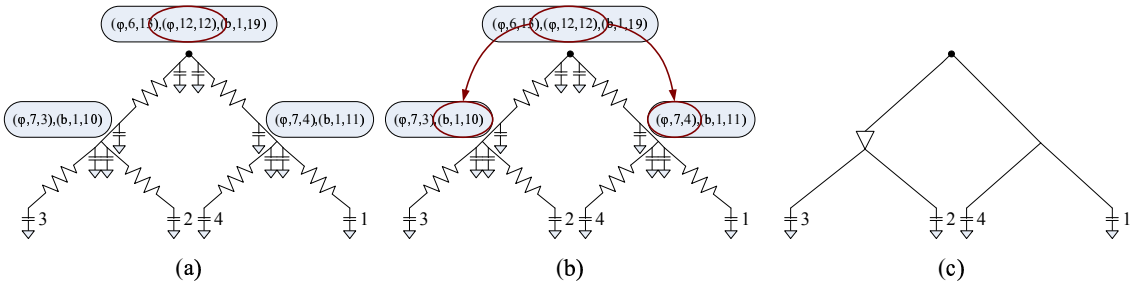
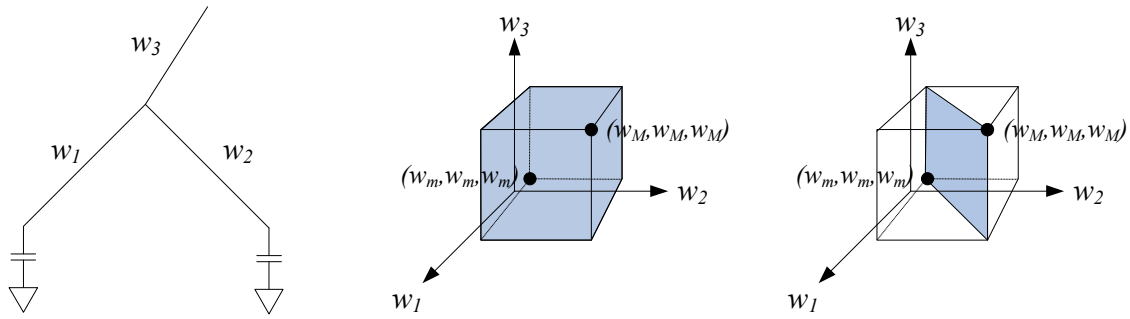


Figure 2.2: The top-down phase of van Ginneken's algorithm.



(a) clock tree (b) design space (c) zero-skew design space

Figure 2.3: The design space concept.

position. This will result in combinatorial explosion.

2.2 Zero-Skew Clock Tree Optimization with Wire Sizing

2.2.1 Design Space Concept

Figure 2.3(a) shows a small clock tree with three wire segments. Assuming the range of wire width is (w_m, w_M), the design space of the clock tree is a cube as shown in Figure 2.3(b). It is clear that not every point in the design space is a zero-skew solution. If the two branches connecting to the clock sinks have equal length and the two clock sinks have equal capacitance, the zero-skew solution space is a square region as shown in Figure 2.3(c). The set of wire widths, (w_1, w_2, w_3), that yields the lowest clock power is (w_m, w_m, w_m) since it gives the lowest wire capacitance over the entire zero-skew design space. However, it is not easy to see which wire sizing gives the lowest clock delay. For a clock tree with n wire segments, the design space is a hypercube in an n-dimensional space.

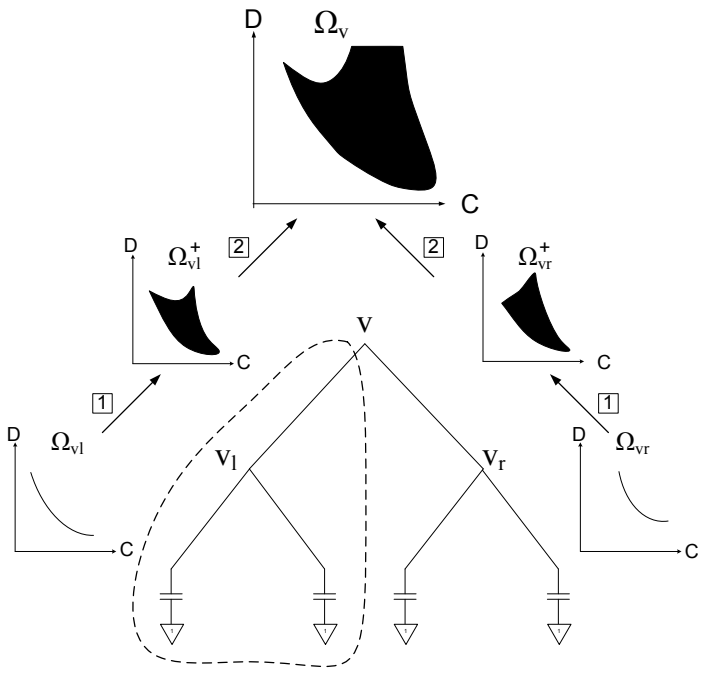


Figure 2.4: The DC regions of the subtrees in a clock tree T_v.

2.2.2 The DC Region Approach

In the Elmore delay model each subtree is characterized by the total downstream capacitance at its root node. For a zero-skew clock tree T_v, each set of wire widths, or an *embedding*, determines a pair of clock delay and capacitance values (d_v, c_v). The whole zero-skew design space of T_v can be represented by the *DC region* Ω_v on the *DC plane*, an X-Y plane where the coordinates of the Y-axis and the X-axis are the clock delay and capacitance values. Figure 2.4 shows the DC regions of T_v, T_{v_l}, and T_{v_r} considering only wire sizing. The DC region of a level-1 node such as Ω_{v_l} is a curve and the DC regions of higher level nodes have complex shapes. The minimum delay and minimum power achievable by wire sizing can be observed from the DC regions. The Y-coordinate of the bottom-most point in a DC region is the minimum delay and the X-coordinate of the left-most point in a DC region is the minimum power.

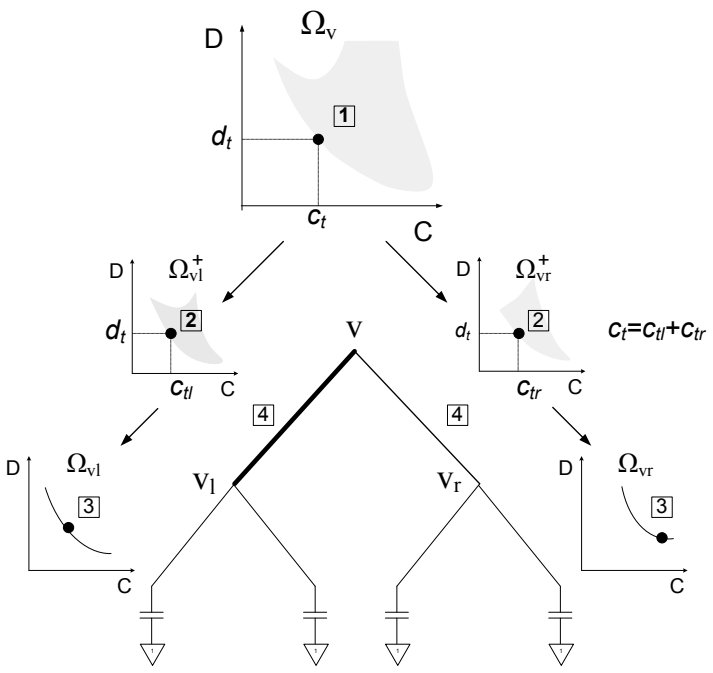


Figure 2.5: Illustration of the embedding selection steps.

A clock tree is composed of two *branches*. The left branch of T_v is enclosed in the dashed circle in Figure 2.4 and is denoted as $T_{v_l}^+$. The DC region of $T_{v_l}^+$, a *branch DC region*, is denoted as $\Omega_{v_l}^+$ and it can be obtained by applying a transformation on Ω_{v_l}. The zero-skew constraint requires the delays along both branches to be the same, thus Ω_v can be obtained by an equi-delay merge operation on $\Omega_{v_l}^+$ and $\Omega_{v_r}^+$. Therefore, the DC region of a clock tree can be constructed in a bottom-up fashion recursively with two major steps. First, the branch DC regions are obtained by transforming the DC regions of the left and right subtrees. Second, the DC region is obtained by merging the branch DC regions.

Generating embeddings from the DC regions is an inverse process. The process is illustrated in Figure 2.5. First the target clock delay and capacitance, d_t and c_t, are chosen from the DC region of the root node of the clock tree. The capacitance is then split into two branches. The DC regions of the left and right subtrees are scanned and feasible target clock delays and capacitances of the subtrees are

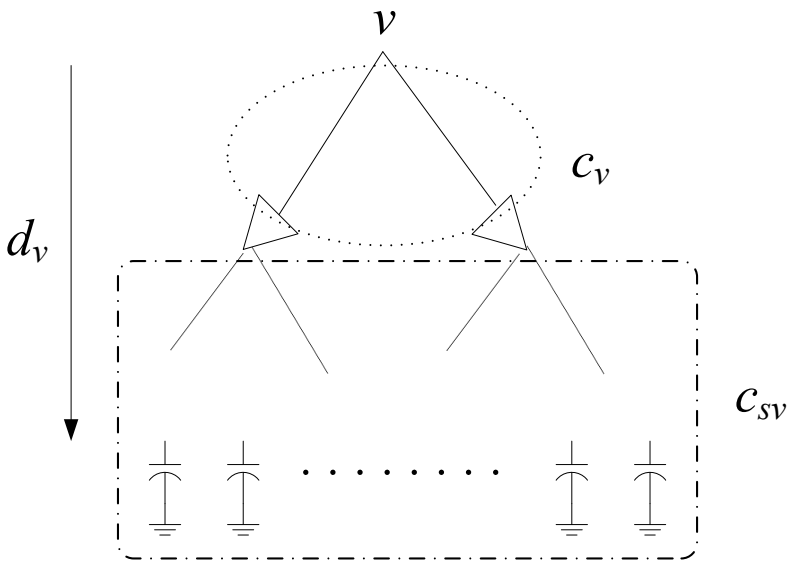


Figure 2.6: Characterize a buffered clock tree T_v with three parameters.

selected. From the selected points in the branch DC regions and the subtree DC regions, the wire widths above the subtrees can be determined.

The use of DC regions avoids the need of directly handling high dimensional design spaces. By propagating DC regions, all zero-skew solutions are preserved. Since DC regions have irregular shapes, they are stored as a set of horizontal segments.

2.3 Simultaneous Buffer Insertion, Buffer Sizing and Wire Sizing

2.3.1 DC Regions of Buffered Clock Trees

A buffered clock tree T_v can be characterized by the three parameters as shown in Figure 2.6. The first two parameters, d_v and c_v, are the clock delay and the capacitance seen at the root node v. The total switching capacitance of T_v is the sum of c_v and c_{sv}, the capacitance shielded from v by the buffers.

Thus the zero-skew design space can be represented with a three-dimensional DC region, where the coordinates of the Z-axis, X-axis, and Y-axis are the clock delay, capacitance seen at the root node, and the shielded capacitance. The major steps for handling three-dimensional DC regions are highlighted in the following sections.

2.3.2 Branch DC Regions and Projected Scan-Line Sampling

When there is no buffer inserted above the root of the subtree, only the effects of wire sizing from the wire segment connecting the root of the branch to the root of the subtree need to be considered. To capture the irregular shape of a branch DC region, sampling on wire-width is applied and the intersection of the branch DC region and a set of clock delay scan-lines are obtained. The branch DC region can then be represented by a set of horizontal segments. Since the branch DC region is the projection of a zero-skew design space, the project-sample-scan procedure is referred to as *projected scan-line sampling*. The transformation is defined by the following equations obtained directly from the Elmore delay model.

Wire Sizing Transformation:

$$d_v^+ = d_v + \frac{l_v r_0}{w_e(v)} \left(\frac{l_v w_e(v) c_0}{2} + c_v \right) \tag{2.1}$$

$$c_v^+ = c_v + l_v w_e(v) c_0, \tag{2.2}$$

$$\forall\, (d_v, c_v) \in \Omega_v.$$

The edge connecting v to its parent node is e_v and the length of e_v is l_v. The wire-width of e_v is $w_e(v)$ and $w_m \leq w_e(v) \leq w_M$. Since wire sizing does not affect c_{sv}, $c_{sv}^+ = c_{sv}$ and a three-dimensional sampled branch DC region can be obtained by projected scan-line sampling. It is denoted by

$$\Omega_v^+ = \mathbb{W}_v(\Omega_v).$$

If a buffer is inserted above a subtree, the effects of both wire sizing and buffer sizing need to be considered. Therefore, the transformation is defined by different equations.

Buffer Insertion Transformation:

$$d_v^+ = d_v + t_c + \frac{l_v^2 r_0 c_0}{2} + \frac{l_v r_0 c_b w_b(v)}{w_e(v)} + \frac{r_b c_v}{w_b(v)} \tag{2.3}$$

$$c_v^+ = c_b w_b(v) + l_v w_e(v) c_0, \tag{2.4}$$

$$c_{sv}^+ = c_{sv} + c_v, \tag{2.5}$$

$$\forall\ (d_v, c_v, c_{sv}) \in \Omega_v.$$

Equation (2.3) defines the difference of clock delays between a branch and its subtree, which is composed of the buffer intrinsic delay, the delay caused by wire and buffer capacitance, and the buffer delay by driving the shielded capacitance. The width of the buffer above v is $w_b(v)$. Equations (2.4) (2.5) define the change of the capacitance seen from the root and the shielded capacitance of a branch. Samplings are applied on buffer-width and shielded capacitance, and projected scan-line sampling is used to obtain the sampled DC regions with a set of horizontal segments along the X-axis. It is denoted by

$$\Omega_v^+ = \mathbb{B}_v(\Omega_v).$$

The equi-delay merge takes one segment from each sampled branch DC region with the same clock delay and generate a new segment where $c_v = c_{v_l} + c_{v_r}$ and $c_{sv} = c_{sv_l}^+ + c_{sv_r}^+$. The merge operator is denoted as $\curlywedge$ and the merge operation is written as

$$\Omega_v \leftarrow \Omega_{v_l}^+ \curlywedge \Omega_{v_r}^+.$$

Figure 2.7 shows the steps of projected scan-line sampling and equi-delay merge in obtaining the sampled DC region of an unbuffered two-level clock tree.

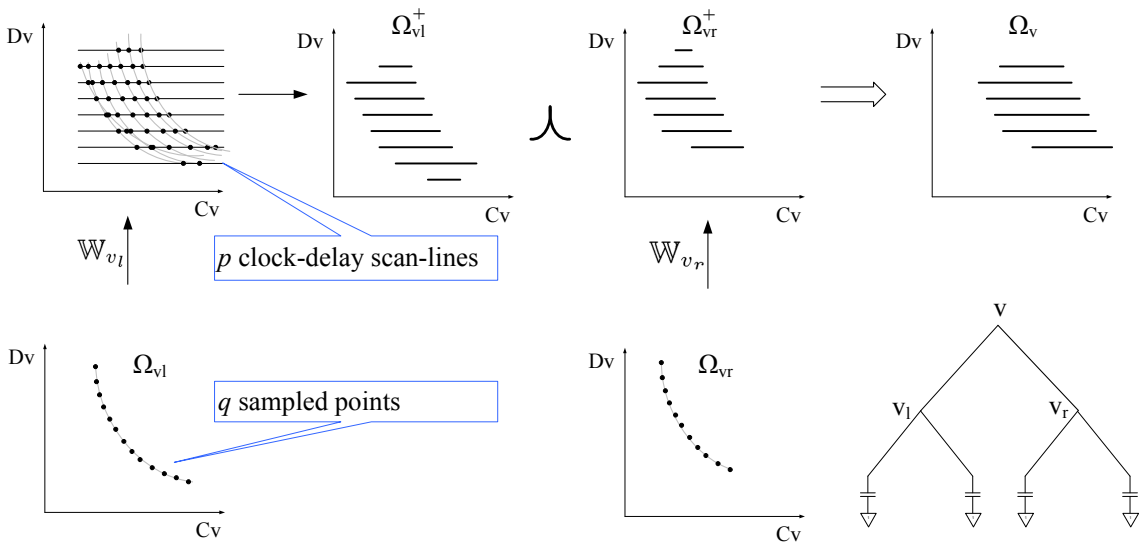


Figure 2.7: Obtain the sampled DC region of an unbufferd two-level clock tree.

2.3.3 Inverter Insertion

In practice, inverter insertion uses lesser area and is preferred over buffer insertion. However, inserting an inverter causes a 180 degree phase shift on the clock phase. The clock phase issue is easily accommodated by keeping two DC regions and branch DC regions at each node v, where

$$\Omega_v = \Omega_{vp} \cup \Omega_{vn},$$

$$\Omega^+_v = \Omega^+_{vp} \cup \Omega^+_{vn}.$$

From this point on in this dissertation the term buffer refers to inverter. Figure 2.8 shows the steps of projected scan-line sampling in obtaining the sampled branch DC region from the sampled DC region. The complete bottom-up DC region construction algorithm is presented in Figure 2.9.

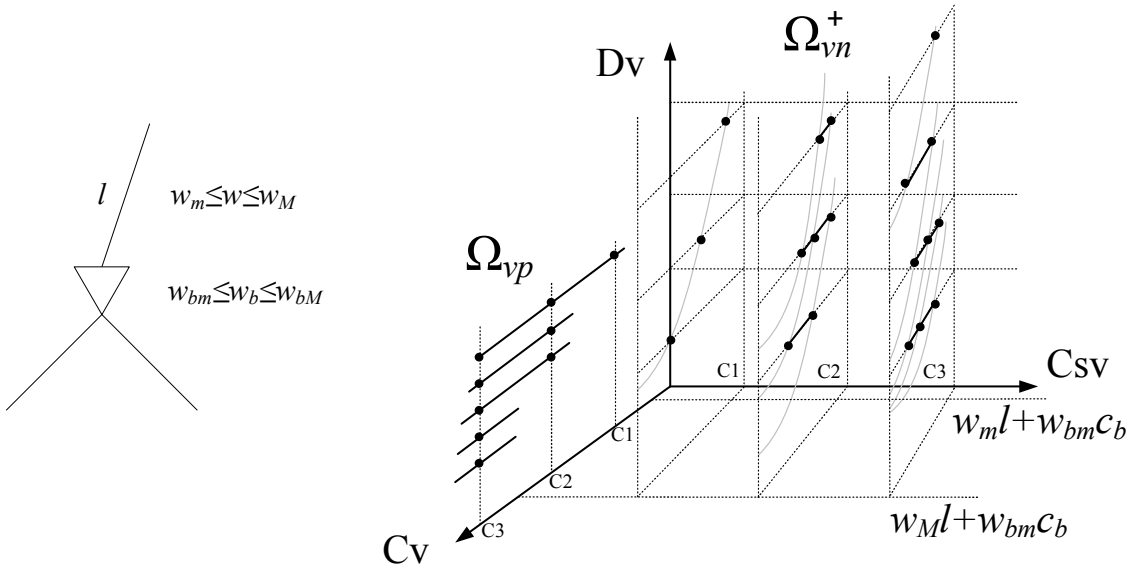


Figure 2.8: Obtain the sampled branch DC region of a buffered branch. Only the curves with $w_b = w_{bm}$ are shown.

Procedure: *ConstructDC(T_v)*

Input: a clock tree T_v with given routing rooted at node v

Output: DC regions and branch DC regions of T_v

if v *is a leaf node* **then**
  $\Omega_{vp} \leftarrow (d_v, c_v, 0), \Omega_{vn} \leftarrow \phi$
else
  call *ConstructDC(T_{v_l})*
  call *ConstructDC(T_{v_r})*
  $\Omega^+_{v_l p} \leftarrow \mathbb{W}_{v_l}(\Omega_{v_l p}) \cup \mathbb{B}_{v_l}(\Omega_{v_l n})$
  $\Omega^+_{v_l n} \leftarrow \mathbb{W}_{v_l}(\Omega_{v_l n}) \cup \mathbb{B}_{v_l}(\Omega_{v_l p})$
  $\Omega^+_{v_r p} \leftarrow \mathbb{W}_{v_r}(\Omega_{v_r p}) \cup \mathbb{B}_{v_r}(\Omega_{v_r n})$
  $\Omega^+_{v_r n} \leftarrow \mathbb{W}_{v_r}(\Omega_{v_r n}) \cup \mathbb{B}_{v_r}(\Omega_{v_r p})$
  $\Omega_v \leftarrow (\Omega^+_{v_l p} \curlywedge \Omega^+_{v_r p}) \cup (\Omega^+_{v_l n} \curlywedge \Omega^+_{v_r n})$
end

Figure 2.9: The bottom-up DC region construction algorithm.

2.3.4 Complexity

Let the number of nodes in a clock tree be n, the numbers of clock delay scan-line for a sampled DC region be p, and the numbers of samples taken on buffer width and shielded capacitance be q, and r. A sampled DC region is represented by r groups of horizontal segments lying on X-Z planes, with each group containing p segments. To generate a branch DC region, $q \cdot r$ curves are generated from each segment, and each of the $p \cdot q \cdot r^2$ curves is used to intersect with p scan-lines. Thus the complexity of the algorithm is $O(np^2qr^2)$ and memory usage is $O(npr)$. Note that there can be more than one segment on a clock delay scan-line. However, the gaps between the segments tend to be filled-up quickly as the DC regions are propagated upward toward the root node. For example, multiple segments can overlap and become a single segment either when the sampled branch DC regions are created or the sampled branch DC regions are merged by the $\curlywedge$ operator. In practice, the number of segments on a scan-line is always less than four and this factor is excluded from the complexity analyses.

While the approaches basing on van Ginneken's algorithm suffer from exponential growth on the size of the solution set, the proposed algorithm only takes polynomial runtime and memory usage. This advantage comes from the projected scan-line sampling. Since each line segment captures an infinite number of solutions, the proposed algorithm encodes more information with the same amount of memory and results in a better runtime and memory usage.

2.3.5 Slew-Rate Control and Useful-Skew

One of the purposes of buffer insertion is to adjust the clock slew-rate. If the load capacitance of a buffer is too large, the output signal has a slow rise and fall time, which in turn increases the short-circuit power of its downstream buffers. One way to control the slew-rate is to limit the load capacitance to a

certain value. This constraint is accounted for in the proposed algorithm by limiting c_v in the bottom-up phase and discarding long slew-rate designs. In the bottom-up phase, the DC regions can become large because of the embeddings with excessive buffers, which have large clock delay and total capacitances. By setting upper limits on d_v and $(c_{sv} + c_v)$, those ill-buffered embeddings are excluded from the DC regions.

Recently, useful-skew [77] concepts have been widely proposed to speed up timing convergence and compensate for timing uncertainties in physical layouts. The proposed algorithm accommodates DC region generation for a useful-skew design space by assigning useful-skew values to d_v at each leaf node. Furthermore, in low-power designs not requiring high clock frequencies, DC regions can be generated for a bounded-skew design space to allow more aggressive power optimizations. In this case, the skew-bounds are assigned to d_v. Thus Ω_v of a leaf node becomes a vertical segment, of which the Z-coordinates of the end points are the maximum and minimum acceptable clock arrival times and the X-coordinate is the sink capacitance.

2.3.6 Embedding Selection

The top-down embedding selection algorithm is illustrated by the following five steps:

1. If node v is the root, select desired $(\hat{d}_v, \hat{c}_v, \hat{c}_{sv}) \in \Omega_v$.

2. With $(\hat{d}_v, \hat{c}_v, \hat{c}_{sv}) \in \Omega_v$, determine $(\hat{d}_v, \hat{c}^+_{v_l}, \hat{c}^+_{sv_l}) \in \Omega^+_{v_l}$ and $(\hat{d}_v, \hat{c}^+_{v_r}, \hat{c}^+_{sv_r}) \in \Omega^+_{v_r}$ by breaking $\hat{c}_v$ and $\hat{c}_{sv}$ into $\hat{c}^+_{v_l}$, $\hat{c}^+_{v_r}$, and $\hat{c}^+_{sv_l}$, $\hat{c}^+_{sv_r}$ such that $\hat{c}_v = \hat{c}^+_{v_l} + \hat{c}^+_{v_r}$, $\hat{c}_{sv} = \hat{c}^+_{sv_l} + \hat{c}^+_{sv_r}$.

3. With $(\hat{d}_v, \hat{c}^+_{v_l}, \hat{c}^+_{sv_l}) \in \Omega^+_{v_l}$, find $(\hat{d}_v, \hat{c}_{v_l}, \hat{c}_{sv_l}) \in \Omega_{v_l}$ such that either (2.1)-(2.2) or (2.3)-(2.5) are satisfied.

