Low-power Clock Trees for CPUs

Dong-Jin Lee, Myung-Chul Kim and Igor L. Markov
EECS Department, University of Michigan, Ann Arbor, MI
University of Michigan, 2260 Hayward St., Ann Arbor, MI 48109-2121
{ejdjsy, mckima, imarkov}@eecs.umich.edu

***Abstract*—Clock networks contribute a significant fraction of dynamic power and can be a limiting factor in high-performance CPUs and SoCs. The need for multi-objective optimization over a large parameter space and the increasing impact of process variation make clock network synthesis particularly challenging. In this work, we develop new modeling techniques and algorithms, as well as a methodology, for clock power optimization subject to tight skew constraints in the presence of process variations. Key contributions include a new time-budgeting step for clock-tree tuning, accurate optimizations that satisfy budgets, modeling and optimization of variational skew. Our implementation, Contango 2.0, outperforms the winners of the ISPD 2010 clock-network synthesis contest on 45nm benchmarks from Intel and IBM.**

I. INTRODUCTION

Processor-based systems fueled the development of electronics since the 1960s. PCs were the main driver of growth in electronics in the 1990s, and in the 2000s mobile phones and other battery-powered consumer devices became a significant market segment, followed by automotive electronics. The emphasis in CPU design has shifted from high performance to power-performance-cost trade-offs, including the advent of multicore CPUs and the growing popularity of low-power ARM CPUs. In the netbook market, the low-power 1.6GHz Atom CPU from Intel is currently competing with ARM's multicore 2GHz Cortex-A9 CPUs and the 1GHz Cortex-A8, but 98% of world's mobile phones rely on ARM-based CPUs [13] which currently offer better power-performance-cost trade-offs than Intel CPUs [24].

ARM cores often drive system-on-chip (SoC) designs, laid out using low-power ASIC methodologies. Such methodologies perform automated clock-tree synthesis *after placement*, whereas traditional high-performance CPU methodologies pre-design clock networks and use active deskewing to lower clock skew and susceptibility to process variations [19]. Clock trees are more susceptible to variations than meshes (common in CPUs), but are 2-4 times more power-efficient. This is significant because clock distribution networks and corresponding sequential elements consume up to 70% of CPU power and can affect power-performance comparisons between CPUs [20].

Recent developments in embedded CPU design stress the need for low-power clock trees, yet also impose stringent skew limits, especially in the presence of process, voltage and temperature (PVT) variation for sub-45nm CMOS technology. Previous clock-tree methodologies rely on symmetric and regular tree topologies, such as H-trees and fishbones [1, chapter 43], which do not require sophisticated design algorithms (see Section II). However, these topologies experience difficulties with layout obstacles, non-uniform sink distributions, and varied sink capacitances. Fully-automated clock-tree synthesis supported by commercial EDA tools offers clear advantages in terms of capacitance, but may not be able to ensure sufficiently low skew for use in a 2GHz CPU. For example, the authors of [17] report clock trees generated by Cadence tools with skew that is orders of magnitude higher than the single-ps skew provided by clock meshes.

In this paper, we pursue the following research questions.

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

Our approach to answering these questions is inspired by the ISPD 2010 clock-network synthesis contest, which used several 2GHz CPU benchmarks from IBM and Intel to compare tools submitted by 10 teams across the world (downselected from 20 initial registrants). To evaluate the quality of the clock networks, difficult slew and skew constraints were checked against 45nm Monte-Carlo SPICE simulations that modeled PVT variations. Clock networks that cleared all constraints were compared by their total capacitance — a proxy for dynamic power. In this context, we developed a suite of algorithms for the design and thorough optimization of clock trees. The results of the ISPD 2010 contest offer a rare opportunity to compare multiple strategies for clock-network synthesis — the third-place team used symmetric trees [23], the second-place team used clock meshes, and our team won the contest by optimizing clock trees built by the DME algorithm [2], [6].

Specific innovations in our comprehensive methodology for clock-network synthesis include

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

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

TABLE I
CLOCK NETWORKS IN INDUSTRY CPUS [1, CHAPTER 43] AND ISPD 2010 BENCHMARKS FROM INTEL AND IBM (TABLE II).

Our empirical results are compared to those of the winners of the ISPD 2010 clock-network contest, where each team violated prescribed skew constraints (7.5 ps in most cases) on at least some benchmarks in the presence of variations. However, results reported in this paper satisfy skew constraints on every benchmark. Our clock trees have 4.2× smaller capacitance than clock meshes produced by CNSrouter [25], while exhibiting smaller skew.

The remainder of this paper is organized as follows. Section II covers background and prior work. Section III describes optimization objectives and variation modeling. Section IV explains initial tree construction with buffering. In Section V details the techniques for robustness improvements. Section VI outlines our skew optimization techniques. Our empirical results are described in Section VII. Conclusions are given in Section VIII.

