Hindawi Publishing Corporation
VLSI Design
Volume 2011, Article ID 407507, 12 pages
doi:10.1155/2011/407507

Research Article

CONTANGO: Integrated Optimization of SoC Clock Networks

Dong-Jin Lee and Igor L. Markov

EECS Department, University of Michigan, 2260 Hayward Street, Ann Arbor, MI 48109-2121, USA

Correspondence should be addressed to Dong-Jin Lee, ejdjsy@umich.edu

Received 26 November 2010; Accepted 20 January 2011

Academic Editor: Rached Tourki

On-chip clock networks are remarkable in their impact on the performance and power of synchronous circuits, in their susceptibility to adverse effects of semiconductor technology scaling, as well as in their strong potential for improvement through better CAD algorithms and tools. Existing literature is rich in ideas and techniques but performs large-scale optimization using analytical models that lost accuracy at recent technology nodes and have rarely been validated by realistic SPICE simulations *on large industry designs*. Our work offers a methodology for SPICE-accurate optimization of clock networks, coordinated to satisfy slew constraints and achieve best tradeoffs between skew, insertion delay, power, as well as tolerance to variations. Our implementation, called Contango, is evaluated on 45 nm benchmarks from IBM Research and Texas Instruments with up to 50 K sinks. It outperforms all published results in terms of skew and shows superior scalability.

1. Introduction

Accurate distribution of clock signals is a major limiting factor for high-performance integrated circuits when unintended clock skew narrows down the useful portion of the clock cycle. Historically, clock skew became one of the first victims of semiconductor scaling, when wire delay started growing in significance relative to transistor delay. H-trees, popular in the industry, offered symmetric distribution networks that guaranteed nearly equal geometric lengths from the chip's center to individual clocked elements. However, H-trees did not immediately account for different sink capacitance and uneven distribution of sinks throughout the chip and did not minimize wire capacitance. The first geometric algorithms for clock routing evaluated skew in terms of wirelength from the source to sinks and produced minimum-wirelength trees for a given sink clustering (which is not difficult to optimize) using *the deferred merging and embedding* (DME) principle [1]. The tree structure facilitated powerful dynamic programming, and DME algorithms were extended to (1) handle skew in terms of Elmore delay, (2) balance uneven sink capacitance, and (3) minimize wire capacitance under nonzero skew bounds [2]. The DME family of algorithms were a major research achievement, both as mathematical insights and in terms of their computational performance. BST-DME algorithms [3] developed in the late 1990s reduced skew to single ps in fairly large circuits, while requiring only minutes of CPU time.

Semiconductor scaling in the 1990s made clock optimization more challenging. While transistors continued scaling, interconnect lagged in performance [4]. This phenomenon boosted demands for repeaters in clock networks, raised their power profile, and complicated their synthesis. Research in delay-driven buffering of single signal nets—arguably an easier problem and on a smaller scale—has blossomed well into the late 2000s, leaving clock-tree synthesis a difficult, high-value target. As the accuracy of compact delay models for transistors and wires deteriorated, clock-network design in the industry moved to SPICE-driven optimizations [5, 6].

Clock networks were among the first circuits to suffer the impact of process, voltage and temperature variations. Systematic variations can affect paths to different sinks in different ways, making effective skew higher than nominal skew. Intradie variations may be stronger on some paths than on others, which would further increase effective skew. These challenges have motivated research at the device, circuit,

and algorithm levels [7]. In general, smaller sink latencies and shorter tree paths decrease exposure to variations. Some researchers tried to increase the tolerance of buffers to CD changes and to temperature [8], some proposed to tune wires or buffers based on postsilicon measurements [9], and some developed methodologies for inserting cross-links into the trees [10–12], arguing that such links can decrease the impact of variation on skew. Existing literature tends to (1) rely on closed-form delay models during large-scale optimization, (2) frequently focus on a single optimization technique in analysis and evaluation, and (3) neglect the difficulties in modifying highly optimized clock trees. Our work seeks to address these omissions and develops a practical methodology for effective SPICE-accurate optimization, rather than elegant algorithms with provable abstract properties. With process variation in mind, microprocessor designers combine regular meshes with local or global trees [6]. However, meshes have much higher capacitance and use more power.

Our work focuses on clock-network synthesis for ASICs and SoCs, where clock frequencies are not as aggressive as in high-performance CPUs, but power is limited, especially for portable applications. In this context, tree topologies remain the most popular choice, potentially with further tuning and enhancements. The SoC context introduces another twist—layout obstacles. SoCs include numerous pre-designed blocks (CPUs, RAMs, DSPs, etc.) and datapaths. While it may be possible to route wires over such obstacles, buffer insertion is typically not allowed. One can fathom the difficulty of such optimization through comparison to signal-net routing, where obstacle-avoiding Steiner trees currently remain an active area of research [13]. Our contributions include the following:

(i) a careful analysis of design steps and optimizations for high-performance clock trees, including the range, accuracy, and substitutability of specific techniques,

(ii) notions of *slowdown and speedup slack* for clock trees,

(iii) tree optimizations driven by accurate delay models,

(iv) a simple and robust technique for obstacle avoidance in clock trees subject to slew constraints,

(v) a provably-good sink-polarity correction algorithm,

(vi) a methodology for clock-tree optimizations that outperforms the best results at the ISPD'09 contest *on every benchmark* by 2.15–3.99 times, while reducing skew to 2.2–4.6 ps (Table 5). It outperforms all published results in terms of skew (Table 6). On newer Texas Instruments benchmarks with up to 50 K sinks, skew remains <11 ps.

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

In the remainder of this paper, Section 2 reviews relevant previous work and the ISPD'09 CNS contest. Section 3 describes our analysis of the clock-network synthesis problem and introduces slowdown and speedup slacks. Major optimization steps are described in Sections 4 and 5 presents empirical results.

Table 1: The impact of skew bounds on *ispd09f22*.