   - To satisfy (2.1)-(2.2), $b_{v_l} = \phi$, $\hat{c}_{sv_l} = \hat{c}^+_{sv_l}$, and $w_e(v_l)$ is determined by $\hat{c}_{v_l}$.

- To satisfy (2.3)-(2.5), $\hat{c}_{v_l} = \hat{c}^+_{sv_l} - \hat{c}_{sv_l}$, and $w_b(v_l)$, $w_e(v_l)$ are determined by $(\hat{d}_{v_l}, \hat{c}_{v_l}, \hat{c}_{sv_l})$.

4. Repeat Step 3 for $(\hat{d}_v, \hat{c}^+_{v_r}, \hat{c}^+_{sv_r}) \in \Omega^+_{v_r}$.

5. Repeat the complete process at v_l and v_r.

To highlight the purpose of each step, Step 1 determines the target clock delay, capacitance seen at the root node, and shielded capacitance, Step 2 splits the capacitance budgets to two branches T^+_{vl} and T^+_{vr}, and Step 3 determines buffer widths for b_v and wire widths for e_v.

2.4 Experimental Results

The proposed algorithm is implemented in C++ and run on a 1GB 1.2GHz Pentium IV PC. The benchmarks $r1$-$r5$ are taken from [12]. All simulations use $r_0 = 0.03\Omega$, $c_0 = 2 \times 10^{-16}F$, $w_m = 0.3\mu m$, $w_M = 3\mu m$. The parameters of the buffers are $c_b = 40fF$, $r_b = 100\Omega$, $t_c = 30ps$, $w_{bm} = 1$, $w_{bM} = 10$, and the channel length equals $0.3\mu m$. The initial routings are generated by the BB+DME [13] algorithm. The sampling resolution is set to $p = q = r = 64$.

2.4.1 Delay and Power Minimization

Table 2.1 shows the minimum delay and minimum power solutions captured by projected scan-line sampling. The delays shown are the Elmore delays multiplied by $ln2$. The 'Load' columns show the total clock tree capacitances, including wire, buffer, and clock sink capacitances. The delay gains are the initial delays divided by the optimized delays and the load gains are the load reductions divided by the initial loads. With simultaneous buffer insertion, buffer sizing and wire sizing, the minimum power embeddings consume $23\% \sim 34\%$ less power and the delay improves $2X \sim 24X$ over their initial routings. Minimum delay solutions have more than 2X speedup compared with minimum power

Input	Initial($w = 1\mu m$)		*ConstructDC* ($w_m = 0.3\mu m, w_M = 3\mu m, w_{bm} = 1, w_{bM} = 10, p = q = r = 64$)										
	Delay	Load	Minimum delay solution					Minimum power solution					CPU
	(ns)	(pF)	Delay	Gain	Load	Gain	Buffers	Delay	Gain	Load	Gain	Buffers	(min.)
$r1$(267)	1.097	45.2	0.145	7.57	36.6	19.0%	34	0.500	2.20	29.8	34.1%	25	1.4
$r2$(598)	3.210	93.6	0.218	14.72	78.9	15.7%	61	0.818	3.93	65.0	30.6%	67	3.2
$r3$(862)	4.590	126.7	0.236	19.43	107.9	14.8%	106	0.563	8.15	84.6	33.2%	64	3.6
$r4$(1903)	13.184	266.2	0.454	29.02	198.6	25.4%	116	1.199	10.99	189.5	28.8%	115	10.5
$r5$(3101)	24.883	413.0	0.545	45.65	331.5	19.8%	384	0.999	24.90	317.6	23.2%	280	16.1

Table 2.1: Clock delay and power consumption before and after optimization.

solutions. The power difference between minimum delay and minimum power solutions decreases with larger circuits and drops to 5% in $r5$. Compared with the results using only wire sizing, which achieves an average of 3.3X delay reduction with 21.6% more power (with $1\mu m \leq w \leq 4\mu m$) [66], simultaneous buffer insertion, buffer sizing and wire sizing is not only much more efficient in reducing clock delay but also more power efficient. Figure 2.10 shows the DC region of $r5$.

2.4.2 Robustness to Process Variations

Figure 2.11(a) shows the clock delay and power trade-off curves of $r1$-$r5$. The power differences between minimum delay and minimum power embeddings are relatively small compared with their clock delay differences. Figure 2.11(b) shows the worst-case process-induced clock skews of the minimum delay and minimum power embeddings of $r1$-$r5$ with four systematic process variation models. In these models, cross-chip feature-size variation increases linearly from $0\mu m$ to $0.05\mu m$ affecting wire-widths, buffer-widths, and buffer channel-lengths. The four process variation models are: top-to-bottom, bottom-to-top, left-to-right, and right-to-left. Not surprisingly, process-induced clock skew is highly correlated to clock delay and Figure 2.11(a) is a good approximation for process-induced clock skew and power trade-offs. For low-power applications, clock delay and robustness to process variation can be traded for clock power by choosing minimum power embeddings. For high-performance designs, minimum delay

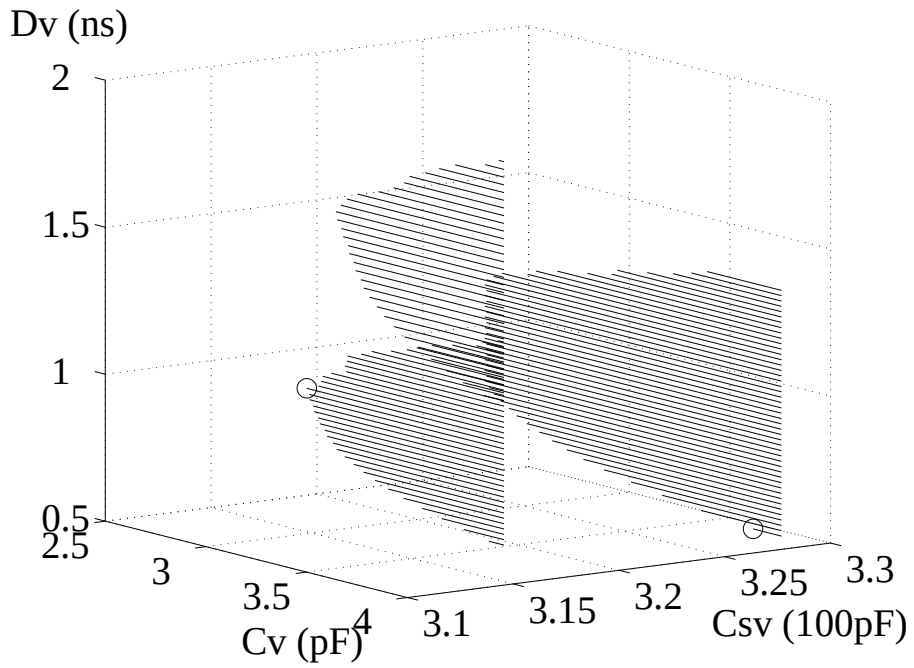


Figure 2.10: The sampled DC region of $r5$. The circles indicate the minimum delay and minimum power solutions.

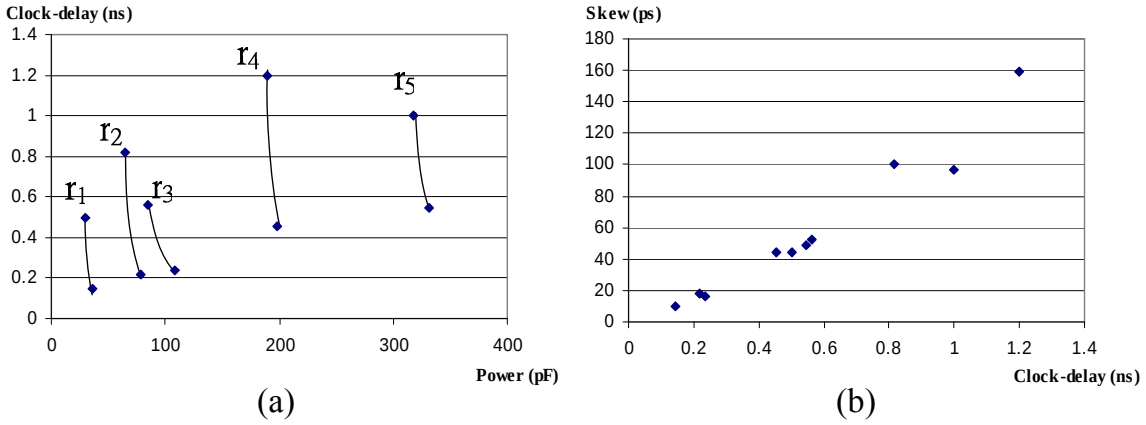


Figure 2.11: Trade-off among clock delay, clock power and clock skew. (a) Clock delay and power trade-off curves. (b) Worst-case process-induced clock skews of the minimum delay and minimum power embeddings of $r1$-$r5$.

Input	Discretization-induced clock skew(ps)			Process-induced clock skew(ps)
	$\Delta W = 0.05\mu m$	$\Delta W = 0.10\mu m$	$\Delta W = 0.15\mu m$	$\Delta W_{max} = 0.05\mu m$
r1(267)	**1.81**	4.63	3.99	**8.87**
r2(598)	**2.14**	7.40	6.01	**13.23**
r3(862)	**2.76**	6.39	9.88	**19.42**
r4(1903)	**9.36**	18.88	29.41	**37.88**
r5(3101)	**9.66**	21.94	35.03	**40.84**

Table 2.2: Comparison between discretization-induced clock skew and process-induced clock skew.

embeddings are in general preferable for their higher process variation tolerance and near-optimal power consumption.

2.4.3 Discrete Sizing

In industrial applications wire and buffer widths usually take discrete values. The buffer and wire widths generated by the proposed algorithm can be discretized to comply with layout restrictions. After discretization the embeddings are no longer zero-skew. Nevertheless, discretization introduces random variations to the clock tree and their effects tend to cancel each other out. Process variation is usually systematic and affects buffer channel-widths as well. Thus discretization-induced clock skew is much less significant than process-induced clock skew and near-zero-skew can still be achieved using the proposed algorithm with an additional discretization step.

Table 2.2 shows the discretization-induced clock skews and process-induced clock skews of minimum delay embeddings from Table 2.1. Upon discretization, all wire and buffer widths are rounded to the nearest multiples of unit widths ΔW. Results show that discretization-induced clock skew is within tolerable range and the proposed algorithm is suitable for industrial applications.

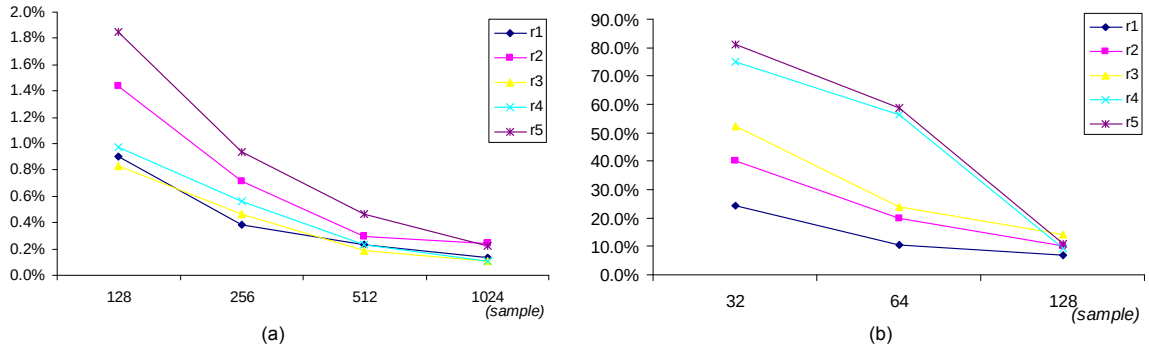


Figure 2.12: The relative distances from minimum delays to optimal delays. (a) In wire sizing problems. (b) In simultaneous buffer insertion, buffer sizing and wire sizing problems. Optimal delays are approximated by nonlinear curve fitting.

2.4.4 Optimality and Runtime

Theoretically, the proposed algorithm requires infinite samples in order for the minimum delay solutions to converge to optimal delay. Since the runtime complexity is polynomial, the convergence rate affects the scalability of the algorithm. Figure 2.12 shows the relative distances from minimum delays to optimal delays in which optimal delays are approximated by nonlinear curve fitting. The results show that it takes reasonable samples in finding good solutions. If the number of samples is fixed, the runtime is linear with respect to the size of the clock tree. Thus the algorithm scales well for large clock trees.

Chapter 3

Statistical-Timing-Driven Clock Scheduling

Timing yield can be improved by better timing margin utilization. The timing margin of a combinational path is defined by its path delay and the clock arrival times of its source and destination sequential elements. Therefore, clock scheduling can be used to improve the timing yield. In the first section, the clock scheduling problem and its use for improving the timing yield is reviewed. In the second section, a novel false-path-aware statistical timing analysis method is proposed to provide the information for clock scheduling. In the third section, statistical timing information is combined with a parametric shortest path algorithm to obtain the clock schedule that maximizes the timing yield efficiently. In the last section, a false-path-aware gate sizing method, which preserves more timing margin than traditional gate sizing algorithms with little or negligible area penalty, is proposed. Significant timing yield improvements are achieved ($> 20\%$) on many of the benchmark circuits due to the extra timing margins and better timing margin utilization.

3.1 Preliminaries

A sequential circuit can be represented by a directed *circuit graph* $\mathcal{G} = (V, E)$. A vertex $v \in V$ represents a latch or a flip-flop and an edge $e_{ij} = (i, j) \in E$ indicates that there is at least one combinational path from vertex i to vertex j. A *path* p is a sequence of vertices $< v_0, v_1, \ldots, v_k >$ in $\mathcal{G}$ such that $(v_i, v_{i+1}) \in E$ for $i = 0 \ldots k-1$ and $|p|$ is the number of edges in the path. A *cycle* c is a path with $v_0 = v_k$ that contains at least one edge. A circuit graph may not be connected if the circuit contains independent sub-circuits. Without loss of generality $\mathcal{G}$ is assumed connected, $|V| = n$, and $|E| = m$ in this chapter.

3.1.1 Clock Period Optimization

Figure 3.1(a) shows an example circuit with three flip-flops. The maximum and minimum path delays between FF_i and FF_j are D_{ij} and d_{ij}, respectively. Let CP be the clock period, and T^j_{setup} and T^j_{hold} be the setup and hold time of FF_j. The clock arrival time to FF_i is T_i and the clock skew between FF_i and FF_j is $\alpha_{ij} = T_i - T_j$. By absorbing the FF delays (C to Q delay, T_{CQ}) as part of the path delays, the hold time and setup time constraints can be written as:

$$\alpha_{ij} \geq T^j_{hold} - d_{ij}. \tag{3.1}$$

$$\alpha_{ij} \leq CP - D_{ij} - T^j_{setup}. \tag{3.2}$$

There are also two sets of constraints stemming from the clock skew definition $\alpha_{ij} = T_i - T_j$. For each flip-flop pair, there could be more than one path in between. The *path constraints* require the sum of the clock skews for these paths to be a constant. The *cycle constraints* require the sum of the clock skews of all cycles to be zero. For a cycle $< 1, 2, 3, 1 >$, the cycle constraint equals $\alpha_{12} + \alpha_{23} + \alpha_{31} = 0$. Although a circuit graph can have an exponentially growing number of path and cycle constraints as the

circuit size increases, many of the constraints are linearly dependent. These constraints can be rewritten as

$$[\boldsymbol{B_\alpha}\ \boldsymbol{I}] \begin{bmatrix} \boldsymbol{\alpha_b} \\ \\ \boldsymbol{\alpha_c} \end{bmatrix} = 0, \tag{3.3}$$

where $\boldsymbol{B_\alpha}$ is a $(m-n+1)\times(n-1)$ matrix, $\boldsymbol{I}$ is a $(m-n+1)\times(m-n+1)$ identity matrix, $\boldsymbol{\alpha_b}$ is a $(n-1)\times 1$ column vector corresponding to a maximum independent set of skew variables, and $\boldsymbol{\alpha_c}$ is a $(m-n+1)\times 1$ column vector corresponding to the rest of the skew variables. The $(n-1)$ elements of $\boldsymbol{\alpha_b}$ can be found by applying any spanning tree algorithm from any vertex v to the (undirected version of the) circuit graph. After $\boldsymbol{\alpha_b}$ is determined, each row of $\boldsymbol{B_\alpha}$ associated with $\alpha_{xy} \in \boldsymbol{\alpha_c}$ can be obtained by taking the difference of the path costs from v to x and from v to y.

The *clock period optimization problem* can be formulated as follows:

$$\begin{aligned} Minimize \quad & CP \\ s.t. \quad & T^j_{hold} - d_{ij} \le \alpha_{ij} \le CP - D_{ij} - T^j_{setup} \\ & [\boldsymbol{B_\alpha}\ \boldsymbol{I}] \begin{bmatrix} \boldsymbol{\alpha_b} \\ \\ \boldsymbol{\alpha_c} \end{bmatrix} = 0. \end{aligned}$$

The constraints can be extracted from the circuit structure and the optimal clock period can be obtained by linear programming solvers as done in [33]. Since the circuit graph is usually available, the problem can also be solved using a graph-theoretical approach without the the constraint extraction step. In [34], a *timing graph* $\mathcal{G}(CP)$ is created by replacing the hold time constraint of α_{ij} with an *h-edge* with cost $-(T^j_{hold} - d_{ij})$ from FF_i to FF_j, and by replacing the setup time constraint of α_{ij} with an *s-edge* with cost $(CP - D_{ij} - T^j_{setup})$ from FF_j to FF_i. Figure 3.1(b) shows the timing graph of the circuit in Figure 3.1(a) with zero T_{hold} and T_{setup}. Clock period CP is feasible if $\mathcal{G}(CP)$ contains no

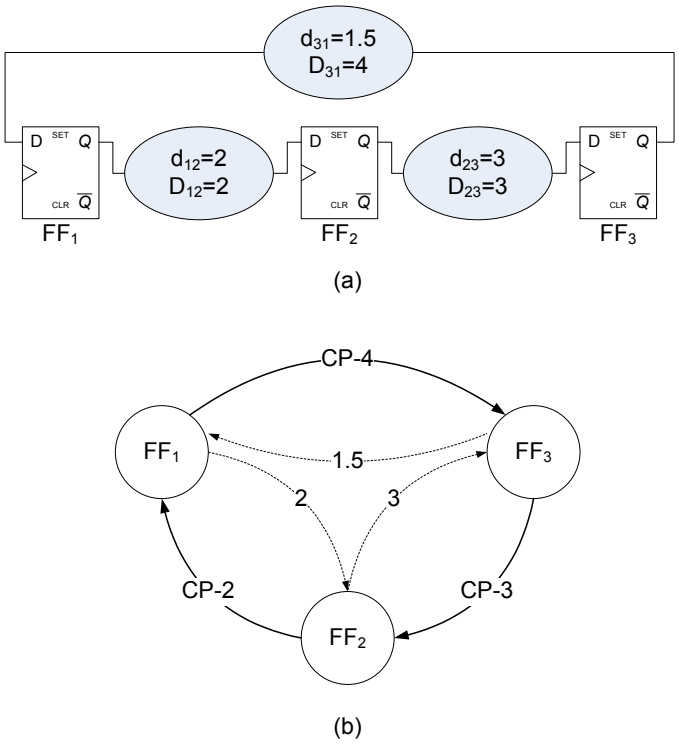


Figure 3.1: Illustration of timing graph. (a) An example circuit. (b) The timing graph.

negative cost cycle. Therefore, the optimal clock period can be obtained through a binary search between $[0, max(D_{ij}) + T^j_{setup}]$ by applying the Bellman-Ford algorithm on $\mathcal{G}(CP)$. An alternative is to solve the parametric shortest path problem by a path-pivoting algorithm [43]. In Figure 3.1(b), the optimal clock period is 3 and the three *s-edges* form a zero-cost cycle.

3.1.2 Clock Skew Optimization

The optimal clock period and the clock schedule found in the previous section will result in low timing yield designs since many skew values are on their upper or lower bounds and have zero slack. Let the *feasible skew region* (FSR) of α_{ij} be $[T^j_{hold} - d_{ij}, CP - D_{ij} - T^j_{setup}]$. It is evident that the choice of clock period only affects the right boundary of the region. Figure 3.2(a) shows the FSRs of the clock

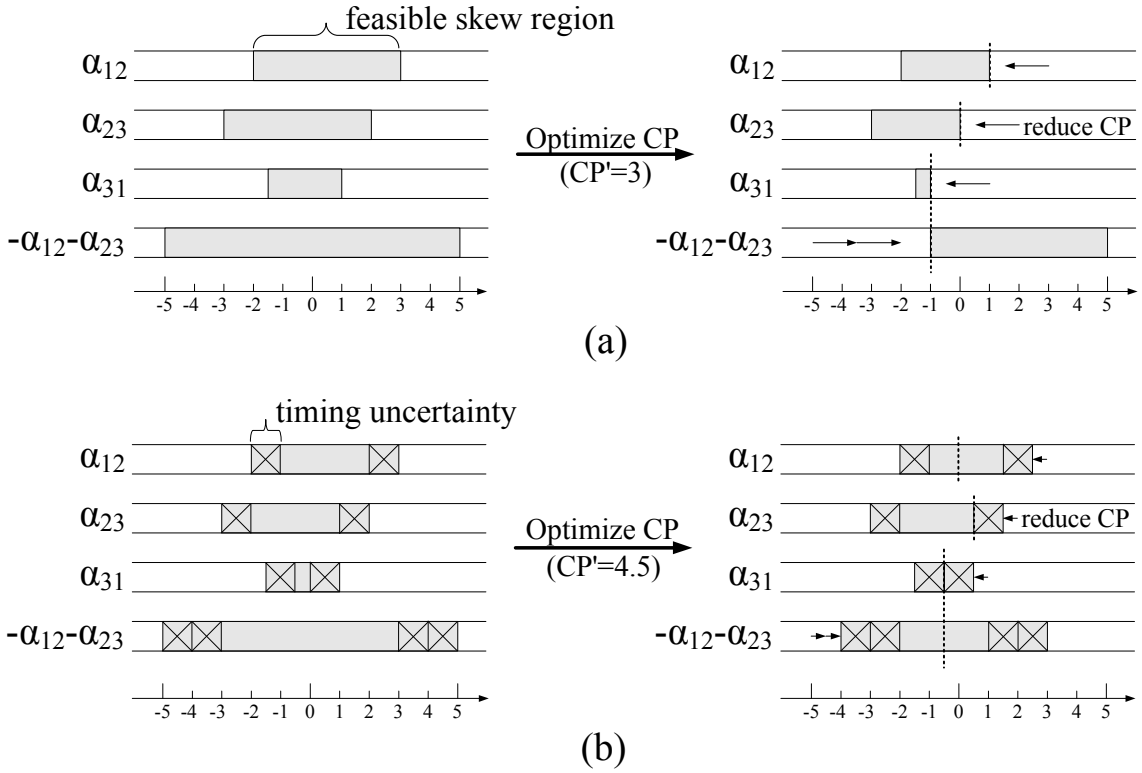


Figure 3.2: The impact of timing uncertainty on clock period optimization.

skew variables in Figure 3.1. Let the initial clock period $CP = 5$. The FSRs of α_{12} and α_{23} are $[-2, 3]$ and $[-3, 2]$ while the FSR of α_{31} is $[-1.5, 1]$. When the clock period is reduced, the right boundaries of the FSRs of α_{12}, α_{23}, and α_{31} move to the left while the left boundary of $-\alpha_{12} - \alpha_{23}$ moves to the right. Due to the cycle-constraint $\alpha_{31} = -\alpha_{12} - \alpha_{23}$, the FSR of α_{31} has to overlap with the range of $-\alpha_{12} - \alpha_{23}$ to yield a solution. Therefore, the optimal clock period CP' is 3 (2 time units less than CP) and the clock skew schedule $(\alpha_{12}, \alpha_{23}, \alpha_{31}) = (1, 0, -1)$ indicated by the dashed lines has zero slack.

Assuming the maximum timing variations of the minimum and maximum path delays are both one time unit. A simple way to guarantee the clock schedule will have slacks at least as large as the maximum timing variations is to pre-allocate a timing margin of one time unit at both ends of the FSRs and then perform clock period optimization. As shown in Figure 3.2(b), one of the feasible clock skew schedules

$(\alpha_{12}, \alpha_{23}, \alpha_{31}) = (0, 0.5, -0.5)$ is indicated by the dashed lines and the optimal clock period CP' is 4.5 (0.5 time unit less than CP). The slacks on (e_{12}, e_{23}, e_{31}) are $(2, 1, 1)$. This approach increases the optimal clock period by an amount no less than the pre-allocated timing margin (1.5X in the above example). It is not desirable to allocate a timing margin equals to the maximum timing variation since (i) the actual timing uncertainty cannot be accurately obtained, (ii) the maximum timing variation is too pessimistic, and (iii) the ratio of clock period increase versus the amount of pre-allocated timing margin can be large. In practice the clock period is usually determined by product requirement and a good clock schedule that maximize the timing yield needs to be determined.