II. BACKGROUND AND PRIOR WORK

Clock networks in microprocessors. A variety of clock network topologies and deskewing techniques were developed for microprocessors. Table I shows key parameters of clock networks in the microprocessors designed by IBM and Intel from the late 1990s to early 2000s [1, Chapter 43]. All those clock networks are regular, and only minimally adapt to sink locations.
Algorithms for clock tree construction. Theoretical developments in the 1990s suggested improvements over H-trees and spines using zero-skew trees (ZST) with minimum total length. To build a min-length ZST, the Deferred Merge Embedding (DME) algorithm was proposed in [2], [6] based on the concept of *merging segments*. Timing optimization based on Elmore delay was also incorporated into DME algorithms. DME algorithms assume a binary clustering of clock sinks. Such clustering can be found by a recursive horizontal-vertical partitioning algorithm called the *method of means and medians* (MMM) in [11] or the geometric matching algorithm (GMA) in [5]. Other simple algorithms for clock-tree synthesis are discussed in [1, Chapter 42].
Several methodologies for clock-tree tuning have recently been developed for the ISPD 2009 clock-network synthesis contest which focused on ASIC and SoC designs. A clock-synthesis methodology for SPICE-accurate skew optimization with tolerance to voltage variations called Contango was proposed in [14]. Dynamic Nearest-Neighbor Algorithm to generate tree topology and Walk-Segment Breadth First Search for routing and buffering were proposed in [22]. A three-stage CLR-driven CTS flow based on an obstacle-avoiding balanced clock-tree routing algorithm, monotonic buffer insertion, as well as wire-sizing and wire-snaking is proposed in [15]. A Dual-MST geometric matching approach is proposed in [16] for topology construction, along with recursive buffer insertion and a way to handle blockages. SoC methodologies often spend significant effort dealing with hundreds of layout obstacles, while CPU layouts include very few obstacles. However, skew constraints are more difficult in CPU clock synthesis. Because of these differences and due to the incorporation of process variation into the ISPD 2010 contest, most of the above techniques were not adopted by the contestants.

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

TABLE II
ISPD 2010 BENCHMARKS BASED ON 45 NM MICROPROCESSOR DESIGNS. Ω_Δ IS THE *local skew limit*, AND Δ IS THE *local skew distance limit* RESPECTIVELY (SEE SECTION III-A). NOMINAL VOLTAGE IS 1.0V AND ON-CHIP VARIATIONS (ν) ARE ACCOUNTED BY 15% VOLTAGE VARIATION AND 10% VARIATION OF WIRE PARASITICS FOR ALL BENCHMARKS [25].

III. MODELING AND OBJECTIVES

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

A. Global and local skew

Common terminology and notation are introduced next.

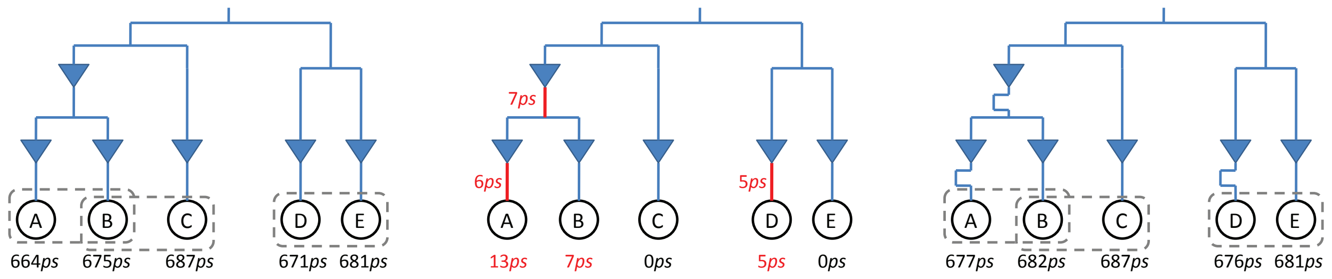
Definition 1: Given a clock tree Ψ, let $\lambda(s_i)$ be the clock latency (insertion delay) at sink $s_i \in \Psi$. Then the skew between two sinks s_i and $s_j \in \Psi$ is defined as

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

Global skew is defined as

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

Nominal values of $\text{skew}^\Psi(s_i, s_j)$ and ω^Ψ are computed neglecting the impact of variations.
Global skew can be improved by decreasing $\max_{i\in\Psi} \lambda(s_i)$ (speeding up the slowest sinks) or increasing $\min_{i\in\Psi} \lambda(s_i)$ (delaying the fastest sinks). Previous publications on clock network synthesis were focused on reducing *global skew* with or without the presence of variations [2], [3], [10], [12], [14]–[16], [22], [26]. However, in a large clock network, skew



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

Fig. 1. Local-skew slack for sinks and edges when $\Omega_\Delta = 5ps$. (a) Sink pairs within distance Δ are enclosed by dashed lines. $\omega_\Delta = 12ps$ based on sink latencies and Δ. (b) Local skew-slack for sinks are computed by Algorithm 1. The algorithm for edge-slack computation is described in [14, Section 3]. (c) ω_Δ is reduced to $5ps$ after optimizations, which satisfies the local skew constraints.

between adjacent and connected sinks is a more meaningful optimization objective [8], [18]. *Local skew* is defined by restricting eligible sink pairs to be within distance $\Delta > 0$, which is determined for a given circuit after timing-driven placement.

