Short Papers

Zero Skew Clock-Tree Optimization With Buffer Insertion/Sizing and Wire Sizing

Jeng-Liang Tsai, Tsung-Hao Chen, and Charlie Chung-Ping Chen

***Abstract*—Clock distribution is crucial for timing and design convergence in high-performance very large scale integration designs. Minimum-delay/power zero skew buffer insertion/sizing and wire-sizing problems have long been considered intractable. In this paper, we present *ClockTune*, a simultaneous buffer insertion/sizing and wire-sizing algorithm which guarantees zero skew and minimizes delay and power in polynomial time. Extensive experimental results show that our algorithm executes very efficiently. For example, *ClockTune* achieves 45 $\times$ delay improvement for buffering and sizing an industrial clock tree with 3101 sink nodes on a 1.2-GHz Pentium IV PC in 16 min, compared with the initial routing. Our algorithm can also be used to achieve useful clock skew to facilitate timing convergence and to incrementally adjust the clock tree for design convergence and explore delay–power tradeoffs during design cycles. *ClockTune* is available on the web (http://vlsi.ece.wisc.edu/Tools.htm).**

***Index Terms*—Buffer insertion, buffer sizing, clock tree, optimization, wire sizing, zero skew.**

I. Introduction

In the multigigahertz design era, clock design plays a crucial role in determining chip performance and facilitating timing and design convergence. First, clock skew directly affects chip performance in a close to one-to-one ratio since it has to be counted as cycle-time penalty. Second, incremental clock-tree adjustment enables fast design convergence by avoiding the potentially divergent design iterations. Since designs are subjected to change on a daily basis, the clock trees need to be incrementally adjusted accordingly with minimum changes to ensure acceptable clock skew. Third, since interconnect delay dominates over gate delay, timing plans often cannot be met due to physical effects. Recently, useful skew [2] concepts have also been widely proposed to speed-up timing convergence in order to compensate for the timing uncertainties resulting from the physical layout. From the above analysis, it is crucial to develop clock tuning algorithms that can balance clock skew with minimum adjustments.

An excellent survey of interconnect optimization techniques can be found in [3]. Among the techniques suitable for clock-tree optimization are buffer insertion/sizing and wire sizing since these do not need to modify the existing routing. In [4], a three-stage optimization algorithm is proposed to minimize the delay and skew of a clock tree. A reported 27 $\times$ delay improvement was achieved by buffer insertion and buffer sizing. In [5], an iterative algorithm performs wire sizing one segment at a time and about 1.5 $\times$ to 3 $\times$ improvement on minimum delay was observed. Two major approaches have been used to integrate buffer insertion/sizing and wire-sizing techniques for delay and power optimization. In [6]–[8], the simultaneous buffer insertion/sizing and wire-sizing problems are formulated as optimization problems, in which the maximum delay of each sink node is constrained. In [9] and [10], bottom-up dynamic programming algorithms, based on the method in [11], are used to find optimal solutions for a subtree and propagate the solutions up toward the root node. These methods perform optimizations without modifying the clock routing, but do not guarantee zero skew.

Recent work [12] integrates wire sizing into the deferred-merging embedding (DME) algorithm [13], which allows a zero skew clock tree to benefit from wire sizing and buffer insertion. However, the zero skew property is achieved by moving the merging points and the clock routing might be changed to accommodate the skew caused by design changes. This may affect the detail routing. To the best of the authors' knowledge, there is no existing simultaneous wire sizing and buffer insertion/sizing algorithm which finds the minimum-delay and minimum-power zero skew solutions without modifying the existing routing.

In this paper, we propose a novel clock-tuning algorithm, *ClockTune*, which considers buffer insertion/sizing and wire sizing at the same time, while maintaining the clock tree zero skew. *ClockTune* first calculates the feasible delay and power information for each node in a bottom-up fashion. After the desired delay and power is chosen from the feasible region, a buffering and wire sizing is determined in a top-down fashion. Although we focus on achieving zero skew, *ClockTune* can also be used to achieve useful skew to tackle timing problems. Moreover, if the clock routing encounters design changes, *ClockTune* is able to rebalance the clock tree by local adjustment.

The rest of this paper is organized as follows. In Section II, we formulate the problems and introduce the models and notations we use in this work. In Section III, the fundamentals of our algorithm are introduced. Section IV provides the algorithm framework and gives the details of our *ClockTune* algorithm. Section V details the complexity analyses. Section VI presents our experimental results and Section VII concludes this paper.

II. Preliminaries

The minimum-delay/power zero skew wire sizing (min-ZSWS) problem was solved in [14], and the proposed method provides a good basis for understanding this work. However, [14] did not consider buffer insertion/sizing, which is a more effective way of reducing clock delay. We first define both problems and repeat part of the conclusions of [14] to make this work self-contained.