A simple observation suggests that the skew values should be chosen as close to the middle points of their FSRs to maximize the slacks. Neves et al. [35] formulate the clock skew optimization problem as a least square error problem where the error is defined as the difference between the skew and the middle point of its FSR. However, the formulation may reduce the slacks of some skew values to zero to minimize the total error. Therefore, the resulting clock schedule may not be optimal in terms of timing yield. Albrecht et al. [39] adopt the minimum balance algorithm [43] to maximize the slacks of all skew values iteratively. In each iteration a *slack optimization problem* that maximizes the minimum slack is solved by a parametric shortest path algorithm. The critical cycle that has the minimum average slack is contracted into a single vertex and the process repeats until all skew values are assigned. This algorithm ensures the slacks along the timing-critical cycle are the same. However, it does not consider the path delay differences between the edges of the cycle. This *slack-balancing* clock scheduling algorithm is the most plausible method in the existing literature.

3.2 False-Path-Aware Statistical Timing Analysis

In nanometer designs, statistical timing analysis on path delays is a critical tool used to measuring the growing impacts of process variations. There are two major approaches to obtain path delay distributions. On one hand, a block-based algorithm performs timing analysis by propagating the delay distributions of intermediate nodes to the output in a breadth-first manner. Although the runtime of block-based algorithms is linear to the circuit size, special treatments are required to account for correlations and re-convergence paths. On the other hand, a path-based algorithm can handle correlations and reconvergence of paths. However, the number of paths in a circuit can grow exponentially with respect to the circuit size. Therefore, efficient algorithms are required to select a set of important paths for analysis.

Most of the statistical timing algorithms, either block-based or path-based, do not consider false paths. In other words, the delay distributions are obtained assuming all structural paths are sensitizable. Liou et al. [48] combine a logically true path selection algorithm with a statistical timer for false-path-aware statistical timing analysis. Runtime is reduced by selecting only the true paths for analysis and the results are more accurate. The logically true path selection algorithm works from PO to PI level by level. New partial paths are created by appending the fan-in gates to the original paths. However, a new partial path will be created only when it is sensitizable and its expected slack is below a predefined threshold. The expected slack is estimated by the worst case statistical timing analysis and is subject to error. Moreover, the number of true paths the algorithm selects is affected by the threshold value.

For clock scheduling, only the long and the short true paths between pairs of flip-flops are of interest. An implicit true path enumeration algorithm is developed to find the k-longest (k-shortest) true paths efficiently. To find the k-longest paths between PI and PO, the expected delay (ED) from each internal node to PO is first calculated. The algorithm traverses from PI to PO by selecting the path with the

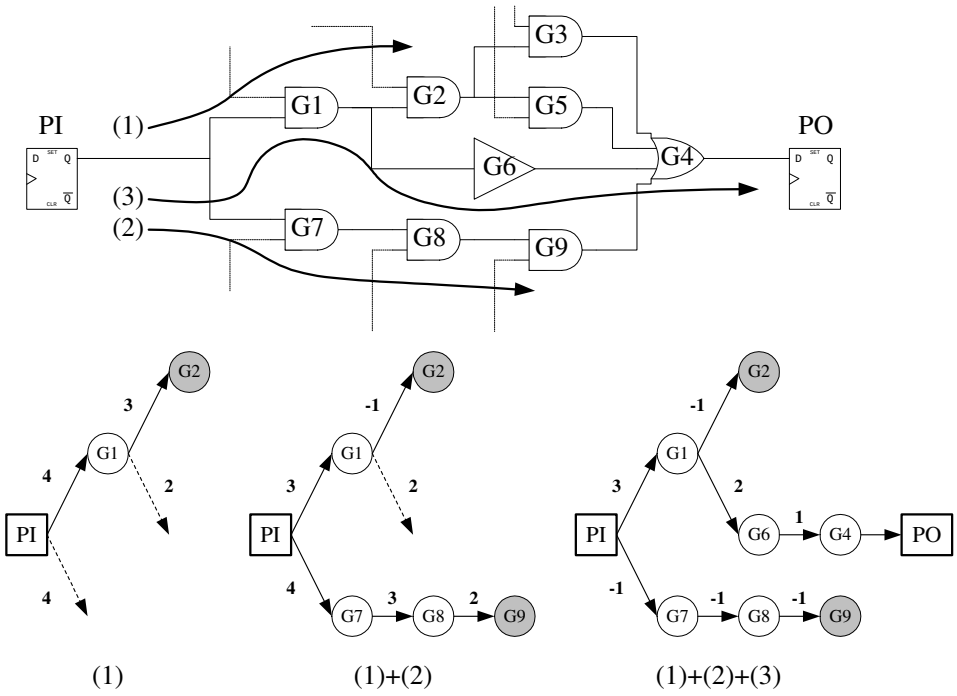


Figure 3.3: Implicit enumeration of true paths using dynamic delay path tree.

highest ED and the gates on the selected path are added to a dynamic delay path tree (DPT). For newly added gates, necessary assignments and maximum implications are performed using path sensitization criteria [46] to check their sensitizability. The search continues if sensitization is successful. Otherwise, the EDs are updated as the algorithm backtracks from the current node of the DPT to PI. The algorithm resumes when PO is reached or it backtracks to PI until the k-longest true paths are found. The paths are then sent to path-based statistical timers, such as [46, 78], for analysis. By taking the negative expected delay as ED, the algorithm finds the k-shortest paths.

Figure 3.3 demonstrates the implicit true path enumeration algorithm using a dynamic delay path tree. The numbers on the edges of the DPT are the EDs. The algorithm first searches along $< G1, G2 >$ and backtracks because $G2$ is not sensitizable. The algorithm continues on $< G7, G8, G9 >$ because it has the highest ED from PI. After the second backtracking the algorithm finds the longest true path

$< G1, G6, G4 >$. The results of maximum implication are stored in the DPT nodes so that they can be reused when the algorithm backtracks to a previously visited path. Although there are two structurally longest paths, $< G1, G2, G3, G4 >$ and $< G1, G2, G5, G4 >$, they are dropped and never explicitly enumerated as soon as $< G1, G2 >$ is found un-sensitizable.

3.3 Statistical-Timing-Driven Clock Scheduling

3.3.1 Clock Scheduling by the Parametric Shortest Path Algorithm

A clock schedule determined at design time is not always feasible for every manufactured chip since d_{ij} and D_{ij} are random variables. It is important to utilize the statistical timing information of d_{ij} and D_{ij} to find the optimal clock schedule that maximizes the timing yield. Define the *slack* of e_{ij} in a timing graph as $s_{ij} = \alpha_{ij} + w(e_{ij}) = T_i + w(e_{ij}) - T_j$. The total slack on a cycle $c =< v_1, v_2, \ldots, v_k, v_1 >$ is

$$s(c) = \sum_{e_{ij} \in c} s_{ij} = \sum_{e_{ij} \in c} w(e_{ij}). \tag{3.4}$$

This $s(c)$ is a limited resource and the slack of an edge cannot be arbitrarily increased. Therefore, clock scheduling can be viewed as the process of distributing cycle slacks to edges.

To optimize the timing yield, cycle slacks should be distributed to an edge according to the standard deviation of its associated path delay. Define the *normalized slack* of h-edge e_{ij} as $\lambda_h(e_{ij}) = \frac{T_i + \mu(d_{ij}) - T_j - T^j_{hold}}{\sigma(d_{ij})}$ and that of s-edge e_{ji} as $\lambda_s(e_{ji}) = \frac{CP + T_j - T^j_{setup} - T_i - \mu(D_{ij})}{\sigma(D_{ij})}$. The hold time and setup time constraints can be rewritten as:

$$T_i + \mu(d_{ij}) \geq T_j + T^j_{hold} + \sigma(d_{ij})\lambda_h(e_{ij}). \tag{3.5}$$

$$T_i + \mu(D_{ij}) + \sigma(D_{ij})\lambda_s(e_{ij}) \leq CP + T_j - T^j_{setup}. \tag{3.6}$$

Note in the above the hold time and setup time slacks, λ_h and λ_s, are normalized with the path delay

uncertainties, and these should be maximized to maximize the timing yield. After optimization, paths with larger path delay uncertainties will obtain larger slacks ($\sigma \cdot \lambda$).

The optimal clock schedule can be obtained by a graph-theoretical algorithm. First, the edge weight of an h-edge e_{ij} in $\mathcal{G}(CP)$ is replaced by a parametric edge weight of $\mu(d_{ij}) - T^j_{hold} - \sigma(d_{ij})\lambda$ and the edge weight of an s-edge e_{ji} is replace by a parametric edge weight of $CP - \mu(D_{ij}) - T^j_{setup} - \sigma(D_{ij})\lambda$. This step yields a *parametric timing graph* $\mathcal{G}^{CP}(\lambda)$[1]. A clock schedule that does not cause any negative cycle in $\mathcal{G}^{CP}(\hat{\lambda})$ guarantees all edges have normalized slacks of at least $\hat{\lambda}$. The *optimal clock scheduling* problem is to find a clock schedule that maximizes λ given a parametric timing graph $\mathcal{G}^{CP}(\lambda)$.

Held et al. [40] and Albrecht et al. [39] use a *parametric shortest path* algorithm [43] to solve a special case of the problem where $\sigma(\cdot) \triangleq 1$. The parametric shortest path algorithm is adopted and $\sigma(\cdot)$ is obtained by statistical timing analysis. To find the optimal clock schedule, the algorithm starts from a feasible clock schedule $\mathbf{T_0}$ obtained by applying the Bellman-Ford shortest path algorithm on $\mathcal{G}^{CP}(\lambda_0)$, where $\lambda_0 = 0$. Since CP is always chosen to be greater than the optimal clock period CP^*, which is obtained by the methods described in 3.1.1, $\mathbf{T_0}$ always exists. *Path pivoting* [43] is performed and new pairs of $(\lambda_k, \mathbf{T_k})$ are found where $\lambda_k \geq \lambda_{k-1}$. When a zero-cost cycle is detected, the *cycle contraction* [43] step is performed. The zero-cost cycle corresponds to a cycle in the circuit where the normalized slack on the cycle cannot be increased further. The current clock schedule on the cycle is thus optimal and the cycle can be contracted and replaced by a *super vertex*. The timing constraints between the rest of the circuit and the vertices on the cycle are updated and imposed to the super vertex. In other words, the edges from/to the cycle are replaced by new edges from/to the super vertex. When $\sigma(\cdot) \triangleq 1$ multiple edges from v_i to v_j can always be collapsed into one edge by discarding non-dominant ones. In

[1]Here CP is a fixed value. Therefore, it is moved to the superscript.

the new formulation, multiple edges might need to be kept if their parameter coefficients are different. The path pivoting and cycle contraction steps are repeated until the parametric timing graph becomes a single super vertex.

3.3.2 Complexity Analysis

The parametric shortest path algorithm is a graph-theoretical version of the Simplex algorithm [79], in which a basic feasible solution ($\mathbf{T_i}$) is moved to another basic feasible solution ($\mathbf{T_{i+1}}$) with increasing objective (λ) through row pivoting on the Simplex tableau. Young et al. [43] prove that for a graph with $|V| = n$ and $|E| = m$, the parametric shortest path algorithm takes $O(nm + n^2 logn)$ time on the special case where $\sigma(\cdot) \triangleq 1$. Although the worst case running time of the Simplex algorithm is exponential in some specially synthesized problems, it usually takes much less time for general problems [80]. Experiments show that the number of iterations required for obtaining the optimal clock schedule is comparable to the number of iterations used in solving the special case. Furthermore, the timing yield of the optimal clock schedule is significantly increased in many benchmark circuits.

3.3.3 A Statistical Timing Enhanced Yield Model

To estimate the timing yield of a clock schedule, circuit samples need to be generated and setup time and hold time constraints need to be verified. Although the error of a Monte Carlo simulation is proportional to $\frac{1}{\sqrt{N}}$, where N is the number of samples, and independent of the dimension of the sample space, the time to verify a circuit increases significantly as the circuit size grows. Unlike the Gaussian distribution which extends to $\pm\infty$, gate delay usually varies no more than a few standard deviations from its mean value. Using *extend factor* κ such that $\kappa\sigma$ is the maximum timing variation of a gate, the Gaussian distribution outside $\mu \pm \kappa\sigma$ is truncated and the truncated distribution is re-normalize to describe the

delay distribution of a gate. The new distribution is the *truncated Gaussian* distribution. Agarwal et al. [81] propose a statistical timing analysis algorithm which generates the upper bound and the lower bound of the maximum and minimum path delay distributions. It is proven that for a two-input gate with the probability density functions of the inputs being f and g and the cumulative density functions being F and G, $fG + gF$ gives a pessimistic estimation of the distribution of the latest input arrival time. Likewise, $f(1 - G) + g(1 - F)$ gives an optimistic estimation of the distribution of the earliest input arrival time. Using this algorithm, the bounds of D_{ij} and d_{ij}, $\overline{D_{ij}}$ and $\underline{d_{ij}}$, can be obtained from the distributions. After clock scheduling, the check to verify if α_{ij} is between $[T^j_{hold} - \underline{d_{ij}}, CP - \overline{D_{ij}} - T^j_{setup}]$ for each edge of the timing graph is performed. If the check is satisfied, e_{ij} is timingly safe under process variations and D_{ij} and d_{ij} do not need to be calculated for each sample. In general, most of the timing edges satisfy this condition. Therefore, Monte Carlo simulations only need to be performed for the combinational paths between a few pair of flip-flops and the simulation time is significantly reduced by a factor of $10X \sim 1000X$ depending on the circuit size.

3.3.4 Experimental Results

The experiments are conducted on a Pentium III 1GHz PC. Truncated Gaussian distributions with $(\mu, \sigma) = (1, 0.15)$ and $\kappa = 3$ are used for gate delays and each gate is assumed independent. Path delay uncertainties are approximated by $\sqrt{n}\ \sigma$, where n is the number of gates on a path. To give reasonable comparisons, the clock periods are chosen so that the timing yields of traditional clock schedules are between 60% to 80%. The clock arrival times of all primary inputs and outputs are assumed the same and the corresponding vertices are contracted to a single super vertex before clock scheduling. Monte Carlo simulations are performed and false path information is used to calculate the timing yields for both traditional and optimal clock schedules.

Circuits	Total FF pairs	d-diff pairs	D-diff pairs	Circuits	Total FF pairs	d-diff pairs	D-diff pairs
b05s	700	6	93	s1488	266	0	0
b06	44	0	0	s5378	2313	1	0
b07s	1155	0	24	s9234.1	3260	0	0
b10	165	0	0	s13207.1	4721	0	0
b11s	631	23	65	s35932	7595	0	1600
b12	1630	0	3	s38417	34351	72	852
b13s	393	0	0	s38584.1	20444	12	17

Table 3.1: Structural and true longest/shortest paths.

			Traditional clock scheduling				Statistical-timing-driven clock scheduling				
Circuits	CP^*	CP	pivot	contract	yield	CPU	pivot	contract	yield	Gain	CPU
s1488	16.0	16.62	0	6	72.3%	0.4s	4	6	75.1%	3.9%	0.4s
s5378	21.0	22.50	107	179	63.8%	0.4s	169	179	64.2%	0.6%	0.5s
s9234.1	38.0	40.86	160	210	74.1%	0.7s	167	210	84.2%	**13.6%**	0.8s
s13207.1	51.0	52.73	276	620	60.2%	1.9s	397	620	60.2%	0.0%	2.0s
s35932	28.0	31.96	1790	1728	64.3%	5.3s	1108	1728	98.7%	**53.5%**	11.9s
s38417	31.5	34.07	1893	1636	74.0%	66.3s	1641	1636	88.6%	**19.7%**	79.9s
s38584.1	48.0	50.24	1722	1426	72.8%	7.6s	1699	1426	84.7%	**16.3%**	9.9s

Table 3.2: Timing yield and runtime comparison between traditional clock scheduling and statistical-timing-driven clock scheduling.

Table 3.1 shows the structural and true path information on *ISCAS89* benchmark circuits. The names of the circuits are shown in the first column. The numbers of flip-flop pairs that have combinational paths in between are shown in the second column. The third column, d-diff, shows the number of flip-flop pairs that have the shortest true path delays larger than the shortest structural path delays. The forth column, D-diff, shows the number of flip-flop pairs that have the longest true path delays smaller than the longest structural path delays. It shows that the longest/shortest true path delays are usually the same as the longest/shortest structural path delays. However, in s35932, more than 20% of the flip-flop pairs have longest true path delays smaller than their longest structural path delays.

Table 3.2 shows the comparison of runtime and timing yield on *ISCAS89* benchmark circuits between a traditional clock scheduling algorithm with only structural longest/shortest path delay information and

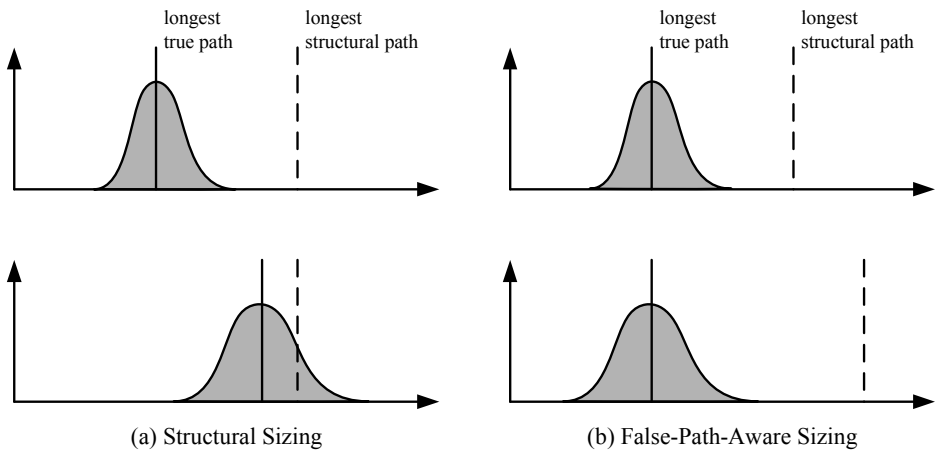


Figure 3.4: Path delay distributions. (a) By traditional gate sizing. (b) By false-path-aware gate sizing.

the proposed false-path-aware statistical-timing-driven clock scheduling algorithm. The results show that the proposed clock scheduling scheme takes similar runtime compared to the traditional clock scheduling scheme. Moreover, the proposed clock scheduling scheme achieves more than 10% timing yield improvements on four of the circuits, and the improvement is as high as 53.5% on s35932. Analysis on the results in Table 3.1 and 3.2 show that the proposed clock scheduling scheme achieves better timing yields by 1) better timing margin utilization and 2) identifying and making use of the timing margins that are masked by false paths.

3.4 False-Path-Aware Gate Sizing

The experimental results in the previous section suggest that the timing margins masked by false paths may be used to improve the timing yield, especially when these false paths are on timing-critical cycles. However, those timing margins can be wiped out in the gate sizing stage that does not consider false paths. Figure 3.4(a) shows the path delay distribution for a circuit before and after gate sizing in a conventional design flow. When the longest true path is shorter than the longest structural path, a traditional gate

sizing algorithm will size down the gates (or skip repeater insertions) on the true paths to reduce area and power. This can cause the true path delay distribution to shift to the right. For high-performance designs this will cause a significant performance degradation and will require serious manual ramifications to fix the timing problems. However, false-path-aware gate sizing will not increase the longest true path delay as illustrated in Figure 3.4(b). Thus, costly engineering-change-orders (ECOs) can be prevented.

3.4.1 The False-Path-Aware Gate Sizing Flow

To maximize the timing yield, the clock schedule should be determined based on the statistical timing information. In particular the slacks should be distributed to each path according to its delay uncertainty. Since gate sizing changes the path delay distributions, an optimization method for sequential circuits must perform statistical timing analysis and clock scheduling in each gate sizing iteration for timing yield estimation. However, false-path-aware statistical timing analysis and clock scheduling can be a bottle neck if used in the inner loop of the optimization.

The design flow proposed in Figure 3.5 breaks the circuit optimization stage into two separate steps. The differences between the proposed flow and a traditional flow are that the proposed flow uses a false-path-aware gate sizing algorithm, and the clock schedule is determined after gate sizing is completed thus avoiding costly iterations of the processes. Details of false-path-aware gate sizing is presented in the next section.

3.4.2 Problem Formulation

A gate sizing formulation based on the work by Chen et al. [27] is shown below. Let d_i be the delay of gate i and assume the gate size is inversely proportional to the gate delay, the false-path-aware gate

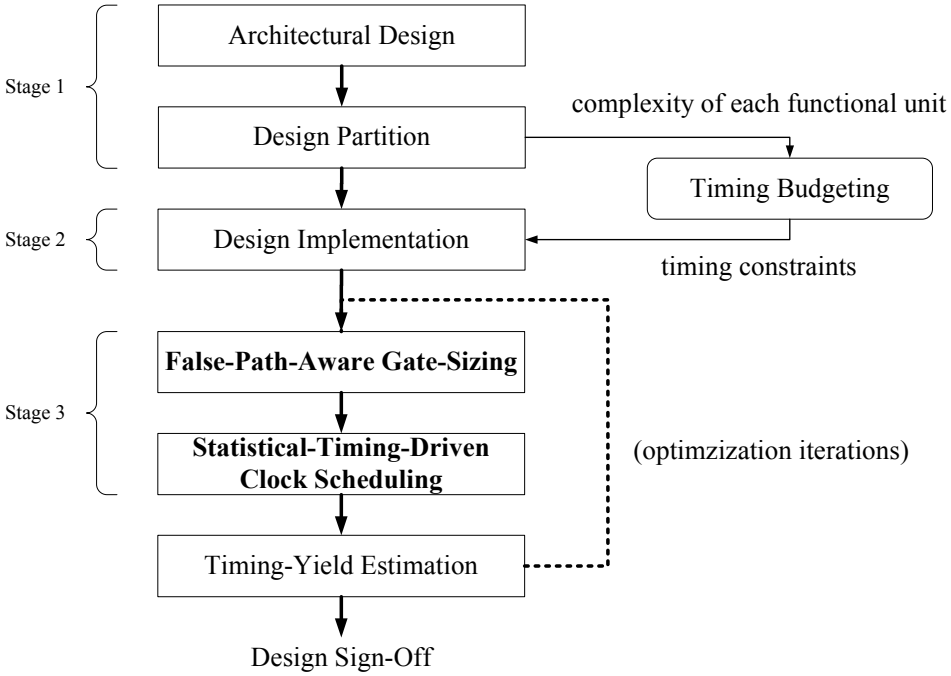


Figure 3.5: The false-path-aware gate sizing flow.

sizing problem can be formulated as follows.

$$(\mathcal{GS}) \quad min. \quad \sum_{i=1}^{n} \frac{1}{d_i} \tag{3.7}$$

$$s.t. \quad \sum_{i \in p} d_i \leq A_0 \quad \forall p \in P - F \tag{3.8}$$

$$L_i \leq d_i \leq U_i \quad i = 1 \ldots n \tag{3.9}$$

Where A_0 is the target delay, P is a set of all possible paths, and F is a set of all false paths. The constraint (3.9) limits the lower and upper bound of the gate delays. By solving the above optimization problem, the gate delay d_i will be increased until it reaches U_i, or the delay of some path passing through gate i reaches A_0.

Consider the circuit in Figure 3.6 with two false paths, $< 8, 7, 5, 3, 2, 1 >$ and $< 9, 7, 5, 3, 2, 1 >$. Let $L_i = 1, i = 1 \ldots 7$, and $A_0 = 5$, a conventional sizing method will assign a delay of 1 to all the

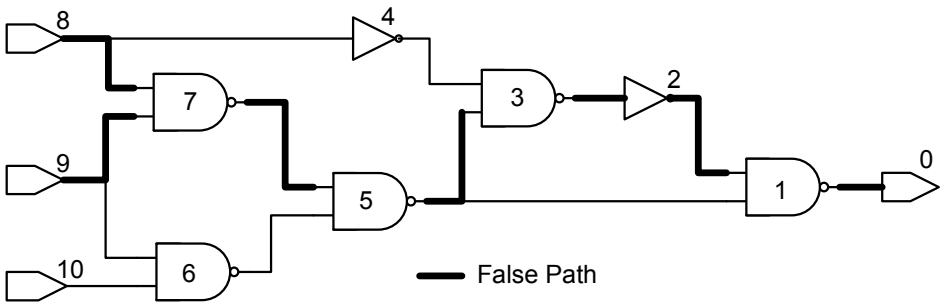


Figure 3.6: Gate sizing with false paths.

gates on $< 7, 5, 3, 2, 1 >$. However, if the two false paths are identified, the corresponding path delay constraints can be removed from (3.8). Note that after removing the constraints on false paths, gate 7 can have a longer delay (smaller gate size) since the path $< 7, 5, 1 >$ is the only sensitizable path through gate 7.