Definition 2: Given a clock tree Ψ and a *local skew distance bound* $\Delta > 0$, let $\mathrm{dist}(s_i, s_j)$ be the Manhattan distance between sinks s_i and $s_j \in \Psi$. Then *the worst local skew* [25] is defined as

$$\omega_\Delta^\Psi = \max_{\mathrm{dist}(s_i,s_j)<\Delta} \mathrm{skew}^\Psi(s_i, s_j) \quad (3)$$

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

Definition 3: Consider a clock tree Ψ, a local skew distance bound $\Delta > 0$, variation model ν and target yield $0 < y \le 1$. Let Ψ_ν be the clock tree Ψ with variation ν and $f(t)$ be the cumulative distribution function of $\omega_\Delta^{\Psi_\nu}$. Then *the worst local skew with variation* is defined as

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

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

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

B. Local-skew slack

Given a clock tree with known sink latencies, one can optimize it using delay budgets derived from the sink- and edge-slack calculation [14, Section 3], followed by global skew optimization to reduce global skew below Ω_Δ. This strategy is sound because local skew ω_Δ cannot exceed global skew. However, global skew optimizations attempt to reduce skew between sinks more distant than Δ, which may require unnecessary increase in power.

To tune the clock tree on a tight power budget, we propose the concept of *local-skew slack*.

Definition 4: Given a clock tree Ψ and local-skew constraints Ω_Δ, the *local-skew slack* $\sigma(s)$ for a sink $s \in \Psi$ is the minimum amount of additional delay in picoseconds for s, so that the tree satisfies $\omega_\Delta^\Psi < \Omega_\Delta$.

The Δ-neighborhood of sink s_i is $\mathcal{N}(s_i) = \{ s \in \Psi \mid \mathrm{dist}(s, s_i) < \Delta\}$. It is used in Algorithm 1 to calculate $\sigma(s)$ for every sink. This algorithm uses $\mathrm{varEst}(s_i, s_j) = 0$ in the absence of variations, and otherwise the definition in Section III-C.

Once local-skew slacks $\sigma(s)$ are computed for all sinks, we define local-skew slack of tree edge e as the smallest slack of a downstream sink. Edge slacks in the entire tree can be computed by one recursive tree traversal in linear time, giving the optimal amount of tuning to improve worst local skew [14, Section 3]. Figure 1 illustrates the computation of local-skew slack for sinks and edges.

C. Modeling process variation

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

When two sinks can be connected by a short path in the tree, variation of skew between them is small. On the other hand, variational skew between sinks that are geometrically close can be significant if the unique tree-path between them is long. This is illustrated in Figure 2.

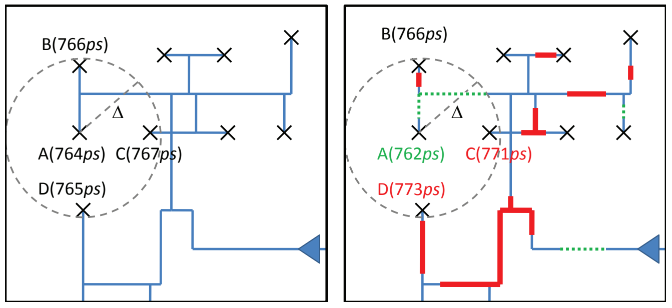
Our key insight is that the impact of variations on skew between two sinks is closely correlated with tree path length and how the tree path is buffered. Therefore, for a given technology node, buffer library, wires and variation model, we propose to build a look-up table with comprehensive information regarding the worst-case variation on skew for various paths between pairs of sinks.

Algorithm 1 Computing local-skew slack for sinks

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

Definition 5: Given a technology node $\mathcal{T}$, buffer and wire library $\mathcal{B}$, variation model ν and desired yield $0 < y \leq 1$, let $\Xi_{\mathcal{T},\mathcal{B},\nu,y}[w, b, t]$ be the variation-estimation table which returns the worst-case increase in skew (with probability y) between two sinks connected by a tree path of length w with b buffers and the buffer type t. When multiple buffer types are used in the tree path, t is the smallest type in the tree path, so as to avoid under-estimation of variation.