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

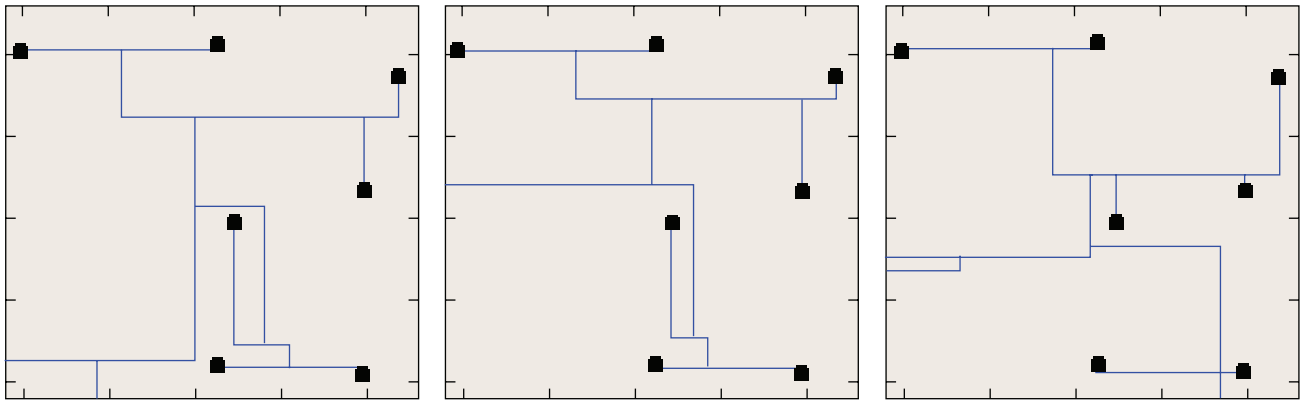
2. Background and Prior Work

DME Algorithms. Traditionally, clock trees have been constructed with respect to simple delay models—geometric pathlength or Elmore delay. In this context, the results in [1, 14–17] show how to build zero-skew trees (ZSTs) with minimal wirelength, improving upon H-trees and fishbones.

The deferred merge embedding (DME) algorithm, using the concept of *merging segment* [1, 14, 15] for constructing zero-skew tree, was extended to the bounded-skew tree (BST) problem. BST/DME algorithms [2, 3] generalize merging segments to merging regions. When BST/DME algorithms were introduced in the early 1990s, many chip designs included one large central buffer to drive clock signals through the entire chip. Today traditional clock trees cannot satisfy slew constraints in large ICs because the maximal length of unbuffered interconnect decreased significantly due to technology scaling [4]. Furthermore, the Elmore delay model used by published clock-tree optimizations lost accuracy due to resistive shielding and the impact of slew on delay.

BSTs allow one to trade off a small increase in skew for reduced total wirelength. Figure 1 shows that BSTs are shorter than ZSTs. However, BSTs are less balanced than ZSTs and Elmore delay used in BST generation is inaccurate, thus the capacitance saved on wires can be lost when compensating for skew with accurate timing analysis. After initial buffer insertion, slow sinks and fast sinks are more clustered in ZSTs. Since our skew optimization techniques exploit these clusters, BSTs need greater resources to reach near zero-skew than ZSTs. Table 1 shows the impact of BST skew bounds on final results (CLR is defined at the end of Section 2). The skew bounds during BST construction are based on Elmore delay, and the final results are based on SPICE simulations. Based on overwhelming empirical evidence against BSTs, Contango does not use them.

Obstacle-Avoiding Clock Trees. The concept of merging regions in BST/DME was extended to obstacle-avoiding trees in [18], where (i) obstacles were assumed rectangular, (ii) no routing over obstacles was allowed, and (iii) buffering was not considered. The authors noted that obstacle processing slowed down their BST/DME algorithm and hinted at more advanced geometric data structures. Unlike in [18], the ISPD'09 contest allowed *routing* but not *buffering* over



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

Figure 1: Min-wirelength trees with zero and bounded skew (Elmore delay). Only fragments of actual clock trees are shown.

obstacles, with SoCs in mind. ISPD'09 benchmarks included abutting obstacles that formed monolithic rectilinear obstacles.

Fast Buffer Insertion. L. van Ginneken introduced an algorithm for buffering RC-trees [19], which minimizes Elmore delay and runs in $O(n^2)$ time, given n possible buffer locations and buffer specification. While not intended for clock trees, it minimizes worst delay rather than skew. The $O(n \log n)$ time variant of van Ginneken's algorithms proposed in [20] is more appropriate for large trees. A key insight into van Ginneken's algorithms and its faster variant makes them applicable to our work—while trying to minimize source to sink latencies, these algorithms insert almost same number of buffers on every path and therefore result in low skew if the initial tree was already balanced.

Other buffering techniques have been proposed as well, for example, a linear-time algorithm from [21] that minimizes the number of buffers while bounding capacitive load and slew rate, but does not minimize delay or skew. A dynamic program from [22] inserts a limited number of buffers subject to a maximal skew in buffer counts on source-to-sink paths. At the ISPD'09 contest, slew constraints were checked by SPICE, but capacitance limits were relatively generous. Our competitors predominantly used greedy bottomup buffer-insertion algorithms that added each buffer as high in the tree as possible, while satisfying slew constraints. Such technique seek to minimize capacitance as the top priority. However, we chose the (faster variant of) van Ginneken's algorithm, which seeks to minimize worst sink latency. Our rationale was that process variations can be moderated by lowering sink latency and that it is relatively easy to slow down paths that are too fast, but it is harder to speed up slow paths. It is difficult to make a rigorous comparison with slew-based buffering [23]. In particular, some of our competitors at the ISPD 2009 contest relied on it and produced relatively poor results, but others did better. In any case, our overall results compare favorably to the best published results, especially in terms of nominal skew, and we were unable to improve them further by using slew-based buffering.

The ISPD'09 clock-network synthesis contest was organized by IBM Austin Research Laboratory and based on a 45 nm technology [24]. Sink latencies and clock skew were evaluated by SPICE. The main objective was the difference between the least sink latency @1.2 V (supply) and the greatest sink latency @1 V (supply). This *Clock Latency Range* (CLR) metric was intended to capture the impact of multiple power modes with different supply voltages [25], but nominal skew was also recorded. The 10%–90% slew rate of 100 ps and total power were strictly limited.