Problem Definition 1: Min-ZSWS Problem: Given a clock tree T, find a set of wire widths with bounded delay and power consumption such that the zero-skew constraint is satisfied and the delay and switching power are minimized.

Problem Definition 2: Min-ZSBWS Problem: Given a clock tree T, find a set of buffer locations, buffer widths, and wire widths with bounded delay and power such that the zero-skew constraint is satisfied, and the delay and power are minimized.

We assume that the initial routing of the clock tree is given and there exists some buffering and sizing combinations such that the clock tree is zero skew. If our algorithm fails to find a zero skew solution, then it is impossible to achieve zero skew by any designer with only buffer insertion/sizing and wire sizing. In these cases, the initial routings should be

Manuscript received April 25, 2003; revised August 24, 2003. This work was supported in part by the National Science Foundation under Grant CCR-0093309 and Grant CCR-0204468. This paper was recommended by Guest Editor C. J. Alpert.

J.-L. Tsai and T.-H. Chen are with the Electrical and Computer Engineering Department, University of Wisconsin, Madison, WI 53706 USA (e-mail: jltsai@cae.wisc.edu; tchen@cae.wisc.edu).

C. C.-P. Chen is with the Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan (e-mail: cchen@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2004.825875

TABLE I
NOTATIONS

T_v	A clock-tree with given routing rooted at node v
v_l	The left child of node v
v_r	The right child of node v
e_v	The edge between node v and its parent node
l_v	The length of edge e_v
b_v	The buffer above node v, $b_v = \phi$ if not buffered
$w(e_v)$	The wire-width of edge e_v
$w(b_v)$	The width of buffer b_v, $w(\phi) = 0$
$r(e_v)$	The resistance of wire e_v, $r(e_v) = \frac{l_v r_0}{w(e_v)}$
$c(e_v)$	The capacitance of wire e_v, $c(e_v) = l_v w(e_v) c_0$
d_v	Elmore-delay from node v to any leaf node in T_v
c_v	Total down-stream capacitance seen at node v
c_{sv}	Total capacitance shielded from node v by buffers
r_0, c_0	Wire resistance and capacitance per μm^2
r_b, c_b	Unit-size buffer resistance and capacitance
t_c	Buffer intrinsic delay
w_m, w_M	Predefined minimum and maximum wire-width
w_{bm}, w_{bM}	Predefined minimum and maximum buffer-size

regenerated. Although our algorithm does not require the initial routing to be zero skew, it is easier to optimize an unbuffered zero skew routing. In this work, we use the BB+DME [13] algorithm to generate initial zero skew routings. It has been shown that useful skew can be used to speed-up timing convergence and improve circuit performance. The DME algorithm can also be used to construct useful skew clock trees by taking into account the relative skews of the clock sinks, while generating merging segments. When a pair of clock sinks has a large relative skew, we have to allow wire snaking in order to find a feasible merging segment. A better solution is to group the clock sinks according to their skews. The clock sinks requiring late clock arrival time are selected for merging first. The sinks requiring early clock arrival time are merged into the clock tree later (therefore, they are closer to the root node). In this way, we can reduce or totally avoid wire snaking. In the rest of this paper, we focus on optimizing the given initial routing.

A. Notations

Table I lists the notations used throughout this paper. In Table I, T_v is a binary tree. However, any tree structure can be represented as a binary tree, if the length of an edge is allowed to be zero. In this paper, buffers are only allowed to be inserted right above a node, and to simplify the discussion, no buffer is allowed to insert above a leaf node.

B. Delay and Power Models

There are two delay components in a clock tree: interconnect delay and gate delay. In this paper, the resistance–capacitance (*RC*) models for interconnects and buffers and the Elmore-delay model for delay calculation are used. For a wire with length l and width w, the wire resistance is lr_0/w and the wire capacitance is lc_0w. The wire capacitance is further divided into two equal capacitors attached at both ends of the wire. For a buffer with gate width w_b, the gate capacitance at its input is w_bc_b. The gate is modeled as a ramp voltage source with an effective output resistance r_b/w_b. The ramp voltage source has a delay t_c, which models the intrinsic delay of the buffer.

The power consumed by the clock tree can be modeled as $P = fCV^2 + P_s + P_l$, where f is the switching frequency, V is the voltage swing, and C is the total interconnect capacitance, gate capacitance of the buffers, and sink loads. P_s accounts for the buffer short-circuit power and P_l accounts for the leakage power. In a usual design, the last two terms are usually much smaller and the total capacitance is a good measure of the total power consumption [15].