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



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

Fig. 2. The impact of variations on local skew. Sinks are indicated by crosses, the clock source is indicated by a solid triangle. Nominal skew of 3 ps is shown in (a). Full skew of 11 ps is shown in (b), where some tree edges are delayed (thick red) and some are sped up (dotted green) by random variations. Only sink A is within the local skew distance from sinks B, C and D.

variation table is accessed by the function $\mathrm{varEst}(s_i, s_j) = \Xi_{\mathcal{T},\mathcal{B},\nu}[\mathcal{L}(s_i, s_j), \mathrm{b_n}(s_i, s_j), \mathrm{b_t}(s_i, s_j)]$.

To estimate the impact of variations when optimizing clock trees we utilize $\mathrm{varEst}()$ when computing local-skew slack for each sink (Algorithm 1). Without considering variations, it is sufficient to satisfy $\mathrm{skew}(s_i, s_j) < \Omega_\Delta$ for all pairs of sinks within Δ. However, in the presence of variations, we have the following result.

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

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

IV. Initial Tree Construction and Buffer Insertion

We invoke the unmodified ZST-DME algorithm [4], [10] and perform initial buffer insertion to minimize source-to-sink Elmore delay, rather than skew or capacitance [7], [21]. Elmore delay is too inaccurate for skew optimization, but our approach creates significant room for tuning the clock tree by delaying fast paths [14]. In the presence of layout obstacles, proper obstacle-handling is required to avoid violations due to obstacles. The ISPD 2010 benchmarks include obstacles over which wire-routing is possible but buffer insertion is not allowed. We adapted a simple and robust technique for obstacle avoidance in clock trees from [14] which repairs obstacle violations in the trees obtained by the ZST-DME algorithm.

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

Selecting buffer types for initial buffer insertion is also important. Given an initial tree without buffers Ψ_0, let $t(s_i, s_j)$

be the type of a buffer required for the tree path between two sinks s_i and $s_j \in \Psi_0$ to satisfy $\mathrm{varEst}(s_i, s_j) < \Omega_\Delta$. $t(s_i, s_j)$ can be found from the variation-estimation table $\Xi_{\mathcal{T},\mathcal{B},\nu}$ with $\mathcal{L}(s_i, s_j)$. Since $\mathrm{b_n}(s_i, s_j)$ is not available at this step, it is difficult to find the exact required $t(s_i, s_j)$. However, because $\mathrm{b_n}(s_i, s_j)$ and $\mathcal{L}(s_i, s_j)$ are highly correlated with each other, $\mathrm{b_n}(s_i, s_j)$ can be estimated by modeling it with the average number of buffers corresponding to $\mathcal{L}(s_i, s_j)$. Once $\mathrm{b_n}(s_i, s_j)$ is estimated, $t(s_i, s_j)$ can be computed as described in Section V. The initial buffer type (t_0) for a given initial tree is computed as

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

Once t_0 is determined, we adopt the fast variant of van Ginneken's algorithm from [21] for initial buffer insertion. $\mathrm{b_n}(s_i, s_j)\ \forall s_i, s_j \in \Psi$ is determined after initial buffer insertion and more accurate $t(s_i, s_j)$ can be obtained. For sink pairs that do not satisfy $\mathrm{varEst}(s_i, s_j) < \Omega_\Delta$, we use the robustness-improvement algorithm from Section V to ensure that the tree eventually satisfies $\omega_\Delta^\Psi < \Omega_\Delta$.

V. Robustness improvements

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

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

Theorem 2: If skew_{est}^Ψ is an upper bound of ω_Δ^Ψ and skew_{est}^Ψ+$\mathrm{varEst}(s_i, s_j) < \Omega_\Delta$ *for all s_i and s_j* then

$$\omega_{\Delta,\nu,y}^\Psi < \Omega_\Delta \tag{7}$$

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

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

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

VI. Skew optimizations

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

A. Wire snaking

Wire sizing and wire snaking are popular techniques for skew optimization and are often able to reduce global or local skew down to the practical skew limit. In this context, however, we exclude wire sizing because narrowing down a wire in the middle of a clock tree is risky due to the impact of variations. We extend the wire snaking technique from [14] to improve its speed and accuracy, while limiting its use of routing resources.

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

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

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

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

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

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

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

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

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

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

If the wire snaking algorithm over-estimates $L_{sn}^i(e)$ and results in $T_{actual}^i(e) > T_{target}^i(e)$ for any edge e in any i-th iteration, then $TotalCap_{sn}$ exceeds $TotalCap_{ideal}$ after all the iterations of the wire snaking algorithm because it means there exists excessive delay of a wire which results in excessive delay of some sinks and possibly increases local skew around the sinks. Therefore the wire snaking algorithm must produce $L^i_{sn}(e)$ which satisfies $T^i_{actual}(e) \leq T^i_{target}(e)$ for optimized power consumption by the clock tree. However, if the gap between $T^i_{actual}(e)$ and $T^i_{target}(e)$ is too big, more iterations will be needed for $T_{actual}(e)$ to approach $T_{target}(e)$. We improve the accuracy of wire snaking in two ways.