Several papers were published inspired by the ISPD'09 contest. Researchers from NTU proposed in [26] a dynamic nearest-neighbor algorithm (DNNA) to generate tree topology and a walk-segment breadth first search (WSBFS) for routing and buffering. To further refine the tree, they use dangling branches to adjust capacitance of wires (see our discussion in Section 4.7). Researchers from NCTU proposed in [27] a three-stage CLR-driven CTS flow based on an obstacle-avoiding balanced clock tree routing algorithm, monotonic parallel buffer insertion, as well as wire-sizing (BIWS) and wire-snaking. A dual-MST (DMST) geometric matching approach is proposed by researchers from HKPU in [28] for topology construction, along with recursive buffer insertion and a way to handle blockages. A timing-model independent buffered clock-tree synthesis is proposed in [29]. The authors proposed a branch-number plan, a cake-cutting partitioning and an embedding-region construction for nonbinary symmetrical buffered clock tree synthesis. They achieved low skew but do not explain how to generate obstacle-avoiding clock trees.

3. Problem Analysis

The design of a clock network offers a large amount of freedom in topology selection, spacing and sizing of inverters, as well as the sizing of individual wires. Traditionally, network topology is decided first. Trees offer unparalleled flexibility

in optimization because latency from the root to each sink can be tuned individually, while large groups of sinks can be tuned by altering nodes and edges high up in the tree.

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

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

To minimize the number of time-consuming SPICE invocations, we pursued several techniques. Runtime can be significantly reduced using *localization* and *batch-mode evaluation*. During localization, one prunes large portions of the clock tree that do not affect latencies to the sinks impacted by the changes in question [12]. This does not reduce the number of SPICE calls, but rather decreases the complexity of each run. On the other hand, a batch of changes can be evaluated by a single SPICE run, as long as multiple changes do not affect the same path from root to a sink.

Another avenue to streamlined SPICE-driven optimizations is to use mathematical properties of circuit delay, such as monotonicity, convexity, and linearity with respect to some parameters. Monotonicity and convexity support binary search, where an optimal value is sought on a certain interval. At each step of the search, the middle point of the interval is evaluated by SPICE (e.g., a wire can be sized half-way) and the result determines whether to recur to the left or right half-interval. Linearity enables extrapolation of multiple values based on several SPICE runs.

3.2. Nominal Skew Optimization. An initial buffered clock tree is constructed early in the design flow. Assuming no slew violations, the latency of each sink s (T_s) is known from SPICE simulations (or faster techniques, such as Arnoldi-based delay calculations), at which point minimal and maximal latencies ($T_{\max}$ and $T_{\min}$) can be found (separately for rising and falling transitions, for each PVT corner.) Since sink latencies are significantly larger than skew ($T_{\max} - T_{\min}$), skew can be improved by either decreasing $T_{\max}$ (speeding up the slowest sinks) or increasing $T_{\min}$ (slowing down the fastest sinks) without critical adverse effect on sink latencies.

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

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

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

Definition 2. Consider a clock tree and its edge e. The *slowdown slack* $\text{Slack}_e^{\text{slow}}$ (*speedup slack* $\text{Slack}_e^{\text{Fast}}$) of e is the amount in ps by which the edge delay can be unilaterally increased (decreased) without increasing clock skew.

Lemma 1. *For any edge e in the tree*

(i) $\text{Slack}_e^{\text{slow}} = \min_{s \in Sinks_e} \text{Slack}_s^{\text{slow}}$,

(ii) $\text{Slack}_e^{\text{Fast}} = \min_{s \in \text{Sinks}_e} \text{Slack}_s^{\text{Fast}}$.

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

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

The flexibility of a tree edge is limited by each downstream sink. Therefore, for edges close to the root we often have $\text{Slack}_e^{\text{slow}} = \text{Slack}_e^{\text{Fast}} = 0$. It is important to note that the validity of slacks-related calculations does not depend on the use of specific delay models or SPICE simulations. When visualizing clock trees, we color their edges with a red-green gradient, indicating low slack with red and high slack with green, as shown in Figure 4.

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

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

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

We found that after nominal skew is sufficiently optimized, both rising and falling transitions can individually limit speedup and slowdown slacks. We handle the two transitions separately and define edge slacks as the smaller of rise-slack and fall-slack. Furthermore, speedup and slowdown slacks can be computed for each process corner given (two in the ISPD'09 contest). In order to improve the multicorner CLR objective, a tree edge can be sped up conservatively by the minimum of its speedup slacks, and can be slowed down by the minimum of its slowdown slacks.

3.3. CLR Optimization. Our methodology pursues two objective functions—nominal skew and the ISPD09 CNS contest metric, CLR, introduced above. Due to significant correlation between CLR and nominal skew, some of the optimizations in our flow target skew optimization, some target CLR, and some address both (see Table 3). In practice this approach achieves a good tradeoff between the two optimization objectives, and is representative of multi-objective optimization required in many practical settings. Recall that the CLR calculation is based on the sink latencies at two different supply voltage settings. There are mainly two strategies to reduce CLR. First, reducing skew directly contributes to reducing CLR until skew becomes very small (e.g., less than 5 ps). Let sink L be the sink with the least sink latency @1.2 V ($T_L^{1.2\,\text{V}}$) and sink G be the sink with the greatest sink latency @1.0 V ($T_G^{1.0\,\text{V}}$). Then $CLR = T_G^{1.0\,\text{V}} - T_L^{1.2\,\text{V}}$. When we consider the latency of sink G @1.2 V ($T_G^{1.2\,\text{V}}$), then $\text{CLR} = (T_G^{1.0\,\text{V}} - T_G^{1.2\,\text{V}}) + (T_G^{1.2\,\text{V}} - T_L^{1.2\,\text{V}})$. We call $(T_G^{1.0\,\text{V}} - T_G^{1.2\,\text{V}})$ the variational part of CLR and $(T_G^{1.2\,\text{V}} - T_L^{1.2\,\text{V}})$ the skew part of CLR. The skew part of CLR can be reduced by skew optimization techniques. Since the corner sinks of skew are not always same to the corner sinks of CLR (sink L and G), CLR needs to be measured after any skew optimization to check CLR improvement. The second strategy for CLR optimization targets the variational component of CLR. The detailed descriptions of optimizations for the skew and variational part of CLR are discussed in Section 4.