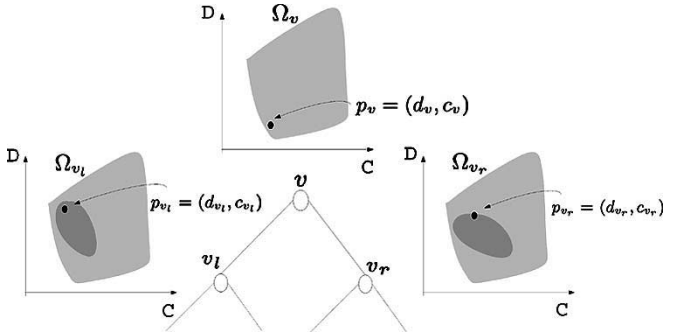

Fig. 1. Illustration of the existence property.

III. Design Space and DC Region

Considering the min-ZSWS problem, if T_v has n edges, then there are n wire widths to be determined. Every *embedding* of T_v (a set of wire widths which satisfies zero skew and wire-width constraints) is a point in the n-dimensional design space. Since we are only interested in the delay and power of the embeddings, we can project all the embeddings onto the *D–C plane*: the X–Y plane with delay value on Y axis and capacitance load value on X axis. The projection of the embeddings form a *DC region*, Ω_v, on the *D–C plane*. The lower-left edge of the *DC region* is the delay/power tradeoff curve, and previous works [9]–[11] have emphasized finding the solutions which lie on this curve while pruning out suboptimal solutions. These approaches have two drawbacks. First, the combinations that lie on the curve grow polynominally [11]. Second, early pruning suboptimal solutions may result in suboptimal global solutions because a suboptimal solution of a subtree can be part of an optimal global solution of the entire clock tree. For example, a clock tree with a small left subtree and a large right subtree would require the left subtree to be sized suboptimally in order to match the delay of the right subtree.

In [14], a different approach is used to solve the min-ZSWS problem which relies on the following property.

Property 1: Existence Property: For every point $p_v = (d_v, c_v) \in \Omega_v$, there exists at least one pair of points $p_{v_l} = (d_{v_l}, c_{v_l}) \in \Omega_{v_l}$ and $p_{v_r} = (d_{v_r}, c_{v_r}) \in \Omega_{v_r}$, such that the corresponding embeddings of T_{v_l} and T_{v_r} are the same as in the embedding of T_v from p_v.

The existence property is the restatement that for a feasible design of T_v, its designs of subtrees T_{v_l} and T_{v_r} are also feasible, thus, their projections are in Ω_{v_l} and Ω_{v_r}. In Fig. 1, the light grey areas are the DC regions of T_v, T_{v_l}, and T_{v_r}. For the projection of a feasible design of T_v, p_v, at least one pair of p_{v_l} and p_{v_r} in Ω_{v_l} and Ω_{v_r} satisfies the following:

$$c_v = \sum_u (c_u + l_u w(e_u) c_0), \qquad u \in \{v_l, v_r\} \tag{1}$$

$$d_v = d_u + \frac{l_u^2 r_0 c_0}{2} + \frac{l_u r_0 c_u}{w(e_u)} \tag{2}$$

and all pairs of p_{v_l} and p_{v_r} form the dark grey areas.

It is worth mentioning that in the Elmore-delay model, a capacitor is used to model an *RC* tree and the calculated delay only matches the first moment of the exact impulse response. If a more accurate delay model is required, an *RC* model or capacitor-*RC* (*CRC*) model can be adopted to model an *RC* tree [16]. By adding another axis to the *D–C plane* for the additional parameter, it forms a *D–C space*. The DC region becomes the projection of the embeddings on the *D–C space*.

In the min-ZSBWS problem, buffering is allowed and c_v is the total capacitance of T_v minus the capacitance shielded by first-level buffers below v. Inserting a buffer also changes the signal polarity, thus, Ω_v is split into two sets. $\Omega_{\rm vp}$ is the projection of embeddings with even