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

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

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

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

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

Optimal node selection for wire snaking. Figure 3 compares two different styles of wire snaking. Figure 3(b) illustrates undesired delay of sinks after wire snaking on non-buffer-output nodes. The increased capacitance and resistance by wire snaking affects the driving buffer which results in additional delay of slow sinks. Wire snaking at buffer output nodes, as in Figure 3(c), is much more accurate than wire snaking at any branch. Limiting wire snaking to buffer output nodes reduces the number of SPICE calls required for clock-tree tuning. This also reduces the number of simulations for building the look-up table by limiting the number of target nodes to be tested. Wire snaking usually increases slew rate of input nodes of downstream buffers. To prevent slew violation, slew rate numbers of downstream buffers are checked and if the worst slew rate is more than 70% of the given slew limit, the target node is excluded from wire snaking.

B. Delay buffer insertion

The local skew of a sink cluster driven by the same final buffer is often negligible. However, highly unbalanced sink capacitances or layout obstacles in those clusters can result in significant local skew. An alternative technique is needed because wire snaking in Section VI-A is inapplicable. In this

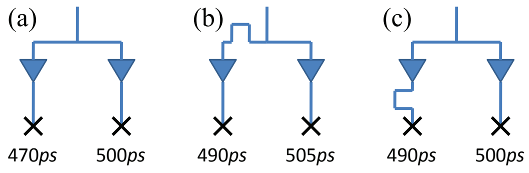


Fig. 3. Comparison of different wire snaking strategies to satisfy $\Omega_\triangle = 10ps$. (a) Unoptimized sink latencies are shown. $20ps$ of additional delay is required for the left sink. (b) Wire snaking at non-buffer output nodes results in undesired delay at the right sink. (c) The snaked wire is isolated from the right sink by the left buffer, therefore only the left sink is delayed and $\omega_\triangle$ satisfies local skew constraints.

case, inserting a buffer at the target node is very efficient for two reasons. First, skew can be reduced by the delay of the inserted buffer. Second, further precise wire snaking is possible because the inserted buffer isolates the target node from the remainder of the cluster.

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

For each path from the buffer to the sinks, inserting at most one buffer is sufficient since the wire snaking algorithm in Section VI-A can be invoked again at the output node of inserted buffers. Figure 4 illustrates delay buffer insertion algorithm followed by wire snaking. When a delay buffer is inserted, it is placed at the node so that the input capacitance of a delay buffer is comparable to the sum of downstream sink and wire capacitance of the target node, thus sink latency in the other path changes very little. (see Figure 4 (b)).

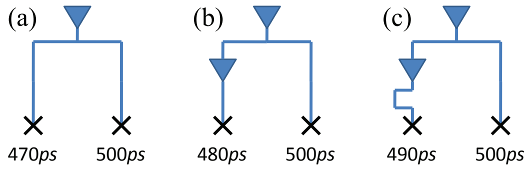


Fig. 4. Delay buffer insertion and subsequent wire snaking when $\Omega_\triangle$=10ps, the delay of the buffer $d(B)$=10ps. (a) Unoptimized sink latencies are shown. (b) Delay buffer insertion for skew reduction and isolation of the target node. (c) The snaked wire is isolated from the right sink by the delay buffer.

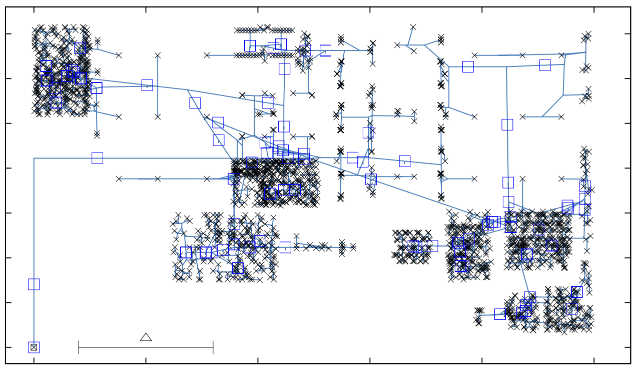


Fig. 5. Our clock tree for *ispd10cns07*. Sinks are indicated by crosses, buffers are indicated by blue rectangles. $\Delta = 600\mu m$ is indicated at the left-bottom of the figure.

VII. Empirical validation

Our implementation, Contango 2.0, is written in C++ and is based on our software Contango 1.0 [14] that shared the first place at the ISPD 2009 clock-network synthesis contest. Contango 2.0 was the sole winner of the ISPD 2010 contest, but we now report significantly stronger results.