3.4. Coordinating Multiple Optimizations. We found that different clock-tree optimizations exhibit different *strength/range* and different *accuracy* (see Tables 3 and 4).

Our strategy in coordinating clock-tree optimizations is to start with optimizations that offer the greatest range, and then transition to optimizations with greater accuracy. Each step should decrease the main optimization objective sufficiently to be within the range of the next optimization.

4. Proposed SoC Clock-Synthesis Methodology

Our proposed clock-network synthesis methodology and its major algorithmic steps are shown in Figure 2. Contango first builds an initial tree using a ZST/DME algorithm [3] and alters it to avoid obstacles. It then uses an $O(n \log n)$ time variant of van Ginneken's buffer insertion algorithm [20] to ensure small insertion delay and to satisfy slew constraints. A series of novel clock-tree optimizations are applied next.

4.1. Obstacle-Avoiding Clock Trees. As we pointed out in Section 2, obstacle-avoiding clock trees can be built by repairing obstacle violations in ZSTs. This approach is attractive when large obstacles abut the chip's periphery because ZSTs naturally avoid areas without clock sinks. This approach is also attractive when obstacles are small or thin enough that a buffer inserted immediately before the obstacle can drive the wire over the obstacle, so that no rerouting is necessary. A third convenient case occurs when a wire can be rerouted around the obstacle without an increase in length. Most obstacles are rectangular in shape, but such rectangles may abut, creating rectilinear-shaped obstacles. When two obstacles abut, we cannot place a buffer between them, and therefore handle them as one compound obstacle. Contango detours wires using the following algorithm, illustrated in Figure 3 for a composite obstacles.

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

Step 2. When a wire crosses an obstacle, Contango captures an entire subtree enclosed by the obstacle (see Figure 3). The total capacitance of the subtree is then measured and compared to the capacitance that can be driven by the driving buffer without risking slew violations. Subtrees that can be driven by the driving buffer do not require detours.

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

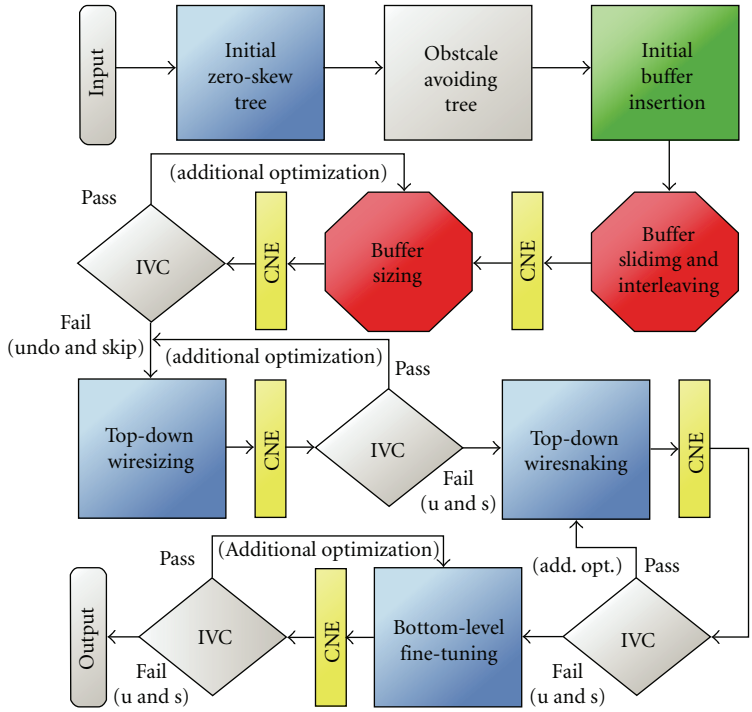


Figure 2: Key steps of the Contango methodology. Blue boxes represent skew reduction techniques, red octilinear shapes show CLR reductions, and the green box with thick border reduces both objectives. An Improvement- and violation-checking (IVC) step follows each clock-network evaluation (CNE) using circuit simulation tools, for example, SPICE. "Fail" indicates no improvement or having slew violations, leading to a transition to the next optimization.

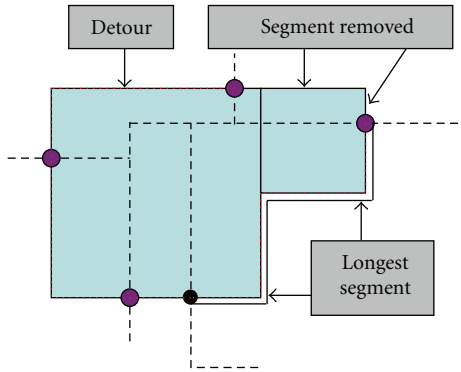


Figure 3: An illustration of our detouring algorithm. Small solid circle indicates the source of detour, and larger circles indicate sinks. The detour is shown with red dotted lines.

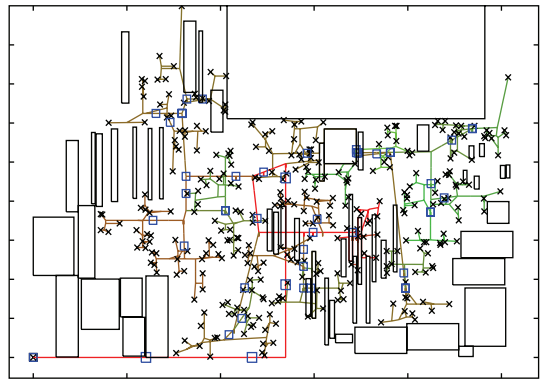


Figure 4: The clock tree produced by Contango on *ispd*09 *fnb*1. Sinks are indicated by crosses, buffers are indicated by blue rectangles. L-shapes are drawn as "diagonal wires" to reduce clutter. Wires are colored by a red-green gradient to reflect slowdown slacks, as described in Section 3.2. The impact of wiresnaking is too small to be visible.

shortest path to the source. The other segment incident to the sink is removed, but the shortest path from its other end to the source is included (see Figure 3).