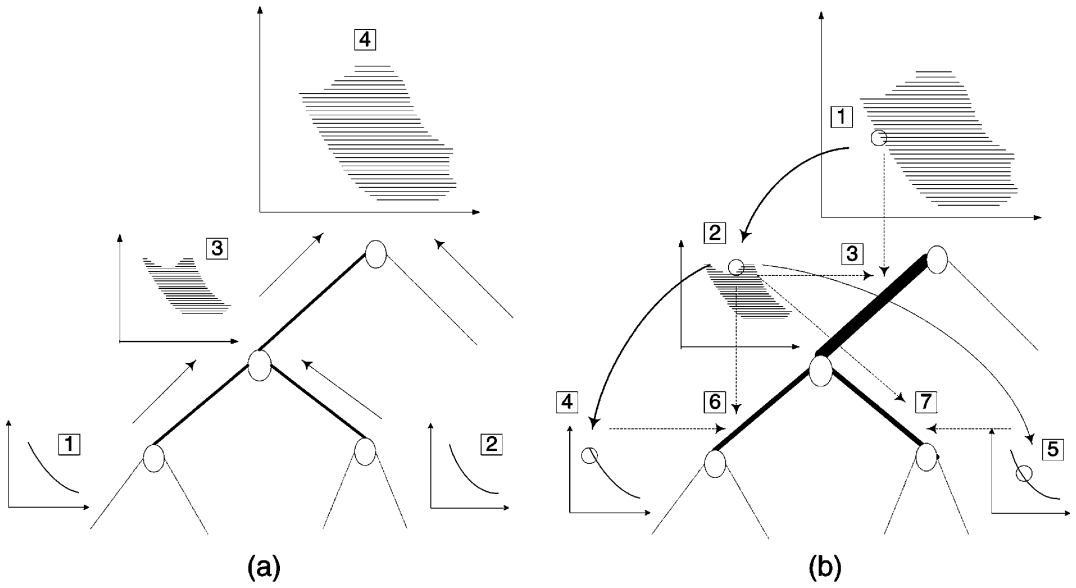

Fig. 2. Illustration of *ClockTune* during (a) the bottom-up phase and (b) the top-down phase.

levels of buffers, $\Omega_{\rm vn}$ is the projection of embeddings with odd levels of buffers, and $\Omega_v = \Omega_{\rm vp} \cup \Omega_{\rm vn}$. Property 1 still holds for the buffered case and (1) and (2) can be rewritten as

$$c_{\rm sv} = \sum_u c_{\rm su},\ u \in \{v_l, v_r\} \tag{3}$$

$$c_v = \sum_u \begin{cases} c_u + l_u w(e_u) c_0, & b_u = \phi \\ w(b_u) c_b + l_u w(e_u) c_0, & b_u \neq \phi \end{cases} \tag{4}$$

$$d_v = \begin{cases} d_u + \frac{l_u^2 r_0 c_0}{2} + \frac{l_u r_0 c_u}{w(e_u)}, & b_u = \phi \\ d_u + \frac{r_b c_u}{w(b_u)} + t_c + \frac{l_u^2 r_0 c_0}{2} + \frac{l_u r_0 w(b_u) c_b}{w(e_u)}, & b_u \neq \phi. \end{cases} \tag{5}$$

By Property 1, at least one set of buffering decisions, buffer widths, and wire widths satisfies (3)–(5) for a given set of p_v, p_{v_l}, and p_{v_r}. The feasible embeddings are actually implied in the DC regions, and we can avoid handling the growing design combinations by storing the DC regions instead. In the next section, we will show how to obtain the DC regions and select p_v, p_{v_l}, and p_{v_r}.

IV. ClockTune Algorithm

We propose a dynamic programming algorithm, *ClockTune*, to solve the min-ZSWS and min-ZSBWS problems. *ClockTune* is composed of two phases. In the first phase, a bottom-up approach is used to obtain the DC regions of all nodes. In the second phase, a top-down approach determines the buffer locations, buffer widths, and wire widths. Fig. 2 illustrates its procedures.

A. ZSWS Algorithm

In this section, we detail the bottom-up and top-down process of *ClockTune* in solving the min-ZSWS problem.

1) Bottom-Up Phase: Conceptually, a zero skew clock tree is formed by combining two *branches* with equal delay. A branch consists of a wire segment and a leaf node or a subtree connected to it. Thus, the left branch of node v is defined as $T_{v_l}^+ = \{e_{v_l} \cup T_{v_l}\}$ and the right branch is defined as $T_{v_r}^+ = \{e_{v_r} \cup T_{v_r}\}$. Let $d_{v_l}^+$ be the delay and $c_{v_l}^+$ be the total downstream capacitance seen at v along e_{v_l}, $T_v = \{T_{v_l}^+ \cup T_{v_r}^+ \mid d_{v_l}^+ = d_{v_r}^+\}$.

We first introduce the definition of the branch DC region and the associated $\curlywedge$ operator to facilitate our discussion followed by introducing the wire-sizing transformation of calculating the branch DC region.

Definition 1: Branch DC Region: The branch DC region of node v, $\Omega_v^+ = \{(d_v^+, c_v^+)\}$, is the projection of all embeddings of $T_v^+ = \{e_v \cup T_v\}$ on the *D–C plane*.