ISPD 2010 benchmarks. Table II lists the statistics of all benchmarks from the ISPD 2010 contest. The contest limited slew to $100ps$, and all reported clock networks satisfy this constraint. Slews in Contango 2.0 trees do not exceed $81ps$. Table III compares Contango 2.0 with CNSrouter and NTUclock. Clock networks produced by our software have smaller capacitance than CNSrouter and NTUclock on average by **4.22**$\times$ and **4.13**$\times$ respectively. The contest imposed local skew constraints with yield $y = 95\%$. Our clock trees always yield $> 95\%$, while CNSrouter violates yield constraints on three benchmarks and NTUclock on all benchmarks except one. All three teams satisfied the 12-hour runtime limit for all benchmarks. Our data suggest that wire snaking usually increases wire length by 1-3% (5.43% in one case), which is small enough to neglect the negative effects of wire snaking. Figure 7 compares probability density functions (pdf) produced by Monte-Carlo SPICE simulations of our clock trees to those of clock meshes produced by CNSrouter. One such clock tree is illustrated in Figure 5. Despite the dramatic differences in network topology and total capacitance between trees and meshes, some of the plots in Figure 7 bear striking resemblance (cns01, cns02, cns04, cns05). To explain this phenomenon, we recall that meshes cannot be buffered directly and are therefore driven by a buffered clock tree. Such a clock tree can be constructed by the same DME algorithm that we use, which is why the pdf profiles in Figure 7 reflect the pointset of sink locations. Apparently, the mesh does not significantly change this profile.

Power versus robustness to variations. Figure 6 describes experiments on benchmark *ispd10cns08* with different local skew constraints. When tight local skew constraints are given, large buffers are required to ensure robustness to variations, increasing the capacitance of the clock tree. On the other hand, a large portion of capacitance can be saved when local skew constraints are loose. To clarify the impact of variation, we plot variational skew (y-axis), defined as $\omega_{\Delta,\nu,y}$ - ω_{Δ} for Δ, ν, y from Table II.

VIII. Conclusions

Power-performance-cost trade-offs are becoming a major issue in modern high-performance CPU clock designs. Mesh structures often sacrifice power to improve robustness to variations. We propose a tree solution for CPU clock routing that improves power consumption under tight skew constraints in the presence of variations. To this end, we introduce the notion of local-skew slack for clock trees, a model for variational skew, a path-based technique to enhance robustness, a new time-budgeting algorithm for clock-tree tuning and accurate optimizations that satisfy budgets. We have shown that clock trees can be tuned to have nominal

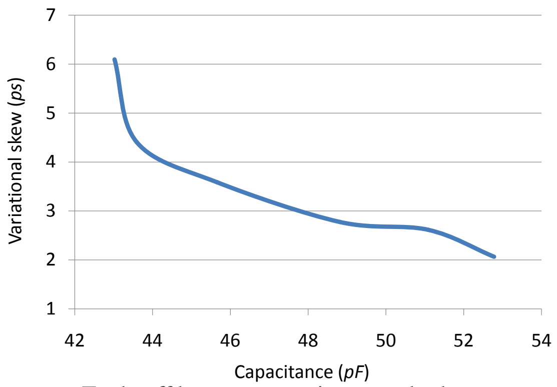


Fig. 6. Trade-off between capacitance and robustness on *ispd10cns08*. The x-axis represents total capacitance of a tree and y-axis represents the maximal variational skew at 95% yield.

skew below 5 ps and total skew in single picoseconds in the presence of variations. Our optimizations not only satisfy given skew constraints and target yield but also lead to $4.22\times$ capacitance improvement on average over mesh structures proposed in the ISPD 2010 contest. Furthermore, our clock trees had a higher yield than the meshes because meshes are not as easy to tune for nominal skew. Our analysis does not consider gated clocks, inductive effects and short-circuit power in meshes, but these factors generally favor trees over meshes. Our strong empirical results suggest that clock trees constructed using accurate variational skew modeling and optimizations have distinct advantage in power consumption and similar robustness as meshes. Hence, our techniques may improve power of future CPUs without sacrificing other performance metrics.

Acknowledgements. This work was partially supported by the US National Science Foundation (NSF) through a CAREER award, as well as by Sun Microsystems and Texas Instruments.