False paths can be identified by checking the sensitization criteria [47]. To verify the sensitizability of a given path, transitions on the target path are assigned and maximal implication is performed on the circuit. If the on-path signal has a transition from non-controlling value to controlling value, all the off path signals (side inputs) for that node must have non-controlling values. If maximal implication of such assignment has conflicts, the target path is considered as a false path since the on-path signal cannot be propagated.

3.4.3 Gate Sizing for Sequential Circuits

In a sequential circuit with clock scheduling, the maximum feasible path delay for the paths between a pair of flip-flops depends on the clock schedule. Moreover, the total number of path delay constraints for a large sequential circuit can be prohibitive. To address these two issues, two heuristic-based solutions that lead to optimal or near optimal solutions are proposed. The implementation problems and the

solutions are also discussed.

Pair-wise sizing

A divide-and-conquer heuristic is proposed to apply $\mathcal{GS}$ to sequential circuits. First, $\mathcal{GS}$ is applied to each pair of flip-flops separately. For each pair of flip-flops, the fan-out cone of PI and the fan-in cone of PO are extracted, and only the gates that are in the cone intersection are targeted for sizing. This approach is desired especially when the timing budget between each pair of flip-flops are different due to clock scheduling. Since a gate can belong to the cone intersections of multiple flip-flop pairs, more than one gate delay could be assigned to the same gate. In this case, the minimum delay will be assigned to avoid creating paths with setup time violations. Although this heuristic is sub-optimal, experimental results show that it still generates relatively good results.

Sizing for K-longest true paths

Experimental results show that most of the benchmark circuits have reasonable number of true paths. For these circuits, $\mathcal{GS}$ can be directly applied for gate sizing. However, some pairs of FFs do have exponential number of true paths in their cone intersections. In those cases, only the k-longest true paths are extracted using the implicit true path enumeration algorithm presented in 3.2. Since this does not guarantee that the setup time constraints for all the true paths will be satisfied, the delays of the longest true paths after sizing are checked for setup time violations. If any timing violation exists, k is increased and the circuit is re-sized.

Linearized implementation for gate sizing

Thus, sizing implementation has four levels of computational complexities: 1) determination of false paths, 2) sizing considering all possible pairs, 3) sizing while meeting constraints for all possible true

paths, and 4) non-linearity of the optimization problem. While implementing the sizing method, false paths are identified using the sensitization criteria and maximum implications described in 3.2. All possible pairs of flip-flops are considered for gate sizing but the value of k (longest true paths) is limited to be less than 1000. This value of k is sufficient to generate feasible sizings for the benchmark circuits and keep the problem size small. Finally, instead of solving the non-linear optimization problem $\mathcal{GS}$ that minimizes $\sum_{i=1}^{n} \frac{1}{d_i}$, it is converted to linear optimization that maximized the function $\sum_{i=1}^{n} d_i$. Note that this change is made only to contain the computational complexity.

3.4.4 Experimental Results

The experiments are conducted on two sequential benchmark suites: *ISCAS89* and *ITC99*. Timing yields are obtained by Monte Carlo simulations that consider only true path delays to reflect the true timing yields. Instead of modeling process variations, such as channel length and threshold voltage variations, it is assumed that the variation sources all manifest themselves into gate delay variations, and all logic gates have an initial gate delay as a truncated Gaussian distribution with $(\mu, \sigma) = (1, 0.15)$ and $\kappa = 3$. Path delay uncertainties are approximated by $\sqrt{n}\,\sigma$, where n is the number of gates on a path.

Table 3.3 shows the results using traditional gate sizing followed by *Prop* clock scheduling, and using false-path-aware gate sizing and *fp-Prop* clock scheduling. In *Prop*, the statistical-timing-driven clock scheduling is performed using structural path delays. In *fp-Prop*, the statistical-timing-driven clock scheduling is performed using true path delays. In this table all columns are self explanatory except the columns marked “Ratio” which provide the ratios of circuit area after gate sizing as compared to the circuit area before gate sizing. Three major conclusions can be drawn from the experimental results:

1. The proposed design flow achieves significantly higher timing yields ($> 20\%$) than the traditional flow for five benchmark circuits with less than a 5% area overhead.

Circuits	CP^*	CP	$\sum d_i$	Traditional sizing + *Prop*			False-path-aware sizing + *fp-Prop*		
				Yield	$\sum d_i$	Ratio	Yield	$\sum d_i$	Ratio
b05s	45.0	45.70	864	**31.9%**	1209	71.5%	**99.9%**	1126	76.7%
b06	4.0	4.75	43	55.7%	45	95.6%	55.7%	45	95.6%
b07s	25.0	26.13	362	67.5%	428	84.6%	67.7%	428	84.6%
b10	9.0	9.68	155	62.8%	166	93.4%	62.8%	166	93.4%
b11s	29.0	30.21	437	71.6%	563	77.6%	**95.8%**	532	82.1%
b12	11.0	11.75	904	65.9%	1011	89.4%	**76.2%**	1003	90.1%
b13s	8.0	8.42	266	**26.9%**	299	89.0%	26.9%	299	89.0%
s1488	16.0	16.60	653	**51.2%**	769	84.9%	51.2%	769	84.9%
s5378	21.0	22.37	2779	**42.9%**	2844	97.7%	42.9%	2844	97.7%
s9234.1	38.0	39.85	5597	**49.3%**	6476	86.4%	49.3%	6476	86.4%
s13207.1	51.0	52.61	7951	78.9%	8224	96.7%	78.9%	8224	96.7%
s35932	28.0	31.17	16065	**44.0%**	19962	80.5%	**92.3%**	19674	81.7%
s38417	31.5	33.60	22179	**55.4%**	26214	84.6%	**81.1%**	26181	84.7%
s38584.1	48.0	49.56	19253	84.9%	20975	91.8%	84.9%	20970	91.8%

Table 3.3: Timing yield and area comparison between traditional and false-path-aware gate sizing flows.

2. Traditional gate sizing has serious (adverse) timing yield impact on the circuits b05s, s35932 and s38417, due to the reduced timing margins as illustrated in Figure 3.4. The proposed design flow only has little or no adverse impact on the timing yields of these circuits.

3. For b13s, s1488, s5378 and s9234.1, both the traditional flow and the proposed flow cause significant timing yield degradations. This can be improved by performing the gate sizing and clock scheduling iteratively.

Chapter 4

Clock Tree Topology Optimization

Reserving slacks for timing uncertainties ensures the correct functionality of manufactured chips. However, the increasing process-induced timing uncertainties and demand for slacks have made timing convergence difficult to achieve. Through clock tree topology optimization, the clock skew uncertainties on timing-critical paths are reduced and faster timing convergence is achieved. In this chapter, an enhanced bipartition algorithm for clock tree topology optimization is proposed. First, the sources of clock skew uncertainty and calculation of negative slack is discussed. The steps to combine both path timing and sequential element placement information in a *partition graph* are then presented. By controlling the edge weights of the partition graph, the clock sink pairs with tight timing constraints will be clustered together. With the reduction of clock skew uncertainties on timing-critical paths, the algorithm recovers the slacks reserved for clock skew uncertainties and reduces the total negative slack of a design. Finally, the experimental results show that the algorithm significantly reduces the total negative slack and speeds up the timing convergence of a design. For ISCAS89 benchmark circuits, the algorithm achieves up to a 67% total negative slack reduction compared to a traditional balanced bipartition algorithm.

4.1 Preliminaries

Clock distribution networks and logic designs are usually done in separate steps; separate slack requirements are imposed for path delay uncertainty and clock skew uncertainty. Therefore, the hold time and setup time constraints are as follows:

$$\alpha_{ij} + \mu(d_{ij}) \geq T^j_{hold} + \kappa[\sigma(\alpha_{ij}) + \sigma(d_{ij})], \tag{4.1}$$

$$CP - \mu(D_{ij}) - \alpha_{ij} \geq T^j_{setup} + \kappa[\sigma(\alpha_{ij}) + \sigma(D_{ij})]. \tag{4.2}$$

The terms $\kappa[\sigma(\alpha_{ij}) + \sigma(d_{ij})]$ and $\kappa[\sigma(\alpha_{ij}) + \sigma(D_{ij})]$ in (4.1) and (4.2) are the hold time and setup time slack requirements, which depend on the choice of κ. Due to the growing design size, a 6σ slack, or $\kappa = 6$, is required for each timing constraint to guarantee a reasonable timing yield. However, the soaring clock frequency (decreasing CP) and increasing process variations (increasing $\sigma(\cdot)$) have made the timing constraints (4.1) (4.2) difficult to satisfy.

When the hold time slack $s^h_{ij} = \alpha_{ij} + \mu(d_{ij}) - T^j_{hold}$ or the setup time slack $s^s_{ij} = CP - \mu(D_{ij}) - \alpha_{ij} - T^j_{setup}$ of α_{ij} is smaller than its slack requirement, the difference needs to be counted toward the *total negative slack* (TNS). TNS indicates the aggregated design effort that needs to be done to achieve timing convergence. Note that in Chapter 3, clock scheduling generates a clock schedule that maximizes the normalized slacks of each path. The clock schedule needs to be checked against (4.1) (4.2) to measure negative slacks and verify timing convergence.

The TNS of a design can be reduced by minimizing the clock skew uncertainties on the paths with negative slack. This can be done by improving the clock tree topology generation process. Let P_i and P_j be the clock distribution paths of FF_i and FF_j. Velenis et al. [58, 59] observe that the *common part* of P_i and P_j, or P^{com}_{ij}, do not contribute to the clock skew uncertainty of α_{ij}. As shown in Figure 4.1,

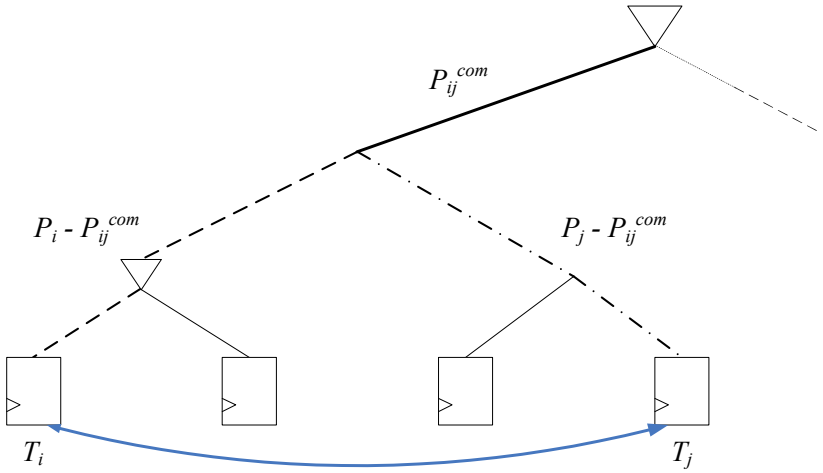


Figure 4.1: Clock skew uncertainty and clock distribution paths.

reducing the non-common part of P_i and P_j, $P_i - P_{ij}^{com}$ and $P_j - P_{ij}^{com}$, is likely to reduce $\sigma(\alpha_{ij})$. In light of the observation, a greedy algorithm that clusters source and target clock sinks of timing-critical paths is proposed to reduce the total clock skew uncertainty and total slack requirement of a design. However, the greedy approach that only takes timing information into consideration might generate an unbalanced clock tree topology. It can also result in an increase on total wire length of the clock tree, which inadvertently increases the total clock skew uncertainty and TNS.

4.2 Topology Design Objectives

The principle for reducing the clock skew uncertainty on a timing-critical path is to reduce the non-common part of the clock distribution paths to its source and target clock sinks. However, the total clock skew uncertainty as well as total clock power may increase if non-common clock distribution path reduction is not conducted carefully. Figure 4.2(a) shows the partitioning of eight clock sinks using a traditional balanced bipartition algorithm. The top two paths with the largest path delay uncertainty are shown by the two solid arcs in Figure 4.2(b). The three dashed arcs show the paths with slightly smaller

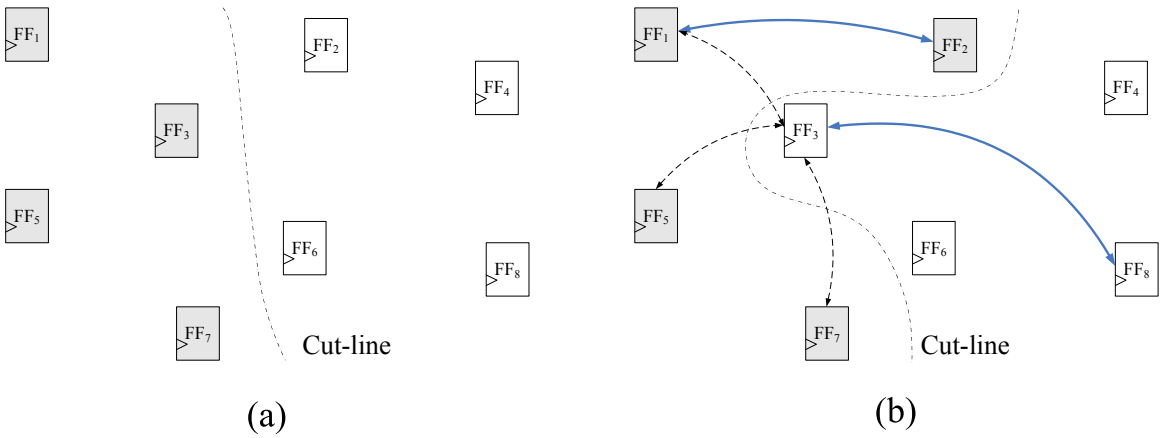


Figure 4.2: Total clock skew uncertainty increases due to non-common clock distribution path reduction.

path delay uncertainties. In Figure 4.2(b), FF_2 and FF_3 are moved to the left and right partitions to reduce the clock skew uncertainties on the two most timing-critical paths. However, this results in an increase on clock skew uncertainties of α_{31}, α_{35}, and α_{37}. As a result, the partitioning in Figure 4.2(b) may result in a larger TNS than that from the partitioning in Figure 4.2(a). Therefore, bipartition objectives should include the followings.

- **Balanced loading**

  Balanced loading ensures that no excessive snaking or buffering is required to balance the clock delays of both partitions. This not only prevents unnecessary increase on clock power but also reduces the clock delay of the final clock tree, which prevents an increase on total clock skew uncertainty.

- **Small extra wire length**

  The minimum achievable clock delay is likely to increase when the total wire length of a clock tree is increased by a large amount. Therefore, bipartitioning should not cause an excessive wire length increase.

- **Short non-common clock distribution paths**

  The clock skew uncertainty between a pair of clock sinks is likely to decrease if the non-common clock distribution paths to both clock sinks are short. Therefore, clock sinks of timing-critical paths should be partitioned into different groups as late as possible.

4.3 Partition-Based Clock Tree Topology Optimization

An enhanced recursive bipartition algorithm that generates binary clock tree topologies is proposed to overcome the drawbacks of greedy clustering-based algorithms. The algorithm can easily be extended to generate k-ary tree topologies by replacing bipartition with k-way partition. In each recursion step, *multiple* partition graphs are created from the given clock sinks and their timing constraints. A min-cut algorithm is then used to bipartition the graphs into balanced sub-graphs. The partitioning that has the minimum partition cost is then selected. Figure 4.3 shows the steps for determining the best partitioning in each iteration. Each step is explained in detail in the following sections.

4.3.1 Reference Set

A common technique for partitioning a set of nodes into multiple partitions is to first find the *reference set* of each partition [13, 82]. A reference set is a set of nodes that will most likely fall into the same partition. For example, two clock sinks that are far apart are likely to belong to different partitions. Therefore, a pair of reference sets, each containing one of the two clock sinks, is a common starting point for bipartitioning. The remaining clock sinks can then be partitioned based on which reference set they are closer to. The drawback of this approach is that the partitioning is easily biased by the positions of the two chosen clock sinks.

Chao et al. [13] observe that the clock sinks that fall on the bounding *octagon* are usually partitioned

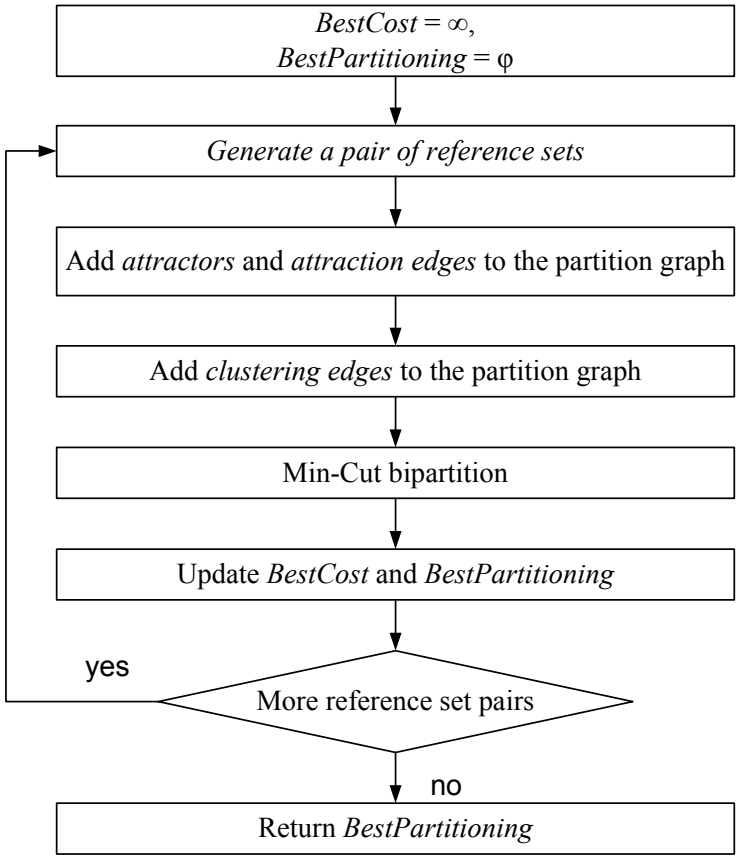


Figure 4.3: The steps in each bipartition iteration.

into two groups with consecutive elements. Let S be the set of clock sinks to be partitioned and $Oct(S)$ be the set of clock sinks that are on the bounding octagon. There are $|Oct(S)|$ ways to create a pair of reference sets with $\lfloor \frac{1}{2}|Oct(S)| \rfloor$ and $\lfloor \frac{1}{2}(|Oct(S)|+1) \rfloor$ clock sinks when $|Oct(S)|$ is an odd number and $\frac{|Oct(S)|}{2}$ ways when $|Oct(S)|$ is an even number. In Figure 4.4, the bounding octagon of the eight clock sinks is shown by the dashed lines. There are five clock sinks on the bounding octagon and one of the five reference set pairs is shown by the black and white clock sinks. Since each reference set usually contains more than one clock sink, the position of a clock sink will be less likely to bias the final partitioning. The enhanced bipartition algorithm adopts this reference set selection method.

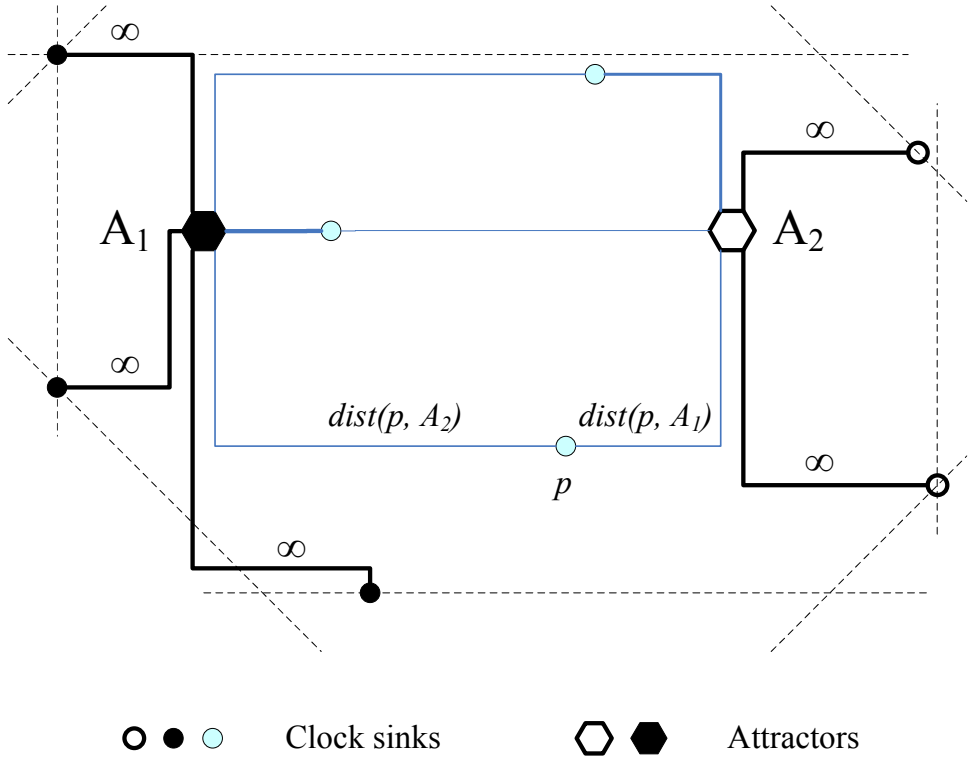


Figure 4.4: Illustration of reference set, bounding octagon, attractors and attraction edges.

There can be a large number of clock sinks lying on one of the the bounding octagon edges. For example, many clock sinks may be placed on one edge of the chip. In this case, only the first and the last clock sinks on this edge are included in $Oct(S)$. This heuristic avoids creating a pair of reference sets that are dis-proportional in their diameters.

4.3.2 Attractor and Attraction Weight

To ensure that the clock sinks in a reference set are grouped together, two artificial *attractors*, A_1 and A_2, are introduced into the partition graph for reference sets REF_1 and REF_2; an *attraction edge* with an infinite attraction weight from each reference set element to its attractor is then added. For each of the remaining clock sinks, two attraction edges, one to each attractor, are added. The attraction weight

is calculated based on the *distance metric* from the clock sink to the reference set of the attractor. The distance metric needs to reflect the wire length increase when the clock sink is grouped with the reference set. Chao et al. [13] use

$$dist(p, A_k) = \min_{r \in REF_k} dist(p, r) + \max_{r \in REF_k} dist(p, r) \tag{4.3}$$

to measure the distance between clock sink p and attractor A_k, where $dist(p, r)$ is the Manhattan distance between clock sinks p and r. The enhanced bipartition algorithm adopts the same distance metric.

When $dist(p, A_1) < dist(p, A_2)$, grouping p to REF_2 (cutting edge (p, A_1) instead of (p, A_2)) results in an increase of $\sim dist(p, A_2) - dist(p, A_1)$ to the total wire length. Therefore, one can assign $\frac{dist(p,A_2)-dist(p,A_1)}{2}$ and $\frac{dist(p,A_1)-dist(p,A_2)}{2}$ as the attraction weights of edges (p, A_1) and (p, A_2). Since many partitioning packages require non-negative edge weights, $\frac{dist(p,A_1)+dist(p,A_2)}{2}$ is added to both attraction weights. This is equivalent to using $dist(p, A_2)$ and $dist(p, A_1)$ as the attraction weights for edges (p, A_1) and (p, A_2). Figure 4.4 shows the reference sets, attractors, and attraction edges of eight clock sinks.

Setting the attraction weight to infinity for edges from reference set elements to their attractors can cause unsatisfactory results. This is because timing constraints are not taken into account when $Oct(S)$ is divided into two reference sets. Therefore, assigning an infinite edge weight hinders the optimization opportunites on these edges. A simple heuristic resolves the issue: all the clock sinks use the same attraction weight function (4.3).

Instead of assigning clock sinks to reference sets greedily, based solely on the distance measure, the distance information is kept in the edge weights of the partition graph. This enables wire length and timing constraints to be considered simultaneously as described in the next section.

4.3.3 Clustering Weight

The most important improvement of the enhanced bipartition algorithm over previous algorithms is that both clock sink positions and timing constraints are considered during partitioning. This is achieved by introducing a *clustering edge* between the two clock sinks of a timing path. Controlling the clustering weight determines when the two clock sinks are partitioned into different groups, which controls the TNS.