Definition 2: $\curlywedge$ Operator: The DC region of v is equivalent to the combination of the branch DC regions of v_l and v_r through the equidelay operator, $\curlywedge$, denoted as $\Omega_v = \Omega_{v_l}^+ \curlywedge \Omega_{v_r}^+$. The operator performs the following operation:

$$(d_v, c_v) \in \Omega_v \iff \exists\ (d_{v_l}^+, c_{v_l}^+) \in \Omega_{v_l}^+ \text{ and } (d_{v_r}^+, c_{v_r}^+) \in \Omega_{v_r}^+ \text{ s.t. } d_v = d_{v_l}^+ = d_{v_r}^+,\ c_v = c_{v_l}^+ + c_{v_r}^+.$$

Lemma 1: Wire-Sizing Transformation: Given $w_m \leq w(e_v) \leq w_M$, Ω_v^+ can be obtained from Ω_v by $\mathbb{W}_v$, denoted $\Omega_v^+ = \mathbb{W}_v(\Omega_v)$, which does the following transformation:

$$d_v^+ = d_v + \frac{l_v r_0}{w(e_v)} \left(\frac{l_v w(e_v) c_0}{2} + c_v \right) \tag{6}$$

$$c_v^+ = c_v + l_v w(e_v) c_0. \tag{7}$$

Algorithm 1 $ClockTune\_\mathrm{DC}(T_v)$ *of min-ZSWS*
Input: a clock tree T_v with given routing rooted at node v
Output: DC regions of all nodes in T_v
if v is a leaf node **then**
  $\Omega_v \leftarrow (d_v, c_v)$
else {v is an internal node}
  call $ClockTune\_\mathrm{DC}(T_{v_l})$
  call $ClockTune\_\mathrm{DC}(T_{v_r})$
  $\Omega_{v_l}^+ \leftarrow \mathbb{W}_{v_l}(\Omega_{v_l})$
  $\Omega_{v_r}^+ \leftarrow \mathbb{W}_{v_r}(\Omega_{v_r})$
  $\Omega_v \leftarrow \Omega_{v_l}^+ \curlywedge \Omega_{v_r}^+$
end if

The $ClockTune\_DC(T_v)$ subroutine of the *ClockTune* algorithm can now be written as Algorithm 1. For a leaf node v, c_v is the load capacitance and, hence, a constant. To enforce the zero-skew constraint,

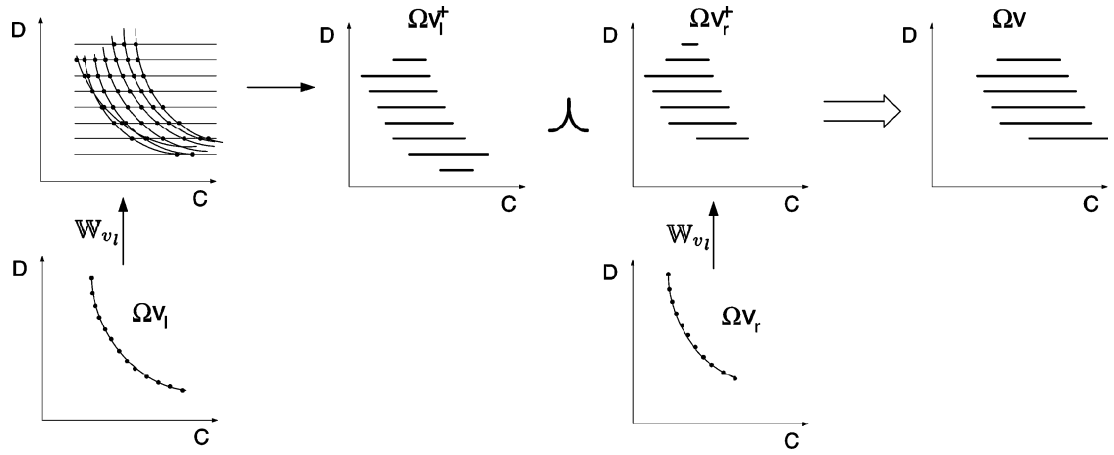

Fig. 3. Obtain the DC region of a level-2 node by sampling techniques.

we set d_v to 0 for all leaf nodes, and $\Omega_v = \{(d_v, c_v)\}$ is a single point on the D–C plane. Although our focus is on the min-ZSWS problem, *ClockTune* can accept arbitrary skew values by simply assigning different d_v for each leaf nodes. *ClockTune* can also be extended to accept bounded-skew constraints, where the Ω_v of a leaf node becomes a vertical segment, of which the Y coordinates of the end points are the maximum and minimum acceptable clock arrival times.