Modern SoC layouts are littered with obstacles, which upset regular structures such as meshes and H-trees. In the ISPD 2009 contest, such layouts required numerous detours. Detouring may significantly increase skew, but the subsequent skew optimization techniques can compensate for that.

4.2. Composite Inverter/Buffer Analysis. Most technology libraries support dedicated clock buffers or inverters that are larger and more reliable than those for signal nets. Industry designs usually offer at least six different sizes. Parallel composition of buffers increases driver strength, helping with slew constraints and improving robustness to variations. Yet, buffer sizes must be moderated to satisfy total power limits. For a given buffer library, we consider many possible composite buffers. Using dynamic programming, we select several nondominated configurations that can be further evaluated during buffer insertion. Algorithmic details

Table 2: Inverter analysis for ISPD'09 CNS benchmarks.

Inverter type	Input	Output	
	Cap., fF	Cap., fF	Res., Ω
1X Large	35	80	61.2
1X Small	4.2	6.1	440
2X Small	8.4	12.2	220
4X Small	16.8	24.4	110
8X Small	**33.6**	**48.8**	**55**

are omitted here because the ISPD'09 contest used only two inverter types—*large* and *small*. Table 2 shows that eight parallel *small* inverters exhibit smaller output resistance than one *large* inverter, and smaller input/output capacitance. Hence, Contango used 8× small inverters instead of *large* inverters, in batches of 16×, 24×, and so forth. This benchmark-independent optimization, along with buffer sizing, plays an important role in our methodology.

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

We adapted the $O(n \log n)$ time variant of van Ginneken's algorithm from [20]. Due to its speed, it can be launched with different inverter configurations, effectively performing simultaneous optimization across multiple parameters. Our experiments indicate that driver strength is a major factor in moderating the impact of supply-voltage variations. Therefore, to reduce the variational part of CLR, $T_G^{1.0\,\mathrm{V}} - T_G^{1.2\,\mathrm{V}}$ (Section 3.3), Contango performs fast buffer insertion with different composite buffers until it finds the best-performing solution with strongest composite buffers within 90% of the power limit. Slew-constraint violations are not a concern at this point since minimizing delay involves avoiding high slew rate (recall that there is positive correlation between delay and slew rate). The experiments on various clock trees with initial buffer insertion suggest that even the worst slew rate is well under 60% of the slew limit. We reserve $\gamma = 10\%$ of power budget to facilitate more accurate optimizations.

The $O(n \log n)$ variant of van Ginneken's algorithm [20] used in our work assumes that all available clock buffers preserve polarity, therefore, the use of inverters typically leads to incorrect polarity at some sinks. The buffering algorithm can be extended to directly account for sink polarity, or it can be postprocessed by inserting additional inverters near sinks with incorrect polarity. To this end, we use the polarity-correction approach described in our conference paper [30]. In practice, it requires very few additional buffers, and its skew overhead is small enough to be compensated for by our downstream optimizations.

4.4. Buffer Sliding and Interleaving. We now discuss targeted improvement of robustness to variations in device performance. The iterative buffer sizing introduced in Section 4.5 is primarily used to reduce the variational component of CLR ($T_G^{1.0\,\mathrm{V}} - T_G^{1.2\,\mathrm{V}}$), while buffer sliding and interleaving are applied as preliminary steps. Extensive experiments suggest that the impact of variations on skew is best reduced by (i) decreasing sink latency (insertion delay), and (ii) using the strongest possible buffers. Since our initial buffer insertion algorithm focuses on the former metric with the latter metric as a secondary objective, it is possible to further improve the variational component of CLR ($T_G^{1.0\,\mathrm{V}} - T_G^{1.2\,\mathrm{V}}$) by emphasizing the latter metric. Therefore, based on the results of initial buffer insertion, Contango attempts to size buffers up.

Sizing up a single inverter increases its input pin capacitance and can lead to slew violations. To prevent such violations, it is often possible to slide the inverter up the tree to reduce upstream wire capacitance and interleave an inverter when two inverters move too far apart after sliding. The increase in downstream wire capacitance is balanced with the increase in the inverter's driving strength. Sizing a single inverter may increase the skew and require further correction. Therefore, we focused on the top-most levels of the tree, whose impact on skew is relatively small. Given a clock source at the chip boundary, DME algorithms generate a long wire leading to the center of the chip, and the tree branches out from the center. This long wire— the *tree trunk*—is later populated with a chain of inverters, which can be up- or downsized without significant impact on skew because this equally affects all sinks. However, since roughly 1/3 to 1/2 of sink latency is due to the tree trunk, it accounts for a large fraction of variational impact on latency.

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

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

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

Table 3: Progress achieved by individual steps of Contango on ISPD'09 benchmarks: the first letter in each acronym indicates top-down (T) or bottom-level (B) optimization, second letter differentiates wires (W) from buffers (B), while "Sz" stands for "sizing" and "Sn" stands for "snaking". Italic numbers indicate whether skew or CLR was the primary optimization objective.

	ISPD09F11		ISPD09F12		ISPD09F21		ISPD09F22		ISPD09F31		ISPD09F32		ISPD09FNB1	
	CLR	Skew	CLR	Skew	CLR	Skew	CLR	Skew	CLR	Skew	CLR	Skew	CLR	Skew
Initial	56.18	30.58	75.81	48.96	89.29	59.17	52.01	31.55	151.8	116.5	121.6	88.19	31.86	21.15
TBSz	*55.61*	46.78	*80.03*	66.24	*89.49*	76.31	*43.16*	33.65	*140.3*	129.2	*110.7*	98.27	*31.54*	21.13
TWSz	23.38	*15.07*	19.70	*8.127*	26.00	*12.25*	16.35	*6.933*	43.08	*32.21*	27.23	*14.84*	30.75	*20.44*
TWSn	13.75	*2.929*	16.21	*3.384*	17.60	*2.826*	12.58	*1.99*	*12.81*	*3.91*	*17.92*	*4.594*	*13.94*	*3.149*
BWSn	*13.36*	*2.867*	*15.27*	*2.611*	*17.40*	*2.738*	*12.36*	*2.227*	*12.81*	*3.91*	*17.92*	*4.594*	*13.40*	*3.5*

Table 4: The "Full flow" column shows *skew change* at each step in the Contango flow, and the final skew. Acronyms are decoded in the caption of Table 3. Subsequent columns show the impact of removing one optimization. These results illustrate the *range* of each optimization and its *impact on final results*.