The negative slack of a path is determined by its path delay and clock skew uncertainties. Since clock skew uncertainty is not available during the topology optimization phase, an estimation method is developed as follows.

Let the *diameter* of S be the maximum distance of two clock sinks in S, or

$$Dia(S) = \max_{p,q \in S} dist(p, q). \tag{4.4}$$

Figure 4.5 shows $Dia(S)$ versus the maximum clock skew uncertainty ($\kappa\sigma(\alpha_{ij})$) within S, where each S is extracted from the subtrees of a zero-skew buffered clock tree of $s35932$. The clock skew uncertainty is approximated by a 30% clock delay of the non-common clock distribution path. The figure leads to the following two conclusions:

1. The maximum clock skew uncertainty among the sinks of a clock subtree increases roughly linearly as the diameter of the clock subtree increases.

2. For two sets of clock sinks with the same diameter, the difference of their maximum clock skew uncertainties is within a certain range.

The second conclusion is contributed to by the following sources.

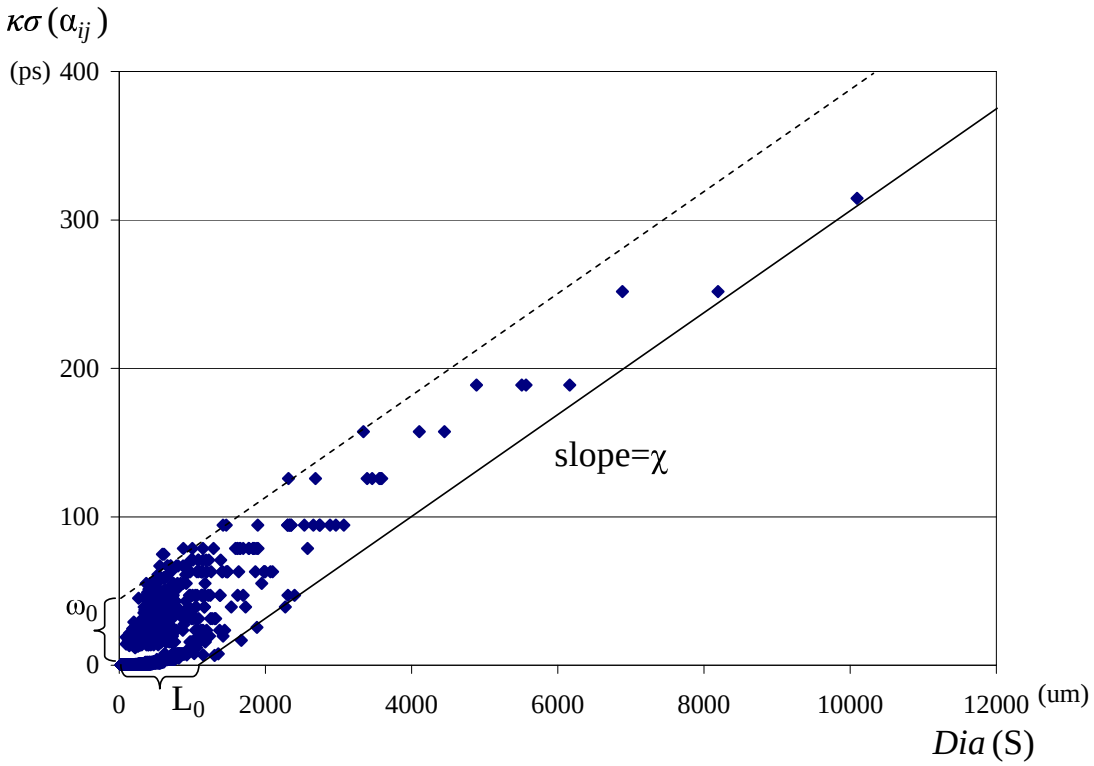


Figure 4.5: Diameter of S versus maximum clock skew uncertainty in $s35932$.

- The clock sink density is not uniform across the chip. Two clock subtrees with the same diameter may contain different numbers of clock sinks and have different clock delays.

- Different buffer insertion, buffer sizing and wire sizing on the two clock subtrees may generate different clock delays even if the two clock subtrees have the same topology.

Accounting for clock sink density and clock tree optimization effects on clock skew uncertainties during clock tree topology optimization is not practical. However, a clock skew uncertainty estimation method can be derived from Figure 4.5.

The *worst clock skew uncertainty* of S is estimated as follows.

$$\omega(S) = \omega_0 + \chi Dia(S). \tag{4.5}$$

The *best-effort clock skew uncertainty* between clock sinks i and j is estimated as follows.

$$\beta_{ij} = max\{0, \chi[dist(i,j) - L_0]\}. \tag{4.6}$$

The parameters ω_0, L_0 and χ depend on the clock sink density of the chip as well as interconnect and clock buffer parameters. They can be obtained by analyzing a zero-skew clock tree generated by zero-skew buffered clock tree optimization algorithms as illustrated in Figure 4.5. In short, $\omega(S)$ and β_{ij} are the pessimistic and optimistic estimations of $\kappa\sigma(\alpha_{ij})$.

Let the *equivalent hold time slack* of α_{ij} be

$$u_{ij}^h = max(s_{ij}^h, \beta_{ij} + \kappa\sigma(d_{ij})), \tag{4.7}$$

and the *equivalent setup time slack* of s_{ij} be

$$u_{ij}^s = max(s_{ij}^s, \beta_{ij} + \kappa\sigma(D_{ij})). \tag{4.8}$$

If $s_{ij}^h < u_{ij}^h$, a negative hold time slack of at least $u_{ij}^h - s_{ij}^h$ is unavoidable; a path with a hold time slack of s_{ij}^h should be treated the same as a path with a hold time slack of u_{ij}^h. Since the worst clock skew uncertainty of S may occur on any pair of clock sinks among S (recall the example in Figure 1.1), the *maximum amount of negative slack that may be avoided* by not partitioning a clustering edge is $[\omega(S) + \kappa\sigma(d_{ij})] - u_{ij}^h$ for the hold time constraint and $[\omega(S) + \kappa\sigma(D_{ij})] - u_{ij}^s$ for the setup time constraint. Thus, the clustering weight functions for hold time and setup time constraints are defined as follows:

$$W_{ij}^h = max\left\{0, \frac{Dia(S)}{M} \times \frac{\omega(S) + \kappa\sigma(d_{ij}) - u_{ij}^h}{\omega(S)}\right\}, \tag{4.9}$$

$$W_{ij}^s = max\left\{0, \frac{Dia(S)}{M} \times \frac{\omega(S) + \kappa\sigma(D_{ij}) - u_{ij}^s}{\omega(S)}\right\}. \tag{4.10}$$

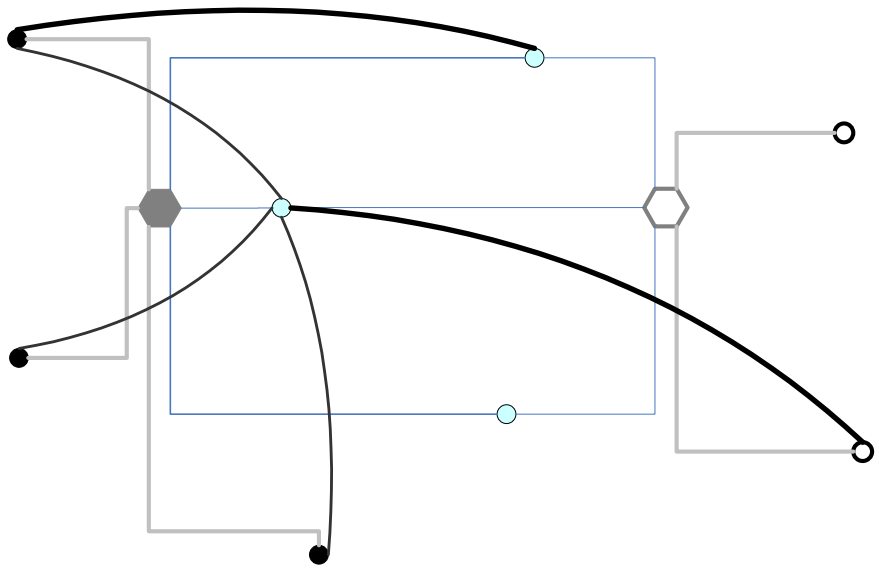


Figure 4.6: A complete partition graph with attraction edges and clustering edges.

To account for both hold time and setup time constraints, the clustering weight is defined as the average of W_{ij}^h and W_{ij}^s. It is also possible to put more weight on W_{ij}^h or W_{ij}^s to target specifically on negative hold time slack or negative setup time slack reduction.

Multiplying the potential negative slack savings by $\frac{Dia(S)}{\omega(S)}$ scales the clustering weights to have the same unit as attraction weights. The relative importance of timing constraints (clustering weights) over wire length (attraction weights) is then controlled by M. The optimal M depends on how much TNS the design has and the amount of wire length increase designers are willing to tolerate. It can be determined empirically by running the enhanced bipartition algorithm repeatedly with an increasing M.

4.3.4 Min-Cut Bipartition

The set of clock sinks S, two attractors, attraction edges, and clustering edges form a partition graph. The complete partition graph of the example circuit in Figure 4.2(b) is shown in Figure 4.6. The topology optimization problem becomes a sequence of standard min-cut problems on partition graphs. The

balanced loading objective is handled by assigning vertex weights according to clock sink capacitances (attractors have zero vertex weight) and enforcing a cut-ratio close to one. An efficient and publicly available partition tool, METIS [83], is used to find the cut line for a partition graph. The complete clock tree topology is obtained by recursive bipartitioning.

4.4 Experimental Results

The enhanced bipartition algorithm is implemented in C++ and executed on a 1.7GHz 512MB Pentium-M laptop computer. The ISCAS89 benchmark circuits are synthesized by SIS [84] with a $180nm$ cell library and placed by Dragon [85]. The clock tree topologies from both the enhanced bipartition algorithm and a traditional balanced bipartition algorithm [13] are taken as the inputs of the Deferred Merge Embedding (DME) algorithm [13] and unbuffered zero-skew clock trees are generated. The clock trees are then optimized for minimum clock delay using the zero-skew buffered clock tree optimization algorithm from Chapter 2. For path timing analysis, an independent Gaussian distribution with $(\mu, \sigma) = (50ps, 10ps)$ is used for all gate delays. A 6σ slack requirement is imposed for path delay uncertainty and clock skew uncertainty. For example, the slack requirement for a path with n gates is $6\sqrt{n}\sigma$. The 6σ slack requirement for clock skew uncertainty is approximated by a 30% clock delay of the non-common clock distribution path.

4.4.1 Comparison and Analysis

Table 4.1 shows the analysis results of the clock trees using the traditional balanced bipartition algorithm. The analysis is done using the method described in 4.3.3 and the parameters ω_0, L_0 and χ are used to guide the enhanced bipartition algorithm. It is found that $M = 10$ gives good results without introducing too much extra wire length. In $s1488$ and $s9234.1$, the TNS is contributed by only a few timing

Circuit	# FF	Wire Length (mm)	Delay (ns)	Cap. (pF)	TNS (ns)	# violations (hold/setup)	ω_0 (ps)	L_0 (μm)	χ ($ps/\mu m$)
s1488	33	6.383	0.073	1.603	0.182	0/10	0	500	0.033
s5378	263	37.627	0.218	10.680	3.491	18/80	10	500	0.036
s9234.1	286	47.636	0.291	12.195	5.571	0/64	10	550	0.044
s13207.1	852	120.267	0.436	33.213	8.944	84/117	10	700	0.041
s35932	2083	371.426	0.872	103.382	598.378	2931/1981	20	800	0.034
s38584.1	1768	322.541	0.872	84.230	106.571	675/1018	20	900	0.045

Table 4.1: Analysis results of the clock trees using the traditional balanced bipartition algorithm.

Circuit	W.L. (mm)	Chg.	Delay (ns)	Chg.	Cap. (pF)	Chg.	TNS (ns)	Chg.	# violations (hold/setup)	CPU (s)
s1488	6.996	9.6%	0.073	0.0%	1.681	4.9%	0.182	0.0%	0/10	4s
s5378	40.175	6.8%	0.218	0.0%	11.498	7.7%	3.297	-5.6%	17/78	31s
s9234.1	49.538	4.0%	0.218	-25.0%	13.528	10.9%	4.346	-22.0%	0/52	42s
s13207.1	126.386	5.1%	0.436	0.0%	34.066	2.6%	7.675	-14.2%	89/105	118s
s35932	385.711	3.8%	0.872	0.0%	100.777	-2.5%	572.904	-4.3%	2717/1717	295s
s38584.1	352.745	9.4%	0.727	-16.7%	93.715	11.3%	34.957	-67.2%	533/208	255s

Table 4.2: Experimental results of the enhanced bipartition algorithm.

violations. Moreover, several circuits have a predominant number of hold time or setup time violations.

As shown in Table 4.2, the enhanced bipartition algorithm achieves TNS reductions up to 67.2% with less than an 11.3% increase in wire length and clock power. In general, both the TNS and the number of timing violations are reduced after optimization; the algorithm does not replace a large timing violation with multiple small timing violations. The decomposition of the TNS is shown in Figure 4.7, in which the TNS from the traditional algorithm is normalized to one. Although the algorithm increases wire length and switching capacitance, this does not necessarily increase the clock delay. In fact, better clock delays are achieved on $s9234.1$ and $s38584.1$. However, wire length increase usually results in clock power increase.

Figure 4.8 shows the clock trees of $s13207.1$ generated by the traditional and enhanced bipartition algorithms. It can be seen that the local clock tree structures generated by the enhanced bipartition algo-

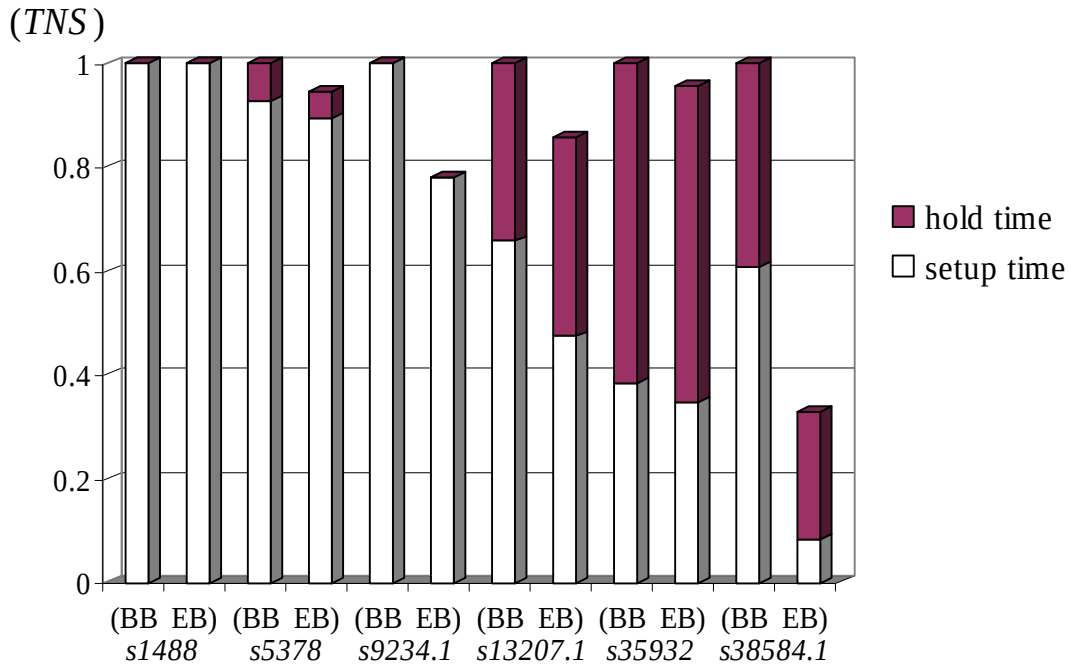
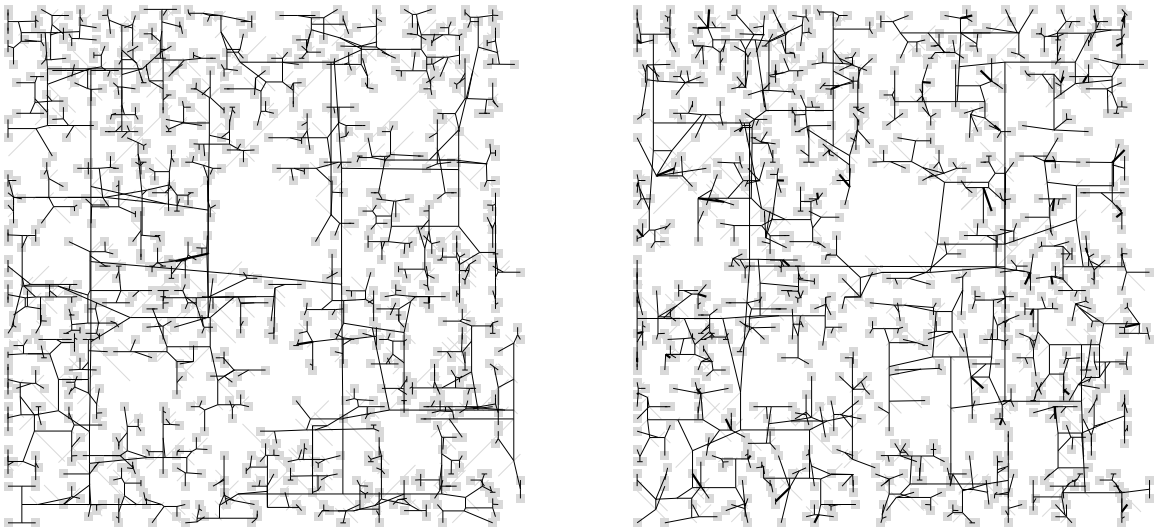


Figure 4.7: TNS improvement analysis.

rithm are slightly different from those generated by the traditional method. For example, the enhanced bipartition algorithm sometimes generates unbalanced partitions to reduce clock skew uncertainty. To balance the clock load, wire snakings (shown as dark segments) are required. Although this increases the total wire length and clock power, the percentage of TNS reduction exceeds that of wire length and clock power increase in four of the six circuits.

Analysis on partitioning results shows that METIS sometimes does not produce good partitionings. In other words, clock sinks may be grouped to wrong partitions and cause unnecessary wire length increases. It is expected that further TNS improvements can be achieved by using more sophisticated partition algorithms such as MLPart [86].



(a) Traditional Bipartition (b) Enhanced Bipartition

Figure 4.8: The clock trees of $s13207.1$ using traditional and enhanced bipartition algorithms.

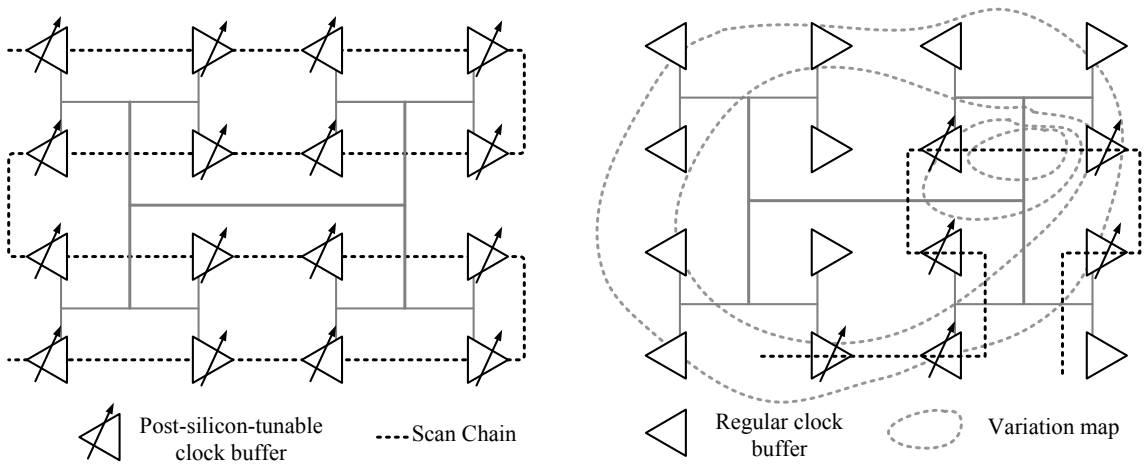
4.4.2 Runtime

Although min-cut problems are NP-Complete problems, they are usually solved very efficiently with near-optimal partition costs. In the experiments, each partition graph is first written to a file and then read by METIS. METIS then writes the partitioning result to another file, which is then read back into the main program. Even with the high disk I/O overhead, the runtime of the enhanced bipartition algorithm is still less than five minutes for the largest benchmark circuit. This runtime is much smaller than the runtime for the subsequent clock tree optimization step. Therefore, the enhanced bipartition algorithm scales well for large problems.

Chapter 5

Post-Silicon Clock Tuning

The current correct-by-construction design principle tackles timing issues by reserving more slacks for larger timing uncertainties. This approach has limited the design productivity due to the growing timing uncertainties in nanometer technologies. An alternative to maximize the timing yield without slowing down the timing convergence is to perform timing adjustments for the chips that do not pass the test. This can be realized by a PST clock tree. A PST clock tree contains programmable clock buffers that allow clock tuning for timing corrections in the post-silicon stage. Existing design approaches for PST clock tree synthesis usually insert a PST clock buffer for each flip-flop or put PST clock buffers in an entire level of a clock tree. This can cause significant over-design and lead to a long tuning time. In this chapter, a bottom-up algorithm is used to identify candidate PST clock buffer locations. Two optimization algorithms are proposed to insert PST clock buffers at both internal and leaf nodes of a clock tree. The algorithms are driven by statistical timing analysis to reduce the hardware cost of a PST clock tree. Experimental results on ISCAS89 benchmark circuits show that the algorithms achieve up to a 90% area or a 90% number of PST clock buffer reductions compared to existing design methods.



(a) Brute-force design

(b) Statistical-timing-driven design

Figure 5.1: Hardware cost reduction through statistical timing analysis.

5.1 Introduction

Post-silicon clock tuning not only improves the timing yield but also reduces clock power by avoiding using grid-based clock distribution networks. However, a brute-force design method that inserts a PST clock buffer for each flip-flop or at each clock tree terminal uses a significant amount of the chip area. There is no systematic way in the literature to construct a PST clock tree that *provides the maximum tuning capability for timing yield improvements with minimum hardware cost*.

Statistical timing analysis is a powerful tool to predict the performance of manufactured chips during design time. A PST clock tree synthesis flow can take the information from statistical timing analysis and reduces the hardware cost of PST clock trees. As illustrated in Figure 5.1, by analyzing the effect of process variations on timing, PST clock buffers can be inserted selectively only at the critical locations in a clock tree. This can greatly reduce the hardware cost of a PST clock tree.

In this chapter, the effect of PST clock buffers on timing yield is studied and two optimization al-

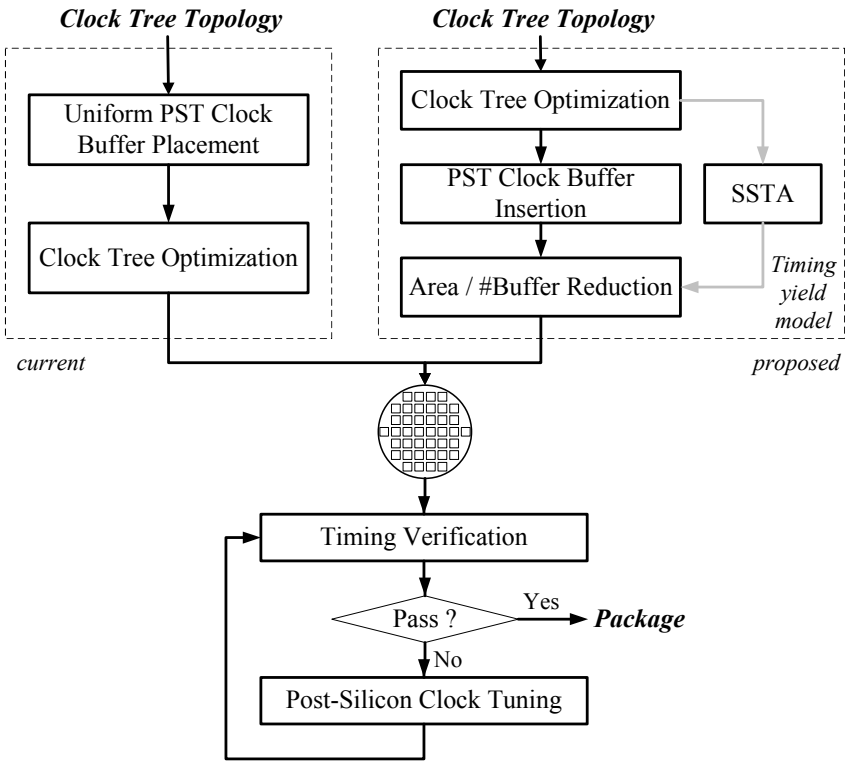


Figure 5.2: PST clock tree synthesis flow.

gorithms for reducing the area or number of PST clock buffers are proposed. Compared with existing design methods, the proposed algorithms achieve up to a 90% hardware cost reduction. Figure 5.2 shows the steps of the proposed PST clock tree synthesis flow.