For a level-1 node v, the closed-form solution of Ω_v can be obtained by solving (6) and (7) for v_l, v_r, and imposing the zero-skew constraint $d_{v_l}^+ = d_{v_r}^+$. The solution is as follows:

$$d_v\Big(w(e_{v_l})\Big) = d_{v_l} + \frac{l_{v_l}^2 r_0 c_0}{2} + \frac{l_{v_l} r_0 c_{v_l}}{w(e_{v_l})} \tag{8}$$

$$c_v\Big(w(e_{v_l})\Big) = c_{v_l} + c_{v_r} + l_{v_l} w(e_{v_l}) c_0 + \frac{l_{v_r}^2 c_{v_r} r_0 c_0}{(d_{v_l} - d_{v_r}) + \frac{r_0 c_0}{2}(l_{v_l}^2 - l_{v_r}^2) + \frac{l_{v_l} r_0 c_{v_l}}{w(e_{v_l})}}. \tag{9}$$

Equations (8) and (9) represent a strictly decreasing curve on the D–C *plane*. The last term in (9) is the wire capacitance of e_{v_r} in which $w(e_{v_r})$ is further substituted by its relation with $w(e_{v_l})$. *ClockTune* then only needs to store the feasible range of $w(e_{v_l})$ to represent this curve [14]. However, the closed-form solution of Ω_v for a level-2 node v is difficult to obtain. Sampling techniques are applied to sample and store $\Omega_{v_l}^+$ and $\Omega_{v_r}^+$, which are then combined into Ω_v. We first take p samples on the delay range $d_{v_l}^+ \cap d_{v_r}^+$, then take q samples for $w(e_u)$ (assuming u is the level-1 child of v). For each sample of $w(e_u)$, (8) and (9) give a single point, and a subset of Ω_u^+, that is also a strictly decreasing curve, can be obtained. The intersection points of these q curves and p delay samples can be calculated, and the ranges those points span can be captured. By taking the same p delay samples on the other child node, $\Omega_v = \Omega_{v_l}^+ \curlywedge \Omega_{v_r}^+$ can be obtained. The procedure is illustrated in Fig. 3. In a sampled DC region, each delay sample is associated with one or more capacitance ranges. The branch DC region of each horizontal segment in a sampled DC region can be solved by (6) and (7) and, again, we perform sampling on the delay to obtain the sampled DC region for level-3 and above nodes [14].

2) Top-Down Phase: The top-down phase is straightforward. We first select a pair of target delay and capacitance load values (d_t, c_t) from Ω_v, which can be the minimum-delay or minimum-power solution. The capacitance load c_t is further divided into $c_{\rm tl}$ and $c_{\rm tr}$, such that $c_{\rm tl} + c_{\rm tr} = c_t$, $(d_t, c_{\rm tl}) \in \Omega_{v_l}^+$, and $(d_t, c_{\rm tr}) \in \Omega_{v_r}^+$. If v_l is a leaf node, then $w(e_{v_l})$ is determined by (2). If v_l is a level-1 node, the feasible range of $w(e_{v_l})$ can be obtained by solving (6). If v_l is a level-2 or above node, then the DC region of v_l is in a sampled form. For each sample in Ω_{v_l}, the range of $w(e_{v_l})$ can be obtained by solving (6) and at least one range of $w(e_{v_l})$ is feasible by Property 1. Once $w(e_{v_l})$ is chosen, the target delay and capacitance load of Ω_{v_l} are determined and we can proceed to determine the wire widths in T_{v_l}. The same approach applies to v_r. $ClockTune\_\mathrm{Embed}()$ is given in Algorithm 2.

Algorithm 2 $ClockTune\_\mathrm{Embed}(d_t, c_t, T_v)$ *of min-ZSWS*
Input: a clock tree T_v with given routing rooted at node v
Output: an embedding of T_v
if v is the root node **then**
  choose (d_t, c_t) from Ω_v
end if
split c_t into $c_{\rm lt}^+$ and $c_{\rm rt}^+$, such that $c_{\rm lt}^+ + c_{\rm rt}^+ = c_t$, $(d_t, c_{\rm lt}^+) \in \Omega_{v_l}^+$, and $(d_t, c_{\rm rt}^+) \in \Omega_{v_r}^+$
foreach child node $u \in \{v_l, v_r\}$
  switch u
   case leaf node
    solve $w(e_u)$ by **(2)**
   case level-1 node
     solve the range of $w(e_u)$ by **(6) and (7)**
     choose a $w(e_u)$ and calculate (d_u, c_u)
     call $ClockTune\_\mathrm{Embed}(d_u, c_u, T_u)$
   case level-2 or above node
    solve the range of $w(e_u)$ for every sample in Ω_u by **(6) and (7)**
     choose a $w(e_u)$ and calculate (d_u, c_u)
     call $ClockTune\_\mathrm{Embed}(d_u, c_u, T_u)$
  end switch
end for