References

[1] C. J. Alpert, D. P. Mehta, S. S. Sapatnekar, Eds., “Handbook of Algorithms for Physical Design Automation,” *CRC Press*, 2009.
[2] K. D. Boese, A. B. Kahng, “Zero-Skew Clock Routing Trees with Minimum Wirelength,” *ASIC*‘92, pp.111-5.
[3] T.-H. Chao et al., “Zero Skew Clock Routing with Minimum Wirelength,” *IEEE Trans. on Circ. & Sys.*, 39(11), pp. 799-814, 1992.
[4] J. Cong et al, “Bounded-Skew Clock and Steiner Routing,” *ACM Trans. on Design Autom.* 1998, pp. 341-388.
[5] J. Cong, A. B. Kahng, G. Robins, “Matching-based Methods for High-performance Clock Routing,” *IEEE Trans. on CAD* 12(8), pp.1157-1169, 1993.
[6] M. Edahiro, “A Clustering-Based Optimization Algorithm in Zero-Skew Routings,” *DAC*‘93, pp. 612-616.
[7] L. van Ginneken, “Buffer Placement in Distributed RC-tree Networks For Minimal Elmore Delay,” *ISCAS*‘90, pp.865-868.
[8] D. Harris, M. Horowitz and D. Liu, “Timing Analysis Including Clock Skew,” *IEEE Trans. on CAD* 18(11), pp.1608-1618,1999.
[9] R. Ho, K. Mai, M. Horowitz, “The Future of Wires,” *Proc. IEEE*, 89(4), pp. 490-504, 2001.
[10] J.-H. Huang, A. B. Kahng, C.-W. Tsao, “On Bounded-Skew Routing Tree Problem,” *DAC*‘95, pp.508-513.

ISPD'10 Bench.	CNSROUTER 3/16/2010				NTUCLOCK 3/16/2010				CONTANGO 2.0 4/14/2010					
	Skew+Var		Total		Skew+Var		Total		Nom.	Skew+Var			Total Cap.	
	Mean	$\omega_{\Delta,\nu,95}$	Cap.	◷	Mean	$\omega_{\Delta,\nu,95}$	Cap.	◷	skew	Mean	$\omega_{\Delta,\nu,95}$	Yield	(wire snaking)	◷
cns01	5.27	7.32	841.2	20	6.71	~~8.66~~	293.9	15	2.51	5.16	7.01	97.4%	**198.3**(2.63)	12015
cns02	6.27	~~8.33~~	1454	81	8.27	~~10.73~~	832.5	176	2.99	5.58	7.34	95.8%	**375.9**(1.26)	25006
cns03	2.07	2.70	162.5	8	6.82	~~8.63~~	167.1	6	1.5	3.03	4.18	99.6%	**55.86**(1.04)	3840
cns04	2.83	3.98	277.1	32	7.45	~~9.55~~	325.2	58	2.07	3.26	4.46	99.9%	**71.84**(1.46)	6075
cns05	2.88	4.38	175.5	7	5.30	6.98	130.4	11	1.50	3.01	4.41	99.9%	**37.69**(5.43)	2406
cns06	12.38	~~14.03~~	133.3	11	406.9	~~416.6~~	1577	10	4.29	5.03	6.05	99.9%	**47.81**(2.99)	2660
cns07	12.39	~~15.74~~	309.2	45	6.17	~~8.12~~	275.6	66	2.22	3.41	4.58	99.9%	**72.66**(1.14)	2351
cns08	6.15	7.33	222.8	19	5.94	~~7.64~~	165.9	7	3.42	4.15	5.15	99.9%	**52.49**(1.86)	1987
Relative			**4.22**				**4.13**						**1.0**	

TABLE III
RESULTS ON THE ISPD 2010 CONTEST BENCHMARK SUITE. SKEW NUMBERS ARE REPORTED IN ps, CAPACITANCE IN pF AND CPU TIME IN s. THE NUMBERS IN PARENTHESES OF THE CAPACITANCE COLUMN REFER TO THE FRACTION OF CAPACITANCE OF THE SNAKED WIRES IN %. SKEW CONSTRAINT VIOLATIONS ARE SHOWN IN STRIKETHROUGH FONT. OTHERWISE, SKEW RESULTS ARE NOT COMPARABLE BECAUSE SKEW CAN BE TRADED FOR CAPACITANCE, WHICH WAS THE PRIMARY OBJECTIVE OF THE CONTEST. ALL NETWORKS PRODUCED BY THESE TOOLS SATISFY SLEW CONSTRAINTS IMPOSED AT THE ISPD 2010 CONTEST. DUE TO LIMITED PAGE SPACE, WE DO NOT INCLUDE RESULTS FOR THE OTHER TEAMS, BUT SIGNIFICANTLY OUTPERFORM THEM IN SOLUTION QUALITY.