The rest of this chapter is organized as follows: in 5.2, two PST clock tree synthesis formulations, PST-A and PST-N, are defined based on two hardware cost metrics. Section 5.3 provides a timing yield model for sequential circuits using statistical timing analysis and Monte Carlo integration. The effect of PST clock buffers on timing yield is analyzed and a timing yield model in the presence of PST clock buffers is developed in 5.4. An iterative linesearch algorithm utilizing a fast gradient approximation algorithm is proposed in 5.5 to solve PST-A. A batch selection algorithm is proposed in 5.6 to solve PST-N. Experimental results on the effectiveness of the proposed algorithms are demonstrated in 5.7.

5.2 Problem Formulation

The tuning capability of a PST clock tree is dependent on the number of PST clock buffers and their tunable range. To achieve the maximum timing yield improvement, a brute-force method may insert PST clock buffers at every terminal of the clock tree and design the PST clock buffer to have a large tunable range. However, this method would increase the design's hardware costs significantly.

To optimize the hardware cost of a PST clock tree, it is essential to choose a cost metric that reflects the actual silicon cost. There are several PST clock buffer designs that achieve variable clock delay with very different circuit design techniques. A common PST clock buffer design consists of two inverters with a bank of passive loads in between [9, 68]. Each passive load can be connected or disconnected to the inverter by programming the control bit of its pass gate. This type of PST clock buffer relies on RC delay to control the clock delay. To achieve a large tunable range, a large passive load is required. Since on-chip capacitors require a large area in a digital VLSI process, the appropriate cost metric for this type of PST clock buffer is the area required to implement the passive loads, which is proportional to the required tunable range of the buffer. Another PST clock buffer design achieves variable delay by changing the driving strength of a buffer. This is either done through controlling the bias voltage of the driver with a digital-analog-converter [64] or introducing contention to the driver [11]. For this type of design, the hardware cost is insensitive to the tunable range and can be treated as a constant. Therefore, the appropriate cost metric for this type of design is the total number of PST clock buffers in the clock tree. Both metrics, *total tunable range* and *total number of PST clock buffers*, for the hardware cost are adopted and two PST clock tree synthesis problems are defined as follows.

Problem PST-A: (To minimize total tunable range)

Given a circuit and its buffered clock tree, determine the required tunable range of each clock buffer such

that the total tunable range is minimized and the target timing yield is achieved.

Problem PST-N: (To minimize the number of PST clock buffers)

Given a circuit and its buffered clock tree, select a minimum subset of clock buffers such that the target timing yield is achieved when the selected clock buffers are converted to PST clock buffers.

The PST-A and PST-N problems require very different optimization approaches but are driven by the same timing yield model. In the following sections, timing yield models for circuits with and without PST clock buffers are studied. Two optimization algorithms are then proposed to solve the PST-A and PST-N problems.

5.3 Timing Yield Model

5.3.1 Timing Constraints and Slack Vector

A sequential circuit is represented by its circuit graph $G = (B, V, E)$, where B is the set of clock buffers, V is the set of flip-flops, and E is the set of timing arcs where e_{ij} indicates that there are combinational paths between i and j. The clock skew between i and j is defined as $\alpha_{ij} = T_i - T_j$, where T_i is the clock arrival time at i. The maximum and minimum path delays from i to j are denoted D_{ij} and d_{ij}.

A circuit needs to satisfy hold time and setup time constraints:

$$\alpha_{ij} + d_{ij} \geq T^j_{hold}, \tag{5.1}$$

$$\alpha_{ij} + D_{ij} \leq CP - T^j_{setup}, \tag{5.2}$$

where T^j_{hold} and T^j_{setup} are the hold time and setup time of flip-flop j and CP is the clock period. Define the hold time slack of (5.1) as $s^h_{ij} = \alpha_{ij} + d_{ij} - T^j_{hold}$ and the setup time slack as $s^s_{ij} = CP - D_{ij} - \alpha_{ij} - T^j_{setup}$ and collect all the slack variables as an $R^{2|E|\times 1}$ *slack vector* $\mathbf{s}$. A circuit satisfies all the

timing constraints if

$$\begin{aligned} \mathbf{s} &\in \mathcal{C}_0, \\ \mathcal{C}_0 &= \{\mathbf{w} \mid w_i \geq 0, i = 1 \ldots 2|E|\}. \end{aligned} \tag{5.3}$$

In other words, a circuit is free from timing failures if its slack vector is in the *feasible region* $\mathcal{C}_0$, which is the non-negative orthant.

Recent statistical timing analysis research has shown that a delay variable d can be represented in a compact and accurate *canonical delay model* [87–89]:

$$d = \mu_d + [\beta_{d,1}\beta_{d,2} \ldots \beta_{d,l}] \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_l \end{bmatrix} = \mu_d + \beta_d \mathbf{f}, \tag{5.4}$$

where μ_d is the mean value of d, $f_1 \ldots f_l$ are global and local variation sources and $\Sigma_{\mathbf{f}}$ is their covariance matrix. The variation sources can be correlated or uncorrelated Gaussian random variables, and $\beta_{d,1} \ldots \beta_{d,l}$ are the sensitivities of d to the variation sources. Therefore, the slack vector can be represented as

$$\mathbf{s} = \mu_{\mathbf{s}} + A\mathbf{f}, \tag{5.5}$$

where A is the $R^{2|E| \times l}$ sensitivity matrix of $\mathbf{s}$. The covariance of s_i and s_j can be computed by [1]

$$\begin{aligned} cov(s_i, s_j) &= E\left[(s_i - \mu_{s_i})(s_j - \mu_{s_j})^T\right] \\ &= \mathbf{a_i} E\left[\mathbf{f}\ \mathbf{f}^T\right] \mathbf{a_j}^T \\ &= \mathbf{a_i} \Sigma_{\mathbf{f}} \mathbf{a_j}^T, \end{aligned} \tag{5.6}$$

[1] The subscripts of s_i and s_j are their indices in the slack vector $\mathbf{s}$.

where $\mathbf{a_i}$ is the i-th row of A. Therefore, the covariance matrix of $\mathbf{s}$ is

$$\Sigma_{\mathbf{s}} = A\Sigma_{\mathbf{f}}A^T, \tag{5.7}$$

and the slack vector can be represented by a multivariate Gaussian distribution

$$\mathbf{s} \sim N\left(\mu_{\mathbf{s}}, \Sigma_{\mathbf{s}}\right). \tag{5.8}$$

5.3.2 Slack Filtering

The dimension of $\mathbf{s}$ is $2|E|$, which can be very large for large circuits. However, many of the timing paths have abundant slack and do not contribute to the timing yield loss. Therefore, it is desirable to filter out non-critical slack variables to reduce the dimension of the slack vector. A slack filtering criteria is as follows:

$$\frac{\mu_{s_i}}{\sigma_{s_i}} \geq p. \tag{5.9}$$

For s_i satisfying (5.9), the i-th rows of $\mathbf{s}$, $\mu_{\mathbf{s}}$, $\Sigma_{\mathbf{s}}$, as well as the i-th column of $\Sigma_{\mathbf{s}}$, are deleted. This brings down the dimension of the slack vector to a manageable size n. Alternatively, n can be controlled by selecting a threshold value p.

5.3.3 Timing Yield Estimation

The *nominal* timing yield of the circuit is

$$\begin{aligned} \mathcal{Y}_0 &= P(\mathbf{s} \in \mathcal{C}_0) \\ &= \int_{\mathcal{C}_0} \cdots \int jpdf(s_1, s_2, \ldots, s_n) ds_1 ds_2 \ldots ds_n, \end{aligned} \tag{5.10}$$

where $jpdf(\mathbf{s})$ is the joint probability density function of $\mathbf{s}$. Since the slack variables are correlated, it is difficult to perform multi-dimensional integration analytically to obtain the timing yield. Instead,

Monte Carlo integration, which is an efficient method to calculate high dimensional integrals, is used to obtain timing yield estimations. First, N slack vector samples are generated according to $\mu_{\mathbf{s}}$ and $\Sigma_{\mathbf{s}}$. The nominal timing yield is then estimated by

$$\mathcal{Y}_0 \cong \frac{N_0}{N}, \tag{5.11}$$

where N_0 is the number of samples that falls in $\mathcal{C}_0$.

From a statistical point of view, $\mu_{\mathbf{s}}$ determines the center of the slack samples and $\Sigma_{\mathbf{s}}$ determines the distribution of the slack samples around the center. Applying eigenvalue decomposition on $\Sigma_{\mathbf{s}}$ gives the following equation:

$$\Sigma_{\mathbf{s}} = WDW^T, W^TW = I. \tag{5.12}$$

Generating slack vector samples according to $jpdf(\mathbf{s})$ involves two steps. First, samples of $\mathbf{x}$, a vector composed of independent standard normal distributions, are generated. A linear transformation is then applied to each sample according to the following equation:

$$\mathbf{z} = \mu_{\mathbf{s}} + W\sqrt{D}\mathbf{x}. \tag{5.13}$$

It is easy to verify that $\mathbf{z}$ follows $jpdf(\mathbf{s})$ as follows.

$$\begin{aligned} E[\mathbf{z}] &= \mu_{\mathbf{s}} + W\sqrt{D}E[\mathbf{x}] = \mu_{\mathbf{s}}. \\ E[(\mathbf{z} - E[\mathbf{z}])(\mathbf{z} - E[\mathbf{z}])^T] &= E[W\sqrt{D}\mathbf{x}\mathbf{x}^T\sqrt{D}^T W^T] \\ &= W\sqrt{D}E[\mathbf{x}\mathbf{x}^T]\sqrt{D}^T W^T \\ &= WDW^T = \Sigma_{\mathbf{s}}. \end{aligned}$$

Figure 5.3 illustrates the process to generate slack vector samples with $\mu_{\mathbf{s}} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$ and $\Sigma_{\mathbf{s}} =$

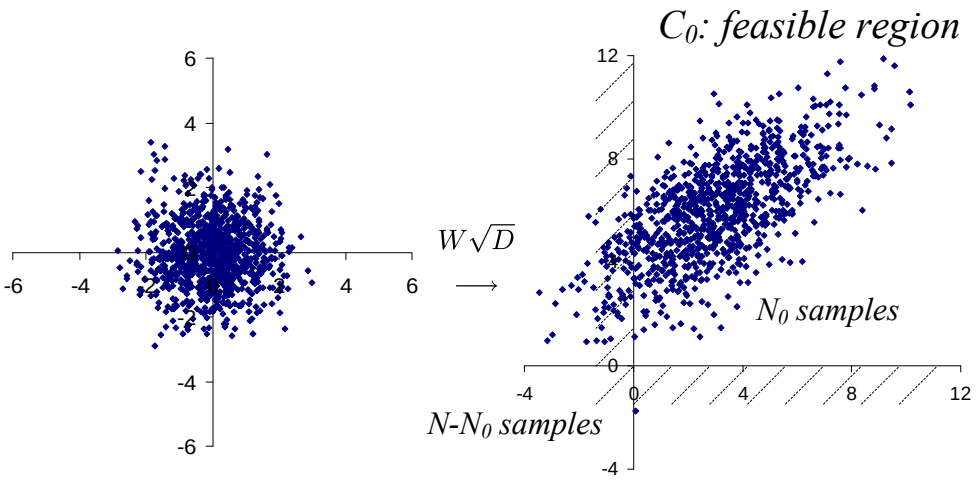


Figure 5.3: Generation of slack vector samples for timing yield estimation.

$\begin{bmatrix} 5 & 3 \\ 3 & 4 \end{bmatrix}$. Checking to determine whether a sample $\mathbf{s}$ falls in $\mathcal{C}_0$ is straight-forward. It is sufficient to ensure that every element in $\mathbf{s}$ is non-negative. It is worth noting that there are other high dimensional integration methods, such as the *parallelepiped*, *ellipsoid*, or *binding probability* methods, for timing yield estimation [90]. However, these methods have their restrictions and Monte Carlo integration is a competitive method even without slack filtering.

5.4 Timing Yield with PST Clock Buffers

PST clock buffers can be used to redistribute path slacks among adjacent timing paths and possibly to correct timing violations. The effect of PST clock buffers on timing yield is studied and a timing yield model in the presence of PST clock buffers is developed. The optimal timing yield that can be achieved through post-silicon clock tuning can be estimated efficiently using the derived model.

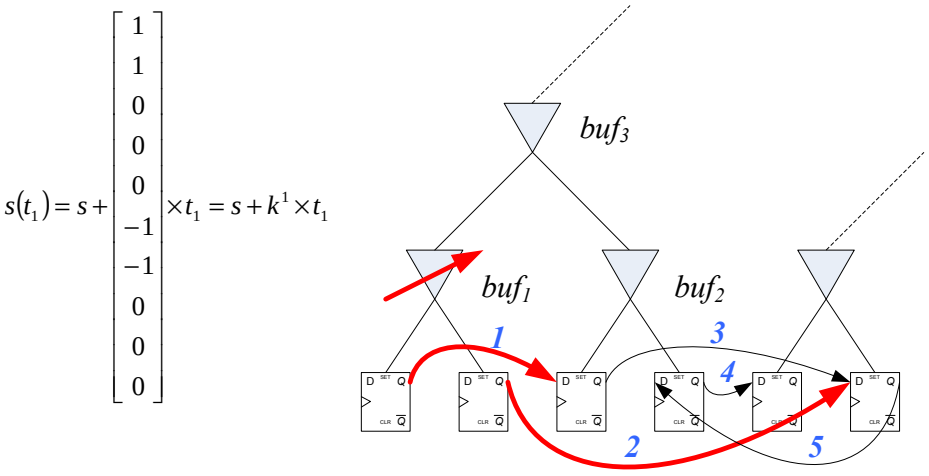


Figure 5.4: Effect of changing buf_1 delay on the slack vector.

5.4.1 PST Clock Buffer and Slack Vector

Figure 5.4 shows a circuit with five timing-critical paths. When buf_1 is converted to a PST clock buffer, the hold time and setup time slacks of paths 1 and 2 can be changed by adjusting the delay of buf_1. Let $s_1 \sim s_5$ be the hold time slacks and $s_6 \sim s_{10}$ be the setup time slacks of paths $1 \sim 5$. The slack vector after a change of t_1 on the buf_1 delay, $s(t_1) = s + \mathbf{k^1} t_1$, is shown in Figure 5.4. Likewise, the slack vector after a change of t_2 on buf_2 delay, $s(t_2) = s + \mathbf{k^2} t_2$, is shown in Figure 5.5.

Similar analysis shows that the effects of tuning the delays of internal clock buffers can be represented by a linear combination of the effects of tuning leaf level clock buffers. As shown in Figure 5.6, the slack vector after adding t_3 to the buf_3 delay, $s(t_3) = s + \mathbf{k^3} t_3 = s + (\mathbf{k^1} + \mathbf{k^2}) t_3$, is equivalent to adding t_3 to both buf_1 and buf_2 delays.

$$s(t_2) = s + \begin{bmatrix} -1 \\ 0 \\ 1 \\ 1 \\ -1 \\ 1 \\ 0 \\ -1 \\ -1 \\ 1 \end{bmatrix} \times t_2 = s + k^2 \times t_2$$

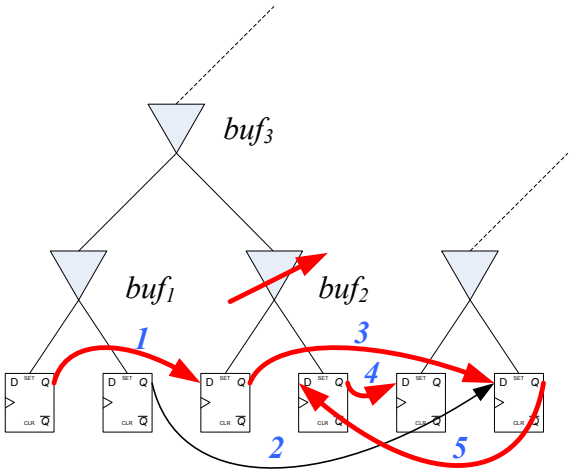


Figure 5.5: Effect of changing buf_2 delay on the slack vector.

$$s(t_3) = s + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ -1 \\ 0 \\ -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \times t_3 = s + \left(k^1 + k^2\right) \times t_3$$

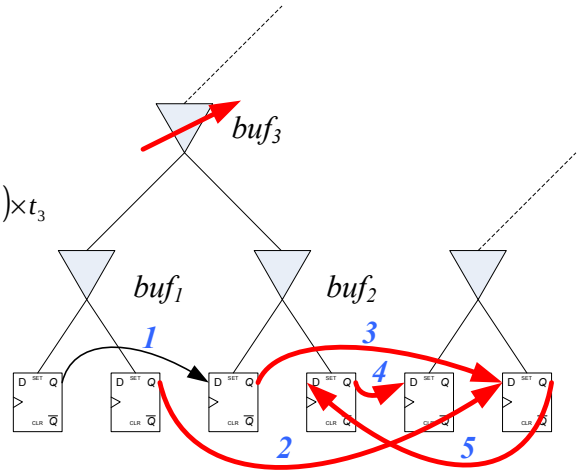


Figure 5.6: Effect of changing buf_3 delay on the slack vector.

5.4.2 Tuning Vector and Buffer Filtering

Let $\mathbf{t}$ be an $R^{|B|\times 1}$ vector corresponding to the tuning amount of the B clock buffers, the slack vector after applying $\mathbf{t}$ is

$$s(\mathbf{t}) = \mathbf{s} + K\mathbf{t}, \tag{5.14}$$

where K is the *tuning matrix*. The i-th column vector of K, $\mathbf{k^i}$, is the *tuning vector* of the i-th clock buffer.

After slack filtering, some clock buffers are not connected to timing-critical paths and their corresponding tuning vectors are zero vectors. Moreover, tuning the delay of a clock buffer may have the same effect as tuning another clock buffer. Therefore, these clock buffers can be filtered out to reduce the number of candidate PST clock buffer locations from $|B|$ to m. Figure 5.7 shows a bottom-up algorithm, which produces the reduced tuning matrix K and the corresponding candidate PST clock buffers U. The algorithm propagates tuning vectors upward toward the root node and filters out the buffers with zero or duplicate tuning vectors. Figure 5.8 illustrates the result of the algorithm on an example circuit.

5.4.3 Parameterized and Optimal Timing Yield

With post-silicon clock tuning, a circuit is considered functional if there exists a delay configuration $\mathbf{t}$ that can bring its slack vector to the feasible region $\mathcal{C}_0$. An alternative view is to regard the effect of post-silicon clock tuning as an enlargement on the feasible region. Let r_i be the tunable range of the i-th candidate clock buffer, the *parameterized* timing yield that can be achieved given the tunable range

Procedure: *SelectCandidate*

Input: Circuit graph $G(V, E, B)$, slack vector $\mathbf{s}$

Output: Tuning matrix K, candidate buffers U

Number clock buffers in *reverse* topological order

$m = 0,\ K \leftarrow \phi,\ U \leftarrow \phi$

for $i = 1 \ldots |B|$ **do**

  if buf_i *is a leaf buffer* **then**

$$\mathbf{v^i}_j = \begin{cases} 0 & \text{, when } buf_i \text{ is not connected to } s_j \\ +1 & \text{, when } buf_i \text{ drives the source (target) of hold-time (setup-time) slack } s_j \\ -1 & \text{, when } buf_i \text{ drives the target (source) of hold-time (setup-time) slack } s_j \end{cases}$$

    if $\mathbf{v^i} \neq 0$ **then**

      $m = m + 1,\ K \leftarrow [K|\mathbf{v^i}],\ U \leftarrow U \cup \{buf_i\}$

    end

  else

$$\mathbf{v^i} = \sum_{b \in Child(i)} \mathbf{v^b}$$

    if $\mathbf{v^i} \neq 0$ *and* $\mathbf{v^i} \neq \mathbf{v^b},\ \forall b \in Child(i)$ **then**

      $m = m + 1,\ K \leftarrow [K|\mathbf{v^i}],\ U \leftarrow U \cup \{buf_i\}$

    end

  end

end

Figure 5.7: Algorithm to select candidate PST clock buffer locations and generate tuning matrix.

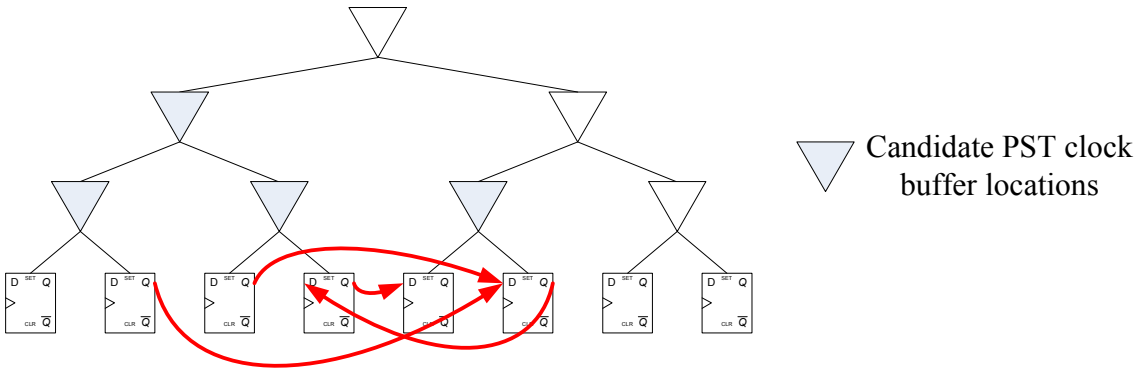


Figure 5.8: Candidate PST clock buffer locations identified by *SelectCandidate*.

vector $\mathbf{r}$ is:

$$\begin{aligned} \mathcal{Y}(\mathbf{r}) &= P(\mathbf{s} \in \mathcal{C}(\mathbf{r})), \\ \mathcal{C}(\mathbf{r}) &= \left\{ \begin{array}{lll} \mathbf{w} & | & \mathbf{w} = \mathbf{y} - K\mathbf{t}, \\ & & y_i \geq 0, i = 1 \ldots n \\ & & r_j \geq t_j \geq -r_j, j = 1 \ldots m \end{array} \right\} \\ &= \left\{ \mathbf{w} \mid \mathbf{w} \succeq -K\mathbf{t}, \mathbf{r} \succeq \mathbf{t} \succeq -\mathbf{r} \right\}, \end{aligned} \tag{5.15}$$

where $\succeq$ and $\preceq$ are element-wise inequalities. Note that the model (5.15) is applicable whether $\mathbf{r}$ is a continuous vector or a discrete vector. Therefore, the same parameterized timing yield model can be used both for PST-A and PST-N.

To check if a slack vector sample $\mathbf{s}$ is in the feasible region $\mathcal{C}(\mathbf{r})$, one needs to solve the linear feasibility problem:

$$\begin{array}{lll} (\mathcal{FP}) \quad \min & 0 \\ s.t. & -K\mathbf{t} \preceq \mathbf{s}, \\ & \mathbf{r} \succeq \mathbf{t} \succeq -\mathbf{r}, \end{array} \tag{5.16}$$

and check to see if (5.16) is feasible. CLP, a high quality Simplex [79] solver of the COIN-OR project [91], is used to solve the feasibility problem for each slack vector sample and get the parameterized timing yield estimation. Note that although the parameterized timing yield is more costly to obtain than the nominal timing yield, the runtime to solve 100,000 instances of the feasibility problem (5.16) with $n \sim 1000$ and $m \sim 2000$ is still less than an hour on a 1.7GHz Pentium-M PC. This is due to the high efficiency of the Simplex algorithm [80].

The *optimal* timing yield can be approximated by assuming all candidate clock buffers have a $(+\infty, -\infty)$

tunable range, or

$$\begin{aligned} \mathcal{Y}_* &= P(\mathbf{s} \in \mathcal{C}_*), \\ \mathcal{C}_* &= \{\mathbf{w} \mid \mathbf{w} \succeq -K\mathbf{t}, \mathbf{t} \in R^m\}. \end{aligned} \tag{5.17}$$

Since there is a diminishing-marginal-return effect between the hardware cost and the timing yield, it is reasonable to set a target timing yield below $\mathcal{Y}_*$. For the following discussions, the target timing yield is set as $\mathcal{Y}_t = \mathcal{Y}_0 + 0.9 \times (\mathcal{Y}_* - \mathcal{Y}_0)$.