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

4.6. Iterative Top-Down Wiresizing. Before skew optimization, Contango computes slowdown slacks at every edge as described in Section 3, and the Δ_e^{slow} parameters. This suggests the amount by which a given tree edge can be slowed down before skew would be negatively affected. Since fast sinks often cluster together, skew can be lowered by slowing down either many bottom-level wires or few wires higher in the tree. Our top-down algorithm pursues the latter, seeking to minimize tree modifications.

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

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

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

Algorithm 1: Iterative wiresizing.

4.7. Iterative Top-Down Wiresnaking. Wiresizing can reduce large skew by applying small changes, which is appropriate after the initial tree construction. An experienced clock-network designer suggested to us that a small amount of wire-snaking is often used to improve clock skew, as long as added capacitance does not significantly affect power. Wiresnaking alters a given route so as to increase its length and can be applied on fast paths.

We develop an accurate top-down wiresnaking process, which we invoke *after* top-down wiresizing. This step uses the same slowdown slack computation we described earlier. A SPICE simulation is performed (other accurate delay model can be used) to measure T_{wn}, the worst-case delay of wiresnaking with unit length l_{wn}. l_{wn} affects the accuracy of the wiresnaking algorithm; smaller l_{wn} offers greater accuracy

Table 5: Results on the ISPD'09 Contest benchmark suite. CLR is reported in ps, capacitance in % of the limit specified in benchmarks, and CPU time in *s*. Best results from the ISPD'09 contest and best results overall are shown in bold. Runtime is dominated by SPICE runs. It was not used for scoring at the ISPD'09 contest and can be improved by using FastSPICE, Arnoldi approximation, and so forth.

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

Table 6: Results from ASPDAC'10 clock routing papers on the ISPD'09 Contest benchmark suite [26–28]. Runtimes may be from different workstations. CLR and skew are reported in ps and CPU time in *s*. Only average skew was published for HKPU [28].

Benchmark	Contango (this work)			NTU [26]			NCTU [27]			HKPU [28]		
	CLR	Skew	🕒	CLR	Skew	🕒	CLR	Skew		CLR	Skew	🕒
ispd09f11	13.36	2.867	6488	19.71	4.478	4639	18.77	7.12	30787	12.2	—	180
ispd09f12	15.27	2.611	6564	17.46	4.088	4231	15.5	3.06	27622	10.9	—	213
ispd09f21	17.40	2.738	6673	19.92	3.868	4629	17.04	3.02	33056	12.1	—	210
ispd09f22	12.36	2.227	3618	16.47	3.671	3937	16.25	4.11	19136	9.9	—	113
ispd09f31	12.81	3.91	21379	31.13	4.762	11112	22.63	7.58	66588	13.4	—	777
ispd09f32	17.92	4.594	12895	23.04	4.234	7293	20.59	5.52	49907	11.5	—	420
ispd09fnb1	13.40	3.5	778	15.73	6.798	3719	14.32	3.77	7643	13.8	—	82
Average	**14.65**	**3.207**	**8342**	**20.49**	**4.56**	**5651**	**17.87**	**4.88**	**33534**	**11.97**	**7.72**	**285**

Table 7: Scalability on Texas Instruments benchmarks. The "Latency" column represents maximum 1.2 V latencies. SPICE runs are counted in parenthesis.

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

but typically leads to more SPICE runs since skew reduction in each round of top-down wiresnaking is smaller. l_{wn} was set based on empirical analysis of the 45 nm technology used at the ISPD contest before contest benchmarks became available. The applicability of wiresnaking depends on the VLSI context. If the clock tree is competing for routing resources with signal nets, then every effort should be taken to reduce the utilization of routing resources. In particular, wiresnaking cannot be used in areas of routing congestion (also, clock trees should avoid such areas to minimize crosstalk noise). On the other hand, some ICs include abundant routing resources. This is the case for pad-limited designs and designs whose area is determined by large IP blocks. The number of available metal layers also plays a major role in the design of clock trees, and can vary dramatically between different designs, ranging from 6 to 12 layers as of 2010. In some high-performance designs, clock networks are given a dedicated metal layer, which makes wiresnaking much more attractive.

One of the top-three teams at the ISPD 2009 clock-tree routing contest (NTU [26]) used *dangling wires* instead of wiresnaking. Rather than elongate a route, this strategy adds a dead-end branch. The goal is to increase wire capacitance, and, therefore, increase the delay. In comparing dangling wires to wire-snaking, we note that the former does not alter the resistance that affects propagation delay. Therefore, to achieve a particular slowdown, a much longer wire branch is needed. On the positive side, the dependence of delay increase on branch length is linear, and this may allow for more accurate tuning. In other words, this technique offers a potentially *greater accuracy*, but *smaller range* because the range of such optimizations is limited by the capacitance budget. Therefore, if dangling wires are found useful, they should be used at a later stage in the optimization flow.

4.8. Bottom-Level Fine-Tuning and Limits to Further Optimization. After two top-down skew reduction phases, skew becomes small enough to perform bottom level optimizations. Bottom-level wiresnaking optimize the wires directly connected to sinks. This technique is more accurate than the top-down optimizations since each sink is tuned individually. Contango performs SPICE-driven bottom-level wiresnaking until the results stop improving. Typically the gain of bottom-level tuning is under 2 ps, but can be a significant fraction of remaining skew.

We found that with skew <5 ps, the corner sinks of rising transition and falling transition are often different.

This *rise-fall divergence* makes further improvements to the clock tree very difficult. Indeed, reducing *rising skew* by slowing down a *fast sink for rising transition* may increase *falling skew* due to excessive slowdown of a *slow sink for falling transition*. In the Contango flow, the average skew after bottom-level tuning is 3.21 ps on ISPD'09 CNS contest benchmarks.