B. Min-ZSBWS Algorithm

Inverter insertion has proven to be more area efficient than buffer insertion [17] and *ClockTune* is extended to handle inverter insertion and its signal polarity issue. From this point on in this paper, the term buffer refers to inverter. In the min-ZSBWS problem, Ω_v is split into $\Omega_{\rm vp}$ and $\Omega_{\rm vn}$. When applying $\mathbb{W}_v$, the polarities of the DC regions remain unchanged. If a buffer is inserted, the total capacitance increases, but the capacitance seen by upstream nodes is reduced. Thus, we need to expand the D–C *plane* into the D–C *space* where d_v is on the Y

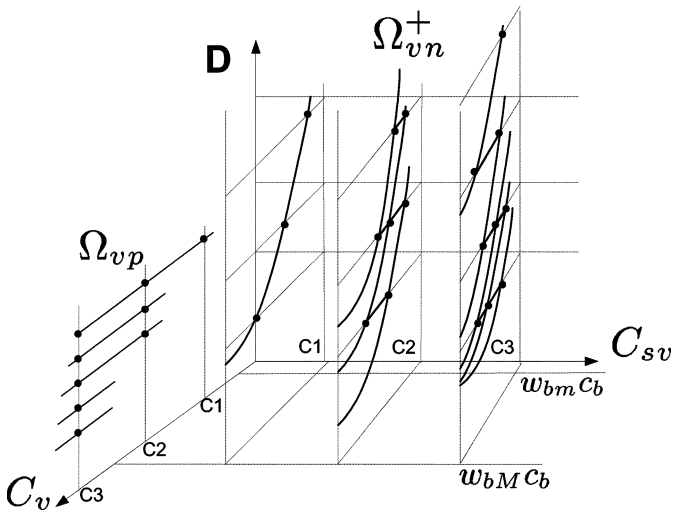

Fig. 4. Illustration of the procedure to generate the branch DC region above a buffered node.

axis, $c_{\rm sv}$ is on the X axis, and c_v is on the Z axis. We now define another transformation for simultaneous buffer insertion/sizing and wire sizing.

Lemma 2: Buffer-Insertion/Sizing and Wire-Sizing Transformation: Given $w_m \le w(e_v) \le w_M$ and $w_{\rm bm} \le w(b_v) \le w_{\rm bM}$, $\Omega^+_{\rm vn}$ with buffer inserted above v can be obtained from $\Omega_{\rm vp}$ by $\mathbb{B}_v$, denoted $\Omega^+_{\rm vn} = \mathbb{B}_v(\Omega_{vp})$, which performs the following transformation:

$$d^+_{\rm vn} = d_{\rm vp} + \frac{r_b c_{\rm vp}}{w(b_v)} + t_c + \frac{l_v^2 r_0 c_0}{2} + \frac{l_v r_0 c_b w(b_v)}{w(e_v)} \quad (10)$$

$$c^+_{\rm vn} = c_b w(b_v) + l_v w(e_v) c_0 \quad (11)$$

$$c^+_{\rm svn} = c_{\rm svp} + c_{\rm vp}, \quad p, n \text{ are interchangeable.} \quad (12)$$

To obtain the three-dimensional DC regions, sampling is first performed on the delay and shielded capacitance values along Y and X directions. To create the branch DC region above an unbuffered node, we need to sample $w(e_u)$ to fix $c_{\rm vp}$ values as before. To create the branch DC region above a buffered node we take samples on $w(b_v)$. However, the sampling originally required on $w(e_u)$ can be eliminated because, for each sample of $c^+_{\rm svn}$, the value of $c_{\rm vp} = c^+_{\rm svn} - c_{\rm svp}$ is fixed. The procedures are illustrated in Fig. 4 and Algorithm 3. The top-down algorithm follows the same procedures as in Algorithm 2 and presented in Algorithm 4. The major differences are that both $c_t = c_{\rm lt} + c_{\rm rt}$ and $c_{\rm st} = c_{\rm slt} + c_{\rm srt}$ have to be satisfied when choosing $p_{\rm vl}$ and $p_{\rm vr}$, and (10)–(12) can also be used to determine $w(e_v)$ and $w(b_v)$.