5.5 Total Tunable Range Minimization

In this section, the PST-A problem is cast into a nonlinear optimization problem. A *simultaneous perturbation* (SP) [92, 93] algorithm is adopted to significantly reduce the time for gradient approximation of the timing yield function using only two Monte Carlo integrations. An iterative SP linesearch algorithm is proposed to solve PST-A efficiently.

5.5.1 Nonlinear Optimization Formulation

The PST-A problem is formulated as a nonlinear optimization problem with simple bound constraints as below:

$$\begin{aligned} (\mathcal{NLP}) \quad \max \quad & L_\gamma(\mathbf{r}) = \mathcal{Y}(\mathbf{r}) - \gamma \sum_{i=1\ldots m} r_i \\ s.t. \quad & r_i \geq 0, i = 1 \ldots m. \end{aligned} \tag{5.18}$$

By choosing a positive *penalty parameter* γ, the tunable ranges of the candidate buffers that do not contribute to the timing yield improvement are 'squeezed' toward zero. This formulation is similar to a typical penalty-function-based optimization that minimizes the total tunable range and a penalty term

on the timing yield violation. However, it will become clear that this formulation provides benefits on selecting γ and allows the optimization to be started from a feasible solution.

The $\mathcal{NLP}$ problem can be solved using linesearch algorithms. Linesearch algorithms require gradient information of the objective function. Since the analytic formula of $\mathcal{Y}(\mathbf{r})$ is not available, the gradient of $\mathcal{Y}(\mathbf{r})$ needs to be approximated using only $\mathcal{Y}(\mathbf{r})$ evaluations. A common gradient approximation method is *finite difference* (FD). A linesearch algorithm using one-sided finite difference approximation follows

$$\mathbf{r}_{k+1} = \mathbf{r}_k + c_k \hat{g}_\gamma(\mathbf{r}_k), \tag{5.19}$$

$$\hat{g}_\gamma(\mathbf{r}_k) = \begin{bmatrix} \frac{\mathcal{Y}(\mathbf{r}_k + b_k \mathbf{e^1}) - \mathcal{Y}(\mathbf{r}_k)}{b_k} - \gamma \\ \vdots \\ \frac{\mathcal{Y}(\mathbf{r}_k + b_k \mathbf{e^m}) - \mathcal{Y}(\mathbf{r}_k)}{b_k} - \gamma \end{bmatrix}, \tag{5.20}$$

where c_k is the step size and b_k is the perturbation size of iteration k. $\hat{g}_\gamma(\mathbf{r}_k)$ is the gradient approximation of $L_\gamma(\mathbf{r})$ at $\mathbf{r}_k$, and $\mathbf{e^i}$ is a unit vector with 1 on the i-th element. Therefore, in each step it takes m parameterized timing yield evaluations to obtain a gradient approximation. This is too computationally expensive.

5.5.2 Simultaneous Perturbation

Recent studies have shown that it is possible to use only two function evaluations to approximate the gradient by taking a random perturbation vector $\mathbf{\Delta}_k$ [92, 93]. The gradient approximation with SP is

$$\hat{g}_\gamma(\mathbf{r}_k) = \frac{\mathcal{Y}(\mathbf{r}_k + b_k \mathbf{\Delta}_k) - \mathcal{Y}(\mathbf{r}_k)}{b_k} \begin{bmatrix} \frac{1}{\mathbf{\Delta}_{k,1}} \\ \vdots \\ \frac{1}{\mathbf{\Delta}_{k,m}} \end{bmatrix} - \gamma \mathbf{1}. \tag{5.21}$$

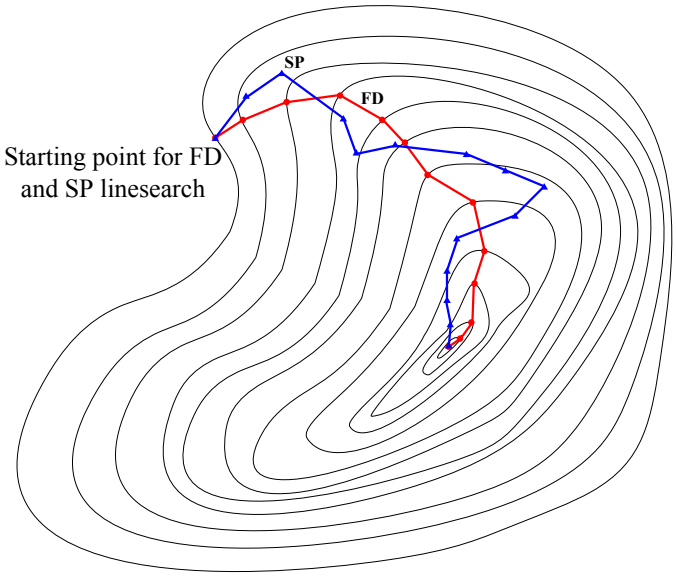


Figure 5.9: Illustration of the convergence of SP and FD linesearch (Source: Spall 1998).

There are a few conditions that must be met in order to guarantee the convergence of a linesearch algorithm using SP. The most important ones are that b_k and c_k need to go to 0 at appropriate rates and $\mathbf{\Delta}_{k,i}$ are independent and symmetrically distributed with $E[\mathbf{\Delta}_{k,i}] = 0$ and $E[|\mathbf{\Delta}_{k,i}|^{-1}] < \infty$. A common choice of the perturbation vector $\mathbf{\Delta}_k$ is the symmetric Bernoulli ± 1 distribution. It has been shown that under mild conditions, the number of measurements of $\mathcal{Y}(\mathbf{r})$ by a linesearch algorithm using SP can approach $\frac{1}{m}$ of that from a linesearch algorithm using FD while achieving the same asymptotic mean squared error of the solution. The intuition behind SP is that the gradient approximation in (5.21) is an *unbiased* approximation and it contains as much information as that from a finite difference approximation. Figure 5.9 illustrates the convergence of linesearch algorithms with SP and FD.

5.5.3 Iterative SP Linesearch

An iterative SP linesearch algorithm is proposed in Figure 5.10 to solve the PST-A problem. The algorithm starts from an initial solution $\mathbf{r}_{init}$, which has a sufficiently large parameterized timing yield ($\mathcal{Y}(\mathbf{r}_{init}) > \mathcal{Y}_t$). For example, a sufficiently large q can be selected such that $\mathbf{r}_{init} = q\mathbf{1}$ satisfies this condition. At the beginning of the optimization, the parameters for SP gradient approximation (b, η, c, C, π) are initialized according to the guidelines given in [93]. The convergence rate of the outer iterative loop (line 4-19) are set as $\chi = 2$, $\theta = 0.9$ and $\epsilon = 0.001$.

In the first iteration, the penalty parameter γ is chosen according to the equation in (line 3) to ensure that the timing yield after the first iteration is still larger than $\mathcal{Y}_t$. In the following iterations, the penalty parameter γ is gradually increased and the step and perturbation sizes are reduced (line 18). Within each iteration, an SP linesearch is used to find the optimal solution of the $\mathcal{NLP}$ problem (line 5-13). The latest intermediate solution that satisfies the target yield is recorded in $\tilde{\mathbf{r}}$ (line 14-16). There are two legalization steps in the algorithm (line 9, 11). The gradient approximation given by (5.21) can generate unrealistically large gradients due to a small perturbation step $\Delta_{k,i}$ in the denominator caused by the legalization step (line 9). This problem is resolved by choosing $\frac{1}{max(0.5,|\Delta_{k,i}|)}$ instead of $\frac{1}{\Delta_{k,i}}$ as the perturbation step size in (5.21). Truncation on the perturbation step can introduce noise to the gradient approximation. The noise has a greater impact to the convergence of the algorithm if it occurs at the beginning of the linesearch iteration when the step size is large. Therefore, choosing $\mathbf{r}_{init}$ instead of the origin as the starting point can reduce the noise effect.

The runtime of the algorithm is dominated by the number of $\mathcal{Y}(\mathbf{r})$ evaluations, which is the same as the number of SP linesearch steps (line 6-13). The iterative SP linesearch loop can be terminated early when a certain number of $\mathcal{Y}(\mathbf{r})$ evaluations is used. For large problems, the proposed iterative SP

Procedure: *IterativeSPLinesearch*

Input: $\mathcal{Y}(\mathbf{r}),\ \mathcal{Y}_t$, initial solution $\mathbf{r}_{init}$

Output: Final tunable range $\tilde{\mathbf{r}}$

Initialize $b, \eta, c, C, \pi, \epsilon, \chi, \theta$

$\tilde{\mathbf{r}} = \mathbf{r}_{prev} = \mathbf{r}_{init}$

$\gamma = \frac{\mathcal{Y}(\mathbf{r}_{init}) - \mathcal{Y}_t}{|\mathbf{r}_{init}|}$

repeat

`/* maximize` $L_\gamma(\mathbf{r})$ `using SP linesearch */`

$k = 1,\ \mathbf{r}_k = \mathbf{r}_{prev}$

repeat

$b_k = \frac{b}{k^\eta},\ c_k = \frac{c}{(C+k)^\pi}$

$\mathbf{\Delta}_k \leftarrow \pm 1$ symmetric Bernoulli random vector

$\mathbf{\Delta}_k = \max(\mathbf{\Delta}_k, \frac{-\mathbf{r}_k}{b_k})$ `// legalization`

Approximate $\hat{g}_\gamma(\mathbf{r}_k)$ by (5.21)

$\mathbf{r}_{k+1} = max(0, \mathbf{r}_k + c_k \hat{g}_\gamma(\mathbf{r}_k))$ `// legalization`

$k = k + 1$

until $|L_\gamma(\mathbf{r}_k) - L_\gamma(\mathbf{r}_{k-1})| < \epsilon$

if $\mathcal{Y}(\mathbf{r}_k) > \mathcal{Y}_t$ **then**

$\tilde{\mathbf{r}} = \mathbf{r}_k$

end

$\mathbf{r}_{prev} = \mathbf{r}_k$

`/* update step size (`b`,` c`) and penalty weight` γ `*/`

$\gamma = \chi\gamma, b = \theta b, c = \theta c$

until $\mathcal{Y}(\mathbf{r}_k) < \mathcal{Y}_t$

Figure 5.10: Algorithm for total tunable range minimization using iterative SP linesearch.

linesearch algorithm can take less than m steps to find a tunable range vector, less than the time for a traditional FD linesearch to take the very first step.

5.6 Reduction of PST Clock Buffers

In this section, a greedy algorithm for solving the PST-N problem is analyzed. A batch selection algorithm is proposed to speed up the optimization.

5.6.1 A Greedy Algorithm

In the PST-N problem, only the number of PST clock buffers used in a PST clock tree is of interest. A digital-to-analog converter controlled PST clock buffer with $\sim 700ps$ tunable range in a $0.18\mu m$ technology is reported in [64]. This tunable range is sufficient to counter process-induced path delay and clock skew variations. Therefore, a PST clock buffer is assumed to have an infinite tunable range in the PST-N problem. Under this assumption, a PST clock tree can be represented by a *selection vector* $\mathbf{r}_{sel}$, where the tunable range $r_{sel,i}$ is ∞ if buffer i is a PST clock buffer, and 0 if buffer i is a regular clock buffer. However, to find a selection vector with a minimum number of non-zero elements (PST clock buffers) that satisfies the target timing yield is a combinatorial optimization problem.

A common approach for combinatorial optimizations is a greedy method. Figure 5.11 shows a greedy algorithm for finding a selection vector. The algorithm starts with an empty selection vector and adds a PST clock buffer in each iteration until the target timing yield is achieved. In each iteration, it checks the potential timing yield improvement of every unselected buffer and chooses the one that gives the maximum improvement. The major issue of the greedy algorithm is that it requires $\frac{m+(m-M+1)}{2} \times M$ parameterized timing yield evaluations, where M is the number of non-zero elements in $\mathbf{r}_{sel}$. This is

Procedure: *Greedy*
Input: Timing yield model $\mathcal{Y}(\mathbf{r})$, target yield $\mathcal{Y}_t$
Output: Selection vector $\mathbf{r}_{sel}$

```
1  r = 0, Y_cur = Y_0
2  repeat
3      b = 0
4      for j = 1 ... m do
5          if r_j = 0 then
6              r_j = ∞
7              if Y(r) > Y_cur then
8                  Y_cur = Y(r), b = j
9              end
10             r_j = 0
11         end
12     end
13     r_b = ∞
14 until Y_cur > Y_t
15 r_sel = r
```

Figure 5.11: Algorithm for greedy selection of PST clock buffers.

unacceptable for large problems where m and M are both large. An algorithm that generates a good selection vector using $\sim m$ parameterized timing yield evaluations needs to be developed.

5.6.2 Batch Selection Algorithm

A batch selection algorithm is proposed in Figure 5.12 to overcome the runtime issue of the greedy algorithm. Instead of selecting one buffer at a time, all the unselected buffers are scanned and a buffer is selected immediately if it provides a timing yield improvement greater than a threshold value $\mathcal{Y}_{th}$ (line 6-7). The threshold value is decreased exponentially (line 13) and the selection only takes a few iterations to complete.

Procedure: *BatchSelection*
Input: Timing yield model $\mathcal{Y}(\mathbf{r})$, target yield $\mathcal{Y}_t$
Output: Selection vector $\mathbf{r}_{sel}$

1 $\mathbf{r} = 0,\ \mathcal{Y}_{cur} = \mathcal{Y}_0,\ \mathcal{Y}_{th} = 0.1 \times (\mathcal{Y}_* - \mathcal{Y}_0)$
2 **repeat**
3 **for** $j = 1 \ldots m$ **do**
4 **if** $r_j = 0$ **then**
5 $r_j = \infty$
6 **if** $\mathcal{Y}(\mathbf{r}) > \mathcal{Y}_{cur} + \mathcal{Y}_{th}$ **then**
7 $\mathcal{Y}_{cur} = \mathcal{Y}(\mathbf{r})$
8 **else**
9 $r_j = 0$
10 **end**
11 **end**
12 **end**
13 $\mathcal{Y}_{th} = 0.5 \times \mathcal{Y}_{th}$
14 **until** $\mathcal{Y}_{cur} > \mathcal{Y}_t$
15 $\mathbf{r}_{sel} = \mathbf{r}$

Figure 5.12: Algorithm for batch PST clock buffer selection.

The number of $\mathcal{Y}(\mathbf{r})$ evaluations required by the batch selection algorithm is ωm, where ω is the number of iterations (line 2-14) to achieve the target timing yield. Since the threshold for buffer selection $\mathcal{Y}_{th}$ decreases exponentially, ω is usually a small constant. Therefore, the overall runtime of the algorithm is $\frac{M}{\omega}$ times faster than the greedy algorithm.

5.7 Experimental Results

The algorithms are implemented in C++ and tested on a 1.7GHz Pentium-M computer. ISCAS89 benchmark circuits are synthesized and placed using SIS [84] and Dragon [85] to generate realistic flip-flop

placements. In practice, a timing-critical path is usually connected to other timing-critical paths and they usually form cycles [2]. The iterative clock scheduling algorithm presented in Chapter 3 is applied to identify timing-critical cycles. The clock period is chosen such that the most timing-critical path has a 3σ slack. The first $500 \sim 2000$ timing-critical paths are selected to build a parameterized timing yield model for each circuit as discussed in Section 5.4. In the parameterized timing yield model, all the slack variables are divided by their standard deviations. Therefore, the parameterized timing yield model is a unit-less model.

For each circuit, an H-tree is generated and a clock buffer is placed at every branching point and terminal of the H-tree. For S9234.1, an eight-level H-tree is generated (256 terminals). For the rest of the circuits, ten-level H-trees are generated (1024 terminals). Only PST clock buffers are shown in the subsequent figures.

For PST-A, the iterative SP linesearch algorithm (*IterSP*) is compared to a regular design method (*Regular*) that inserts identical PST clock buffers to all terminals of the clock tree. For PST-N, the batch selection algorithm (*Batch*) is compared to the greedy algorithm (*Greedy*) and a levelized design method (*Levelized*), which represents a current PST clock tree design strategy that inserts PST clock buffers in an entire level of the clock tree.

5.7.1 Nominal and Optimal Timing Yield

Table 5.1 shows the parameters of the timing yield models as well as the nominal and optimal timing yields of each circuit. The number of timing-critical paths included in the timing yield model is in the second column. The third column shows the number of clock tree terminals. The fourth column shows

[2] Otherwise, a timing-critical path can be eliminated by introducing useful-skew to its source or target flip-flop.

Circuit	# Paths(n)	# Terminal	# PST Candidate (m)	$\mathcal{Y}_0$(%)	$\mathcal{Y}_*$(%)	$\mathcal{Y}_t$(%)	$\mathcal{Y}_* - \mathcal{Y}_0$(%)
S9234.1	500	256	149(75)	92.15	97.48	96.94	5.33
S13207.1	500	1024	417(210)	94.56	99.44	98.95	4.88
S15850.1	1000	1024	641(321)	92.69	99.97	99.24	5.28
S35932	1000	1024	541(271)	54.93	99.99	95.49	45.06
S38584.1	2000	1024	1591(796)	86.69	99.44	98.17	12.75

Table 5.1: Timing yield models of the ISCAS89 benchmark circuits.

the number of candidate PST clock buffers selected by the buffer filtering algorithm presented in Section 5.4.1. The number in the parenthesis is the number of candidate PST clock buffers at the leaf level of an H-Tree. Each timing yield is obtained by Monte Carlo integration with 100,000 samples. The nominal, optimal and target timing yields of each circuit are listed in columns five through seven. As shown in the last column, post-silicon clock tuning provides significant timing yield improvements (5% $\sim$ 45%).

5.7.2 Total Tunable Range Improvement

Table 5.2 shows the comparison of the total tunable range achieved by *IterSP* and *Regular*. In *Regular*, it is assumed that identical PST clock buffers are placed only at the terminals of a clock tree. A binary search is used to find the minimum tunable range that achieves $\mathcal{Y}_t$ to within a $\pm 10^{-3}$ resolution. The tunable range is then multiplied by the number of leaf candidate PST clock buffers.

As shown in Table 5.2, *IterSP* achieves $> 65\%$ total tunable range reduction compared to *Regular*. One of the reasons for the significant improvement is that *IterSP* assigns a tunable range to a clock buffer no larger than what is required. This greatly reduces over design. The other contributor to the large improvement is that *IterSP* distributes the total tunable range among all candidate PST clock buffers located at different levels of the clock tree.

Circuit	*Regular*			*IterSP*					
	# Candidate	T.T.R. (σ)	CPU	# Candidate	T.T.R. (σ)	Reduction	Steps	# $\mathcal{Y}(\mathbf{t})$ eval.	CPU
S9234.1	75	24.94	1.9m	149	8.51	65.9%	52	208	1.1h
S13207.1	210	68.60	2.8m	417	11.33	83.5%	63	252	3.4h
S15850.1	321	104.85	7.5m	641	10.09	90.4%	107	428	6.5h
S35932	271	101.23	42.2m	541	1.90	97.4%	73	292	44.3h
S38584.1	796	238.26	36.7m	1591	17.19	92.8%	189	756	30.8h

Table 5.2: Comparison on total tunable range (T.T.R.) between a regular design method and the iterative SP linesearch algorithm.

Figure 5.13 shows the intermediate tunable range vector $\mathbf{r}_k$ of S9234.1 in the k-th SP linesearch step. As shown in the figure, the gradient approximations provided by SP efficiently guide the linesearch algorithm to reduce the total tunable range in only a small number of steps. In $\mathbf{r}_{52}$, the tunable ranges of about half of the 149 buffers are reduced almost to zero. This is because there are only 75 linearly independent tuning vectors among the 149 candidate buffers.

Averaging four gradient approximations for each SP linesearch step provides a better convergence rate without sacrificing too much runtime. On average, *IterSP* uses $\sim m$ parameterized timing yield estimations.

Although *IterSP* is able to obtain a good tunable range vector using only $\sim m$ parameterized timing yield estimations, it is still too slow for large problems such as S35932. *IterSP* has an extremely long runtime on S35932 because of its low nominal timing yield. As a result, $\sim 45K$ instances of a 1000×541 linear feasibility problem need to be solved in each parameterized timing yield evaluation. For large problems, it is necessary to select a small subset of candidate PST clock buffers and reduce the problem size before applying *IterSP*. The batch selection algorithm is a good candidate for this goal.

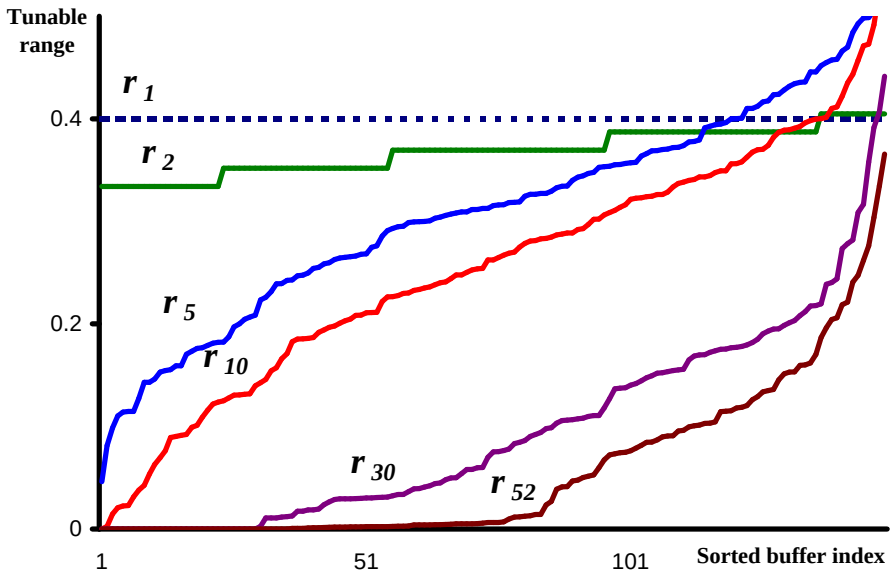


Figure 5.13: The tunable range vector of S9234.1 during iterative SP linesearch.

5.7.3 PST Clock Buffer Count Reduction

Table 5.3 shows the number of PST clock buffers and the runtime necessary to achieve the target timing yield by *Levelized*, *Greedy* and *Batch*. To reduce the number of required PST clock buffers, *Levelized* inserts PST clock buffers at a level as close to the root node of the clock tree as possible provided the target timing yield is satisfied.

The number of PST clock buffers required to achieve the target timing yield does not necessarily depend on the size of the circuit or the number of timing-critical paths included in the timing yield model. For example, S35932 only needs three PST clock buffers to achieve the target timing yield. Figure 5.14 shows the PST clock tree and the distribution of timing-critical paths of S35932. It is clear that either the source or the target flip-flops of most of the timing-critical paths are driven by one of the three clock buffers.

In general, *Levelized* uses PST clock buffers four times more than *Greedy* and *Batch* because *Lev-*

Circuit	# Candidate Buffers	# PST clock buffers					CPU Time		
		Levelized	*Greedy*	Reduction	*Batch*	Reduction	*Levelized*	*Greedy*	*Batch*
S9234.1	149(75)	32	8	75%	10	69%	1.7m	16.9m	4.6m
S13207.1	417(210)	256	16	94%	18	93%	1.9m	35.8m	6.8m
S15850.1	641(321)	128	17	87%	21	84%	13.2m	1.4h	21.6m
S35932	541(271)	16	3	81%	3	81%	1.5h	14.7h	3.1h
S38584.1	1591(796)	512	-	-	162	68%	14.3m	$>$ 2 day	8.9h

Table 5.3: Comparison on number of PST clock buffers and runtime among three design methods.