Table 3 shows the improvement of CLR and skew by each optimization algorithm. Note that after iterative buffer sizing (TBSz), skew is increased but CLR does not change much. This implies that TBSz reduced the variational part of CLR ($T_G^{1.0\,\mathrm{V}} - T_G^{1.2\,\mathrm{V}}$) significantly. TBSz is performed before skew optimization, because it increases the skew part of CLR ($T_G^{1.2\,\mathrm{V}} - T_L^{1.2\,\mathrm{V}}$). The increased skew is reduced below 5 ps after our skew optimizations.

5. Empirical Validation

To validate our proposed techniques, we first present results on ISPD'09 benchmarks with detail comparison to state-of-the-art academic clock network synthesis tools according to the contest protocol, then discuss the significance of specific optimizations used by Contango, and then evaluate the scalability of our C++ implementation on larger benchmarks from our industry colleagues. We measured runtimes on a 2.4 GHz Intel QuadCore CPU running Linux, similar to CPUs used at the ISPD contest.

ISPD'09 benchmarks include seven 45 nm chips up to 17 mm × 17 mm in size, with up to 330 selected clock sinks [24]. Table 5 compares results of our software Contango to the top three teams of the ISPD'09 clock-network synthesis contest. On average, Contango reduces CLR by **2.15**×, **3.99**× and **2.35**× versus contest results by NTU, NCTU and U. of Michigan respectively, excluding failures of NTU and NCTU on benchmarks with many obstacles. All results are within the capacitance limits, but Contango nearly exhausts the limits as a part of its strategy. On ISPD'09 benchmarks, maximum sink latency averages 1120*ps*, while the average number of composite-buffer locations is 223. A clock tree built by Contango is shown in Figure 4.

More recent results for ISPD'09 benchmarks from ASP-DAC'10 [26–28] are summarized in Table 6. a Dynamic Nearest-Neighbor Algorithm (DNNA) for topology construction, along with the results in Table 6 show that Contango outperforms NTU and NCTU by skew and CLR. HKPU [28] claims a 20% advantage in CLR, but more than doubles nominal skew. Another interesting aspect of the HKPU work is that they rely on SPICE very little in their optimizations and instead use the Elmore delay model, which explains their low runtimes. The algorithms in [28] focus entirely on the optimization of nominal skew, which does not explain the results—high nominal skew and low CLR. As the authors of [28] have kindly provided their clock trees on our request, we observed that those trees use very large buffers at the top levels of the tree (including but not limited to the trunk) and small buffers toward the sinks. This strategy minimizes the impact of *supply voltage* variations, but makes it more difficult to optimize nominal skew given a limited capacitance budget.

Significance of Individual Optimization. Several optimizations we have implemented were superseded by more powerful techniques. For example, *skew reduction by buffer insertion* was unnecessary and undermined the robustness to variations. However, it can be used as a last resort when detours around obstacles introduce extremely high skew. Our wiresizing can be refined but probably not beyond the accuracy of subsequent wiresnaking. In practice, wiresnaking is very limited, so as to preserve the routability of signal wires (unless clock wiring is given a dedicated metal layer). Dangling wires, used by NTU instead of wire snaking, would be even less acceptable.

To further study the relative significance of optimizations in Contango, we show in Table 4 the impact of removing each skew optimization step from the flow. It can be seen that each step is necessary to achieve competitive results. Removing top-down wiresizing effects the greatest impact because this optimization offers the greatest range, and subsequent optimizations cannot fully compensate for its omission.

Scalability Studies. The ISPD'09 contest was limited to unrealistically small numbers of sinks due to limitations of the open-source ngSPICE software [31] it relied upon. To evaluate the scalability of our optimizations, we replaced ngSPICE with industry-standard HSPICE software [32].(The numbers produced by ngSPICE and HSPICE were fairly close, with the main difference being runtime and scalability.) Working with a recent Texas Instruments chip sized 4.2 mm × 3.0 mm, we identified locations of 135 K sinks and randomly sampled them to create a family of benchmarks. For this experiment, our algorithm used groups of large inverters instead of groups of 8 parallel small inverters, improving runtime eightfold at the cost of increasing CLR and skew by 1-2 ps and increasing capacitance by 15%. It produced highly optimized clock trees with up to 50 K sinks. Table 7 shows that total capacitance scales linearly with the number of sinks, and skew remains in single ps. The number of HSPICE runs grows very slowly, but HSPICE remains the bottleneck.

6. Conclusions

Existing literature on clock networks offers several elegant algorithms but does not describe end-to-end solutions to

clock-network synthesis that can handle modern interconnect. Our work makes several contributions to this end. First, we develop specialized optimization algorithms necessary to bridge the gaps between well-known point-optimizations. Our emphasis is on robust techniques, that do not require tuning and are amenable to embedding into design flows. Second, we develop an EDA methodology for integrating clock-network optimization steps. Third, we describe a robust software implementation, called Contango, that outperforms best results from the ISPD'09 contest [24] by a factor of two.(The use of two wire sizes, two inverter types, and two process corners in the ISPD'09 contest is not a limitation of our algorithms and methodology. Likewise, any accurate delay evaluator can be used, including FastSpice, and Arnoldi approximations.) Fourth, we scale our implementation to large industrial clock networks.

Based on their strong empirical results, our techniques may improve timing and power of future ASICs and SoCs [5]. In CPU designs, our trees can be integrated with meshes [6]. Here, better trees may facilitate smaller meshes and reduce power consumption, which can be traded off for higher performance or longer battery life in portable applications.

References

[1] Chao, Hsu, Ho, Kenneth D. Boese, and Andrew B. Kahng, "Zero skew clock routing with minimum wirelength," in *Proceedings of the International Conference on Coffee Science (ASIC '92)*, vol. 39, no. 11, pp. 17–21, November 1992.

[2] D. J. H. Huang, A. B. Kahng, and C. W. A. Tsao, "On the bounded-skew clock and Steiner routing problems," in *Proceedings of the 32nd Design Automation Conference (DAC '95)*, pp. 508–513, June 1995.