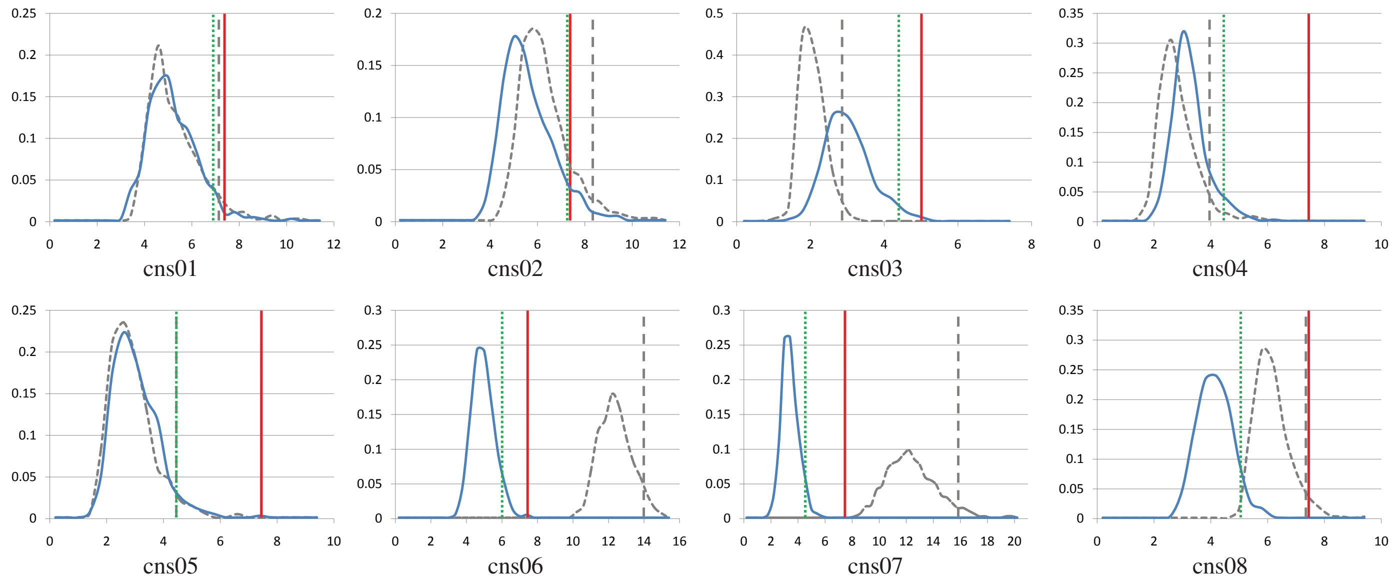


Fig. 7. Probability density functions for worst local skew of our clock trees (blue line) and meshes produced by CNSrouter (gray dashed line) for the eight ISPD 2010 benchmarks, calculated using 500 independent SPICE runs for each benchmark. The x-axis shows skew in picoseconds. Local skew limits (Ω_Δ) are shown with red solid lines, and the 95%-ile of local skew ($\omega_{\Delta,\nu,0.95}$) are shown by dotted green lines (our work) and dashed gray vertical lines (CNSrouter).

[11] M. A. B. Jackson, A. Srinivasan, E. S. Kuh, "Clock Routing for High-performance ICs," *DAC*'90, pp. 573-579.

[12] A. B. Kahng, C.-W. Tsao,"Practical Bounded-Skew Clock Routing," *J. VLSI Signal Proc.* 16(1997), pp.199-215.

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

[14] D.-J. Lee, I. L. Markov, "Contango: Integrated Optimization of SoC Clock Networks," *DATE*'10, pp. 1468-1473.

[15] W.-H Liu, Y.-L Li, H.-C. Chen, "Minimizing Clock Latency Range in Robust Clock Tree Synthesis," *ASPDAC*'10, pp. 389-394.

[16] J. Lu et al, "A Dual-MST Approach for Clock Network Synthesis," *ASPDAC*'10, pp. 467-473.

[17] C.-L. Lung et al, "Clock Skew Optimization Considering Complicated Power Modes," *DATE*'10, pp. 1474-1479.

[18] P. J. Restle et al, "A Clock Distribution Network for Microprocessors," *IEEE JSSC* 36(5), pp.792-799,2001.

[19] R. S. Shelar, "An Algorithm for Routing with Capacitance/Distance Constraints for Clock Distribution in Microprocessors," *ISPD*'09, pp. 141-148.

[20] R. S. Shelar, M. Patyra, "Impact of Local Interconnects on Timing and Power in a High Performance Microprocessor," *ISPD*'10, pp. 145-152.

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

[22] X.-W. Shih et al, "Blockage-Avoiding Buffered Clock-Tree Synthesis for Clock Latency-Range and Skew Minimization," *ASPDAC*'10, pp. 395-400.

[23] X.-W. Shih and Y.-W. Chang, "Fast Timing-Model Independent Buffered Clock-Tree Synthesis," *DAC*'10, pp. 80-85.

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

[25] C. Sze, "ISPD 2010 High Performance Clock Network Synthesis Contest: Benchmark Suite and Results," *ISPD*'10. http://http://www.sigda.org/ispd/contests/10/ISPD2010-cns-contest-v3.pdf

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