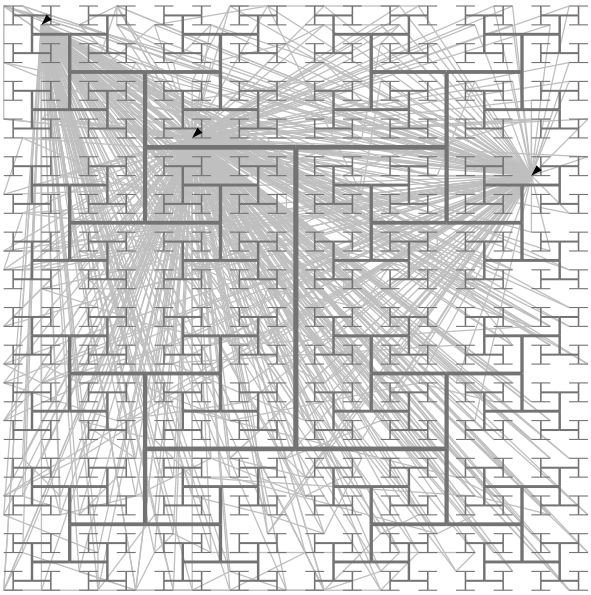
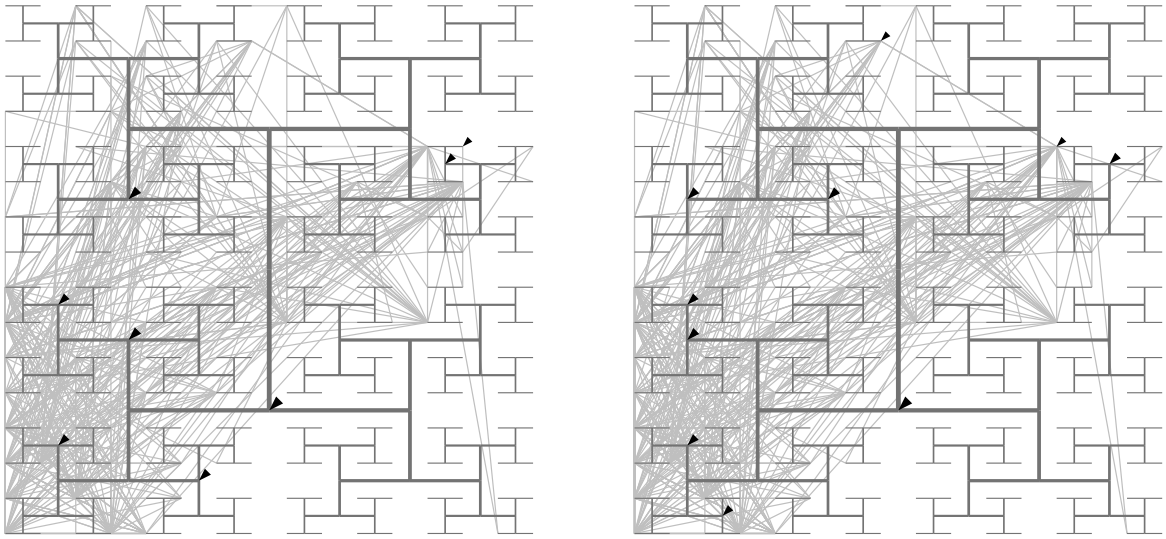


Figure 5.14: The PST clock tree of S35932. The triangles and light gray lines indicate PST clock buffer locations and timing-critical paths.



(a) Greedy (b) Batch

Figure 5.15: PST clock trees of S9234.1 generated by *Greedy* and *Batch*.

elized only inserts one level of PST clock buffers in a clock tree. On the contrary, *Greedy* and *Batch* can utilize *hierarchical tuning* to reduce the number of PST clock buffers. Figure 5.15 shows the PST clock trees of S9234.1 generated by *Greedy* and *Batch*. On the lower left corner of the clock trees, both algorithms generate PST clock trees with three levels of PST clock buffers. A PST clock buffer close to the clock root node can affect many timing paths simultaneously, while a PST clock buffer close to clock sink nodes can adjust the timing of specific timing paths. *By allowing multiple levels of PST clock buffers, the number of PST clock buffers can be reduced by exploring the correlation between timing-critical paths.*

The comparison of the number of PST clock buffers used by *Greedy* and *Batch* show that *Batch* has comparable solution quality to *Greedy*. Moreover, *Batch* provides $\sim$ 4X speedup on average. Therefore, *Batch* is a preferred algorithm for solving large PST-N problems.

Chapter 6

Future Works

This dissertation has provided a clock design flow that deals with timing uncertainty by enhancing existing clock design steps and using post-silicon clock tuning. Although new algorithms for reducing hardware cost of PST clock trees have been presented, further research on the following topics is still required to improve the maturity of post-silicon clock tuning.

6.1 PST Clock Tree Optimization

The PST clock tree synthesis method proposed in Chapter 5 constructs a PST clock tree by replacing regular clock buffers with PST clock buffers after clock tree optimization. This method requires engineering-change-orders (ECOs) to compensate extra insertion delay caused by PST clock buffers. It is desirable to insert PST clock buffers during the clock tree optimization step to avoid ECOs. To achieve this goal, a clock tree optimization algorithm needs to be able to determine when a PST clock buffer should be used during optimization. This requires a fast timing yield estimation method and a clock tree optimization algorithm that considers timing yield and hardware cost besides clock delay and skew.

6.2 Post-Silicon Clock Tuning through Selective Path Delay Testing

Currently, there are two approaches for post-silicon clock tuning. The first approach is to determine a delay configuration through silicon debugging. The delay configuration is then applied to all manufactured chips. This requires expensive hardware and software support as well as knowledge of the process. Moreover, this approach works well only when systematic variations predominate random variations. The second approach is to use genetic algorithms to automatically generate a good delay configuration [64, 65]. These algorithms first generate a set of random delay configurations. For each delay configuration, several sets of test vectors are applied to a chip and a fitness score is assigned based on how many tests the chip passes. This approach may require a long test application time to evolve a good delay configuration.

Post-silicon clock tuning is a clock scheduling process and a good delay configuration can be computed mathematically if path delays are known. Path delays in a manufactured chip can be measured indirectly by applying path delay testing vectors. Figure 6.1 demonstrates the principle for measuring path delays using path delay testing. Let $x_1 \sim x_3$ be the clock arrival times of $FF_1 \sim FF_3$ that can be adjusted by tuning PST clock buffers. Assume that the ranges of the clock arrival times are between one and eight. To measure the longest path delay from FF_2 to FF_3, one can configure the clock arrival times x_2 and x_3, apply test vectors to excite the longest path, scan out the result of FF_3 and compare it with the correct result. The pass and fail information contain the path delay information and can be used to derive the constraint between x_2 and x_3.

A chip may contain millions of FF pairs. Although paths having sufficient slack can be excluded from the test, a significant number of timing critical paths still need to be tested. Since each test requires scanning-in and scanning-out test vectors multiple times, the runtime for post-silicon clock tuning

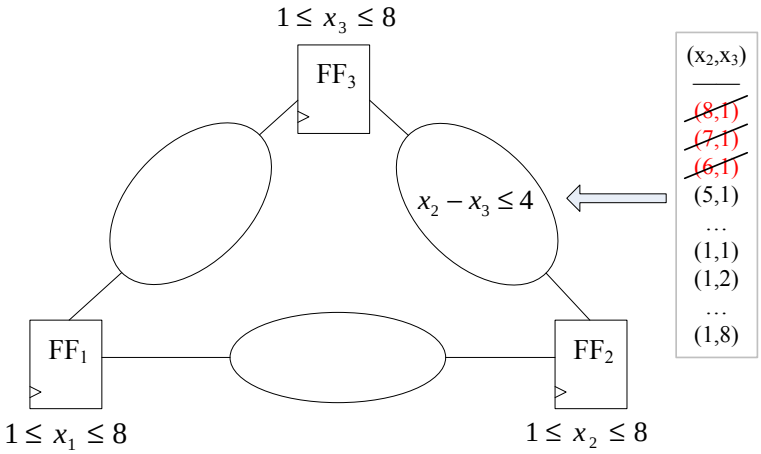


Figure 6.1: Measuring path delay by path delay testing.

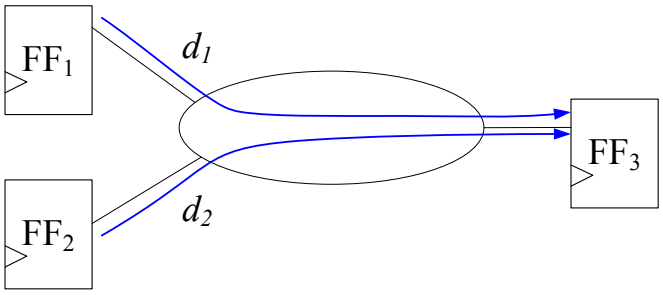


Figure 6.2: Reducing the number of test application by using correlation.

through path delay testing can be prohibitive. To address the runtime issue, one can utilize path delay correlation to reduce the number of test applications. As shown in Figure 6.2, there are two timing critical paths from FF_1 to FF_3 and from FF_2 to FF_3. Since the two paths share a long common path, d_2 can be estimated by d_1 with high confidence. Therefore, the post-silicon clock tuning problem becomes an optimization problem that maximizes the confidence of the delay map with minimum test application time.

Bibliography

[1] Jason Cong, Zhigang Pan, Lei He, Cheng-Kok Koh, and Kei-Yong Khoo. Interconnect design for deep submicron ICs. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 478–485, 1997.

[2] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[3] Qing-Tang Jiang, Ming-Hsing Tsai, and R.H. Havemann. Line width dependence of copper resistivity. In *Proceedings of the IEEE 2001 international interconnect technology conference*, pages 227–229, 2001.

[4] Semiconductor Industry Association. *International technology roadmap for semiconductors 2004 update*. World Wide Web, http://www.itrs.net/Common/2004Update/2004Update.htm, 2004.

[5] Semiconductor Industry Association. *International technology roadmap for semiconductors 2003 edition*. World Wide Web, http://public.itrs.net/Files/2003ITRS/Home2003.htm, 2003.

[6] Martin Eisele, Jörg Berthold, Doris Schmitt-Landsiedel, and Reinhard Mahnkopf. The impact of intra-die device parameter variations on path delays and on the design for yield of low voltage digital circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, 5(4):352–366, Dec. 1997.

[7] Madhav P. Desai, Radenko Cvijetic, and James Jensen. Sizing of clock distribution networks for high performance cpu chips. In *Proceedings of the 33rd annual conference on Design automation*, pages 389–394, 1996.

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

[9] Simon Tam, Stefan Rusu, Utpal Nagarji Desai, Robert Kim, Ji Zhang, and Ian Young. Clock generation and distribution for the first IA-64 microprocessor. *IEEE Journal of Solid-State Circuits*, 35(11):1545–1552, Nov 2000.

[10] Haihua Su and Sachin S. Sapatnekar. Hybrid structured clock network construction. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 333–336, 2001.

[11] Patrick Mahoney, Eric Fetzer, Bruce Doyle, and Sam Naffziger. Clock distribution on a dual-core multi-threaded Itanium-family processor. In *Digest of technical papers of the 2005 international solid-state circuits conference*, pages 292–293, 2005.

[12] R.-S. Tsay. Exact zero skew. In *Proceedings of the 1991 IEEE international conference on Computer-aided design*, pages 336–339, 1991.

[13] Ting-Hai Chao, Yu-Chin Hsu, Jan-Ming Ho, and A.B. Kahng. Zero skew clock routing with minimum wirelength. *Circuits and Systems II: Analog and Digital Signal Processing*, 39(11):799–814, Nov. 1992.

[14] Jason Cong, Andrew B. Kahng, Cheng-Kok Koh, and C.-W. Albert Tsao. Bounded-skew clock and steiner routing. *ACM Trans. Des. Autom. Electron. Syst.*, 3(3), 1998.

[15] Chung wen Albert Tsao and Cheng kok Koh. UST/DME: a clock tree router for general skew constraints. *ACM Trans. Des. Autom. Electron. Syst.*, 7(3):359–379, 2002.

[16] Shen Lin and C. K. Wong. Process-variation-tolerant clock skew minimization. In *Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design*, pages 284–288, 1994.

[17] L.P.P.P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 865–868, 1990.

[18] John Lillis, Chung-Kuan Cheng, and Ting-Ting Y. Lin. Optimal wire sizing and buffer insertion for low power and a generalized delay model. In *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 138–143, 1995.

[19] Takumi Okamoto and Jason Cong. Buffered steiner tree construction with wire sizing for interconnect layout optimization. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 44–49, 1996.

[20] Charles J. Alpert, Anirudh Devgan, and Stephen T. Quay. Buffer insertion with accurate gate and interconnect delay computation. In *Proceedings of the 36th annual conference on Design automation*, pages 479–484, 1999.

[21] Y.-P. Chen and D. F. Wong. An algorithm for zero-skew clock tree routing with buffer insertion. In *Proceedings of the European Design and Test Conference*, pages 230–236, 1996.

[22] A. Vittal and M. Marek-Sadowska. Power optimal buffered clock tree design. In *Proceedings of the Design automation conference*, pages 230–236, 1996.

[23] I-Min Liu, Tan-Li Chou, Adnan Aziz, and D. F. Wong. Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion. In *Proceedings of the international symposium on Physical design, 2000*, pages 33–38, 2000.

[24] Sachin S. Sapatnekar. RC interconnect optimization under the Elmore delay model. In *Proceedings of the 31st annual conference on Design automation*, pages 387–391, 1994.

[25] Rony Kay, Gennady Bucheuv, and Lawrence T. Pileggi. EWA: exact wiring-sizing algorithm. In *Proceedings of the 1997 international symposium on Physical design*, pages 178–185, 1997.

[26] Jason Cong, Cheng-Kok Koh, and Kwok-Shing Leung. Simultaneous buffer and wire sizing for performance and power optimization. In *Proceedings of the 1996 international symposium on Low power electronics and design*, pages 271–276, 1996.

[27] Chung-Ping Chen, Chris C. N. Chu, and D. F. Wong. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 617–624, 1998.

[28] Satyamurthy Pullela, Noel Menezes, and Lawrence T. Pillage. Reliable non-zero skew clock trees using wire width optimization. In *Proceedings of the 30th annual conference on Design automation*, pages 165–170, 1993.

[29] X. Zeng, D. Zhou, and Wei Li. Buffer insertion for clock delay and skew minimization. In *Proceedings of the 1999 international symposium on Physical design*, pages 36–41, 1999.

[30] Chung-Ping Chen, Yao-Wen Chang, and D. F. Wong. Fast performance-driven optimization for buffered clock trees based on lagrangian relaxation. In *Proceedings of the 33rd annual conference on Design automation*, pages 405–408. ACM Press, 1996.

[31] Charles E. Leiserson, Flavio M. Rose, and James B. Saxe. Optimizing synchronous circuitry by retiming. In *Proceedings of the 3rd CalTech conference on Very Large Scale Integration*, pages 87–116, March 1983.

[32] L. W. Cotten. Circuit implementation of high-speed pipeline systems. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 489–504, 1965.

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

[34] Rahul B. Deokar and Sachin S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *Proceedings of the 1994 IEEE International Symposium on Circuits and Systems*, volume 1, pages 407–410, May 1995.

[35] José Luis Neves and Eby G. Friedman. Optimal clock skew scheduling tolerant to process variations. In *Proceedings of the 33rd annual conference on Design automation*, pages 623–628, 1996.

[36] Ivan S. Kourtev and Eby G. Friedman. Clock skew scheduling for improved reliability via quadratic programming. In *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 239–243, 1999.

[37] Xun Liu, Marios C. Papaefthymiou, and Eby G. Friedman. Maximizing performance by retiming and clock skew scheduling. In *Proceedings of the 36th annual conference on Design automation*, pages 231–236, 1999.

[38] Baris Taskin and Ivan S. Kourtev. Performance optimization of single-phase level-sensitive circuits using time borrowing and non-zero clock skew. In *Proceedings of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems*, pages 111–118, 2002.

[39] C. Albrecht, B. Korte, J. Schietke, and J. Vygen. Cycle time and slack optimization for vlsi-chips. In *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 232–238, 1999.

[40] Stephan Held, Bernhard Korte, Jens Maberg, Matthias Ringe, and J. Vygen. Clock scheduling and clocktree construction for high performance asics. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, pages 232–239, 2003.

[41] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 6(1):87–90, 1958.

[42] L. R. Ford and D. R. Fulkerson. Flows in networks. *Princeton University Press*, 1962.

[43] N. E. Young, Robert E. Tarjan, and J. B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21:205–221, 1991.

[44] V. Hrapcenko. Depth and delay in a network. *Soviet Math. Dokl.*, 19(4):1006–1009, 1978.

[45] Karl Fuchs, Franz Fink, and Michael H. Schulz. DYNAMITE: An efficient automatic test pattern generation system for path delay faults. *Computer-Aided Design, IEEE Trans. on*, 10(10):1323–1335, Oct 1991.

[46] Kwang-Ting Cheng and H-C. Chen. Delay testing for non-robust untestable circuits. In *International Test Conference*, pages 954–961, Oct 1993.

[47] Seiji Kajihara, Kozo Kinoshita, Irith Pomeranz, and Sudhakar M. Reddy. A method for identifying robust dependent and functionally unsensitizable paths. In *10th International Conference on VLSI Design*, pages 82–87, Jan 1996.

[48] Jing-Jia Liou, Angela Krstic, Li-C. Wang, and Kwang-Ting Cheng. False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation. In *Proceedings of the 39th annual conference on Design automation*, pages 566–569, 2002.

[49] Yutaka Deguchi, Nagisa Ishiura, and Shuzo Yajima. Probabilistic CTSS: analysis of timing error probability in asynchronous logic circuits. In *Proceedings of the 28th conference on ACM/IEEE design automation conference*, pages 650–655, 1991.

[50] S. Devadas, H.F. Jyu, K. Keutzer, and S. Malik. Statistical timing analysis of combinational circuits. In *Proceedings of the 1992 ICCD conference*, pages 38–43, 1992.

[51] Masaki Uehata, Masakzu Tanaka, Masahiro Fukui, and Shuji Tsukiyama. False paths elimination in statistical static timng analysis. In *Proceedings of the 2002 international technical conference on Circuits/systems, computers and communications*, pages 357–360, 2002.

[52] Andrew Kahng, Jason Cong, and Gabriel Robins. High-performance clock routing based on recursive geometric matching. In *Proceedings of the 28th conference on ACM/IEEE design automation*, pages 322–327, 1991.

[53] Masato Edahiro. A clustering-based optimization algorithm in zero-skew routings. In *Proceedings of the 30th annual conference on Design automation*, pages 612–616, 1993.

[54] Nan-Chi Chou and Chung-Kuan Cheng. Wire length and delay minimization in general clock net routing. In *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 552–555, 1993.

[55] Gary Ellis, Lawrence T. Pileggi, and Rob A. Rutenbar. A hierarchical decomposition methodology for multistage clock circuits. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 266–273, 1997.

[56] Ying Liu, Sani R. Nassif, Lawrence T. Pileggi, and Andrzej J. Strojwas. Impact of interconnect variations on the clock skew of a gigahertz microprocessor. In *Proceedings of the 37th annual conference on Design automation*, pages 168–171, 2000.

[57] Anand Rajaram, Jiang Hu, and Rabi Mahapatra. Reducing clock skew variability via cross links. In *Proceedings of the 41st annual conference on Design automation*, pages 18–23, 2004.

[58] Dimitrios Velenis, Eby G. Friedman, and Marios C. Papaefthymiou. A clock tree topology extraction algorithm for improving the tolerance of clock distribution networks to delay uncertainty. In *Proceedings of the international symposium on Circuits and systems*, pages 4.422–4.425, 2001.

[59] Dimitrios Velenis, Marios C. Papaefthymiou, and Eby G. Friedman. Reduced delay uncertainty in high performance clock distribution networks. In *Proceedings of the conference on Design, Automation and Test in Europe*, page 10068, 2003.

[60] Michael Orshansky, Linda Milor, and Chenming Hu. Characterization of spatial intrafield gate cd variability, its impact on circuit performance, and spatial mask-level correction. *Semiconductor Manufacturing, IEEE Trans. on*, 17(1):2–11, Feb. 2004.

[61] Utpal Desai, Simon Tam, Robert Kim, Ji Zhang, and Stefan Rusu. Itanium processor clock design. In *Proceedings of the 2000 international symposium on Physical design*, pages 94–98, 2000.

[62] Jason Stinson and Stefan Rusu. A 1.5GHz third generation Itanium-2 processor. In *Proceedings of the 40th annual conference on Design automation*, pages 706–709, 2003.

[63] Ferd E. Anderson, J. Steve Wells, and Eugene Z. Berta. The core clock system on the next-generation Itanium microprocessor. In *Proceedings of the 2002 IEEE international conference on Solid-state circuits*, pages 108–424, 2002.

[64] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi. A post-silicon clock timing adjustment using genetic algorithms. In *Digest of technical papers of the 2003 symposium on VLSI circuits*, pages 13–16, 2003.

[65] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi. Post-fabrication clock-timing adjustment using genetic algorithms. *IEEE Journal of Solid-State Circuits*, 39(4):643–650, April 2004.

[66] Jeng-Liang Tsai, Tsung-Hao Chen, and Charlie Chung-Ping Chen. Epsilon-optimal minimum-delay/area zero-skew clock-tree wire-sizing in pseudo-polynomial time. In *Proceedings of the 2003 international symposium on Physical design*, pages 166–173, 2003.

[67] Jeng-Liang Tsai, Tsung-Hao Chen, and Charlie Chung-Ping Chen. Zero-skew clock-tree optimization with buffer-insertion/sizing and wire-sizing. *Computer-Aided Design, IEEE Trans. on*, Volumn 23, NO. 4:565–572, Apr. 2004.

[68] Jeng-Liang Tsai, DongHyun Baik, Charlie Chung-Ping Chen, and Kewal K. Saluja. A yield improvement methodology using pre- and post-silicon statistical clock scheduling. In *Proceedings of the 2004 IEEE/ACM international conference on Computer-aided design*, pages 611–618, 2004.

[69] Jeng-Liang Tsai, DongHyun Baik, Charlie Chung-Ping Chen, and Kewal K. Saluja. Yield-driven false-path-aware clock-skew scheduling. *IEEE Design & Test of Computers*, 22(3):214–222, May-June 2005.

[70] Jeng-Liang Tsai, DongHyun Baik, Charlie Chung-Ping Chen, and Kewal K. Saluja. False path and clock scheduling based yield-aware gate sizing. In *Proceedings of the 2005 International conference on VLSI Design*, pages 423–426, 2005.

[71] W.C. Elmore. The transient response of damped linear networks. *Journal of Applied Physics*, 19:55–63, Jan. 1948.

[72] J.M. Rabaey. *Digital Integrated Circuits*. Prentice Hall Publishers, 1996.

[73] Harry J.M. Veendrick. Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *IEEE Journal of Solid-State Circuits*, 19(4):468–473, Aug. 1984.

[74] José Luis Rosselló and Jaume Segura. Charge-based analytical model for the evaluation of power consumption in submicron CMOS buffers. *Computer-Aided Design, IEEE Trans. on*, 21(4):433–448, April 2002.

[75] Tanay Karnik, Shekhar Borkar, and Vivek De. Sub-90nm technologies: challenges and opportunities for cad. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 203–206, 2002.

[76] Shekhar Borkar. Circuit techniques for subthreshold leakage avoidance, control, and tolerance. In *Digest of technical papers of the 2004 international Electron Devices Meeting*, pages 421–424, 2004.

[77] Joe G. Xi and Wayne W.-M. Dai. Useful-skew clock routing with gate sizing for low power design. In *Proceedings of the 33rd annual conference on Design automation*, pages 383–388. ACM Press, 1996.

[78] A. B. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda. Path-based statistical timing analysis considering inter- and intra-die correlations. In *Proceedings of the 2002 TAU (ACM/IEEE workshop on timing issues in the specification and synthesis of digital systems)*, pages 16–21, 2002.

[79] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, N.J., 1963.

[80] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 296–305, 2001.

[81] Aseem Agarwal, David Blaauw, Vladimir Zolotov, and Sarma Vrudhula. Statistical timing analysis using bounds and selective enumeration. In *Proceedings of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems*, pages 29–36, 2002.

[82] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984.

[83] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[84] Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R. Stephan, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. *Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41*, 1992.

[85] Maogang Wang, Xiaojian Yang, and Majid Sarrafzadeh. Dragon2000: standard-cell placement tool for large industry circuits. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 260–263, 2000.

[86] Andrew E. Caldwell, Andrew B. Kahng, and Igor L. Markov. Improved algorithms for hypergraph bipartitioning. In *Proceedings of the 2000 conference on Asia South Pacific design automation*, pages 661–666, 2000.

[87] Hongliang Chang and Sachin S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 621, 2003.

[88] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Proceedings of the 41st annual conference on Design automation*, pages 331–336, 2004.

[89] Lizheng Zhang, Weijen Chen, Yuhen Hu, and Charlie Chung-Ping Chen. Statistical timing analysis with extended pseudo-canonical timing model. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 952–957, 2005.

[90] J. A. G. Jess, K. Kalafala, S. R. Naidu, R. H. J. M. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. In *Proceedings of the 40th annual conference on Design automation*, pages 932–937, 2003.

[91] Robin Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

[92] James C. Spall. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest*, 19(4):482–492, 1998.

[93] James C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronics Systems*, 34(3):817–823, July 1998.