[3] J. Cong, A. B. Kahng, C. K. Koh, and C. W. Albert Tsao, "Bounded-skew clock and steiner routing," *ACM Transactions on Design Automation of Electronic Systems*, vol. 3, no. 3, pp. 341–388, 1998.

[4] R. Ho, K. Mai, and M. Horowitz, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, 2001.

[5] P.-H. Ho, "Industrial clock design," in *Proceedings of the International Symposium on Physical Design (ISPD '09)*, pp. 139–140, December 2009.

[6] R. S. Shelar, "An algorithm for routing with capacitance/distance constraints for clock distribution in microprocessors," in *Proceedings of the International Symposium on Physical Design (ISPD '09)*, pp. 141–148, April 2009.

[7] A. B. Kahng et al., "Interconnect tuning strategies for high-performance ICs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '98)*, pp. 471–478, 1998.

[8] F. Huebbers, A. Dasdan, and Y. Ismail, "Multi-layer interconnect performance corners for variation-aware timing analysis," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 713–718, November 2007.

[9] V. Khandelwal and A. Srivastava, "Variability-driven formulation for simultaneous gate sizing and postsilicon tunability allocation," *IEEE Transactions on Computer-Aided Design*, vol. 27, no. 4, pp. 610–620, 2008.

[10] S. Hu, Q. Li, J. Hu, and P. Li, "Utilizing redundancy for timing critical interconnect," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 10, pp. 1067–1080, 2007.

[11] W. C. D. Lam, J. Jain, C. K. Koh, V. Balakrishnan, and Y. Chen, "Statistical based link insertion for robust clock network design," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '05)*, pp. 588–591, November 2005.

[12] J. Hu, A. B. Kahng, B. Liu, G. Venkataraman, and X. Xu, "A global minimum clock distribution network augmentation algorithm for guaranteed clock skew yield," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '07)*, pp. 25–31, 2007.

[13] J. Long, H. Zhou, and S. O. Memik, "An O(nlogn) edge-based algorithm for obstacle-avoiding rectilinear Steiner tree construction," in *Proceedings of the ACM International Symposium on Physical Design (ISPD '08)*, pp. 126–133, April 2008.

[14] M. Edahiro, "Clustering-based optimization algorithm in zero-skew routings," in *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pp. 612–616, June 1993.

[15] T.-H. Chao, Y.-C. Hsu, and J.-M. Ho, "Zero skew clock net routing," in *Proceedings of the Design Automation Conference (DAC '92)*, pp. 518–523, 1992.

[16] R. S. Tsay, "Exact zero-skew clock routing algorithm," *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 2, pp. 242–249, 1993.

[17] T. H. Chao, YU. C. Hsu, J. M. Ho, K. D. Boese, and A. B. Kahng, "Zero skew clock routing with minimum wirelength," *IEEE Transactions on Circuits and Systems II*, vol. 39, no. 11, pp. 799–814, 1992.

[18] A. B. Kahng and C. W. A. Tsao, "Practical bounded-skew clock routing," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 16, no. 2-3, pp. 199–215, 1997.

[19] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proceedings of the International Symposium on Computer Architecture (ISCA '90)*, pp. 865–868, May 1990.

[20] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *IEEE Transactions on Computer-Aided Design*, vol. 24, no. 6, pp. 879–891, 2005.

[21] C. J. Alpert, A. B. Kahng, B. Liu, I. I. Măndoiu, and A. Z. Zelikovsky, "Minimum buffered routing with bounded capacitive load for slew rate and reliability control," *IEEE Transactions on Computer-Aided Design*, vol. 22, no. 3, pp. 241–253, 2003.

[22] C. Albrecht, A. B. Kahng, B. Liu, I. I. Măndoiu, and A. Z. Zelikovsky, "On the skew-bounded minimum-buffer routing tree problem," *IEEE Transactions on Computer-Aided Design*, vol. 22, no. 7, pp. 937–945, 2003.

[23] S. Hu, C. J. Alpert, J. Hu et al., "Fast algorithms for slew-constrained minimum cost buffering," *IEEE Transactions on Computer-Aided Design*, vol. 26, no. 11, pp. 2009–2022, 2007.

[24] C. N. Sze, P. Restle, GI. J. Nam, and C. Alpert, "ISPD2009 clock network synthesis contest," in *Proceedings of the International Symposium on Physical Design (ISPD '09)*, pp. 149–150, April 2009.

[25] C. L. Lung, ZI. Y. Zeng, C. H. Chou, and S. C. Chang, "Clock skew optimization considering complicated power modes," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '10)*, pp. 1474–1479, March 2010.

[26] X. W. Shih, C. C. Cheng, Y. K. Ho, and Y. W. Chang, "Blockage-avoiding buffered clock-tree synthesis for clock latency-range and skew minimization," in *Proceedings of the 15th Asia and South Pacific Design Automation Conference (ASP-DAC '10)*, pp. 395–400, January 2010.

[27] W. H. Liu, Y. L. Li, and H. C. Chen, "Minimizing clock latency range in robust clock tree synthesis," in *Proceedings of the 15th Asia and South Pacific Design Automation Conference (ASP-DAC '10)*, pp. 389–394, January 2010.

[28] J. Lu, W. K. Chow, C. W. Sham, and E. F. Y. Young, "A dual-MST approach for clock network synthesis," in *Proceedings of the 15th Asia and South Pacific Design Automation Conference (ASP-DAC '10)*, pp. 467–473, January 2010.

[29] X.-W. Shih and Y.-W. Chang, "Fast timing-model independent buffered clock-tree synthesis," in *Proceedings of the Design Automation Conference (DAC '10)*, pp. 80–85, 2010.

[30] D. Lee and I. L. Markov, "Contango: integrated optimization of SoC clock networks," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '10)*, pp. 1468–1473, March 2010.

[31] P. Nenzi, "NG-SPICE: the free circuit simulator," 2010, http://ngspice.sourceforge.net.

[32] HSPICE, *Simulation and Analysis User Guide*, Synopsys, Mountain View, Calif, USA, 2003.