Algorithm 3 $ClockTune\_\mathrm{DC}(T_v)$ *of min-ZSBWS*
Input: a clock tree T_v with given routing rooted at node v
Output: DC regions of all nodes in T_v
if v is a leaf node **then**
  $\Omega_{\rm vp} \leftarrow (d_v, c_v, c_{sv})$, $\Omega_{\rm vn} \leftarrow \phi$
else {v is an internal node}
  call $ClockTune\_\mathrm{DC}(T_{v_l})$
  call $ClockTune\_\mathrm{DC}(T_{v_r})$
  $\Omega^+_{v_l p} \leftarrow \mathbb{W}_{v_l}(\Omega_{v_l p}) \cup \mathbb{B}_{v_l}(\Omega_{v_l n})$
  $\Omega^+_{v_l n} \leftarrow \mathbb{W}_{v_l}(\Omega_{v_l n}) \cup \mathbb{B}_{v_l}(\Omega_{v_l p})$
  $\Omega^+_{v_r p} \leftarrow \mathbb{W}_{v_r}(\Omega_{v_r p}) \cup \mathbb{B}_{v_r}(\Omega_{v_r n})$
  $\Omega^+_{v_r n} \leftarrow \mathbb{W}_{v_r}(\Omega_{v_r n}) \cup \mathbb{B}_{v_r}(\Omega_{v_r p})$
  $\Omega_v \leftarrow (\Omega^+_{v_l p} \curlywedge \Omega^+_{v_r p}) \cup (\Omega^+_{v_l n} \curlywedge \Omega^+_{v_r n})$
end if

Algorithm 4 $ClockTune\_\mathrm{Embed}(d_t, c_t, c_{\rm st}, T_v)$ *of min-ZSBWS*
Input: a clock tree T_v with given routing rooted at node v
Output: an embedding of T_v
if v is the root node **then**
  choose $(d_t, c_t, c_{\rm st})$ from $\Omega_{\rm vp}(\Omega_{\rm vn})$.
end if
split c_t into $c^+_{\rm lt}$ and $c^+_{\rm rt}$, $c_{\rm st}$ into $c^+_{\rm slt}$ and $c^+_{\rm srt}$, such that $c^+_{\rm lt}+c^+_{\rm rt} = c_t$, $c^+_{\rm slt}+c^+_{\rm srt} = c_{st}$, and $(d_t, c^+_{\rm lt}, c^+_{\rm slt}) \in \Omega^+_{v_l p}(\Omega^+_{v_l n})$, $(d_t, c^+_{\rm rt}, c^+_{\rm srt}) \in \Omega^+_{v_r p}(\Omega^+_{v_r n})$
foreach child node $u \in \{v_l, v_r\}$
  switch u
  case leaf node
    solve $w(e_u)$ by **(2)**
  case level-1 node
    solve the range of $w(e_u)$ by **(6)**
    choose a $w(e_u)$ and calculate (d_u, c_u)
    call $ClockTune\_\mathrm{Embed}(d_u, c_u, 0, T_u)$
  case level-2 or above node
    solve the range of $w(e_u)$ $(w(b_u) = 0)$ for every sample in $\Omega_{\rm up}$ $(\Omega_{\rm un})$ by **(6)**
    solve the range of $w(e_u)$ and $w(b_u)$ for every sample in $\Omega_{\rm un}$ $(\Omega_{\rm up})$ by **(10)** **(12)**
    choose a pair of $w(e_u)$ and $w(b_u)$ and calculate $(d_u, c_u, c_{\rm su})$
    call $ClockTune\_\mathrm{Embed}(d_u, c_u, c_{\rm su}, T_u)$
  end switch
end for

C. Slew-Rate Control

One of the purposes for buffer insertion is to adjust the clock slew rate. If the loading capacitance of a buffer is too large, the output signal will have a slow rise and fall time, and it in turn increases the short-circuit power of downstream buffers. One way to control the slew rate is to limit the loading capacitance to a certain value such that the slew rate of the buffer is bounded to the desired value. This constraint can be taken care of easily by limiting c_v during the bottom-up phase. In this manner, it is guaranteed that the embeddings we get during the top-down phase will not have any buffer driving a load that exceeds the predefined upper limit. During the bottom-up phase, the DC regions might grow very large due to the embeddings with excessive buffers, which have large delay and total capacitance values. Again, we can set upper limits on d_v and $(c_{\rm sv} + c_v)$. Since c_v has been limited by the maximum buffer loading value, which is usually small, imposing the limit on $c_{\rm sv}$ is sufficient. These limits are equivalent to adding three cutting planes in the *D–C space* and only consider the DC regions that lie inside the cuboid on the first octant.

D. Incremental Refinement

When clock routing undergoes design changes and the clock tree is no longer zero skew, *ClockTune* can be used to perform incremental refinement in the way that follows. First, the DC regions are reconstructed from affected nodes until it reaches node v such that T_v covers all design changes. Assume the projection of the original embedding of T_v is $(\hat{d}_v, \hat{c}_v, \hat{c}_{\rm sv})$. If there exists a point in the new DC region with $d_v = \hat{d}_v$, $c_v = \hat{c}_v$, and $c_{\rm sv} = \tilde{c}_{\rm sv}$, we take this point and run $ClockTune\_\mathrm{Embed}(\hat{d}_v, \hat{c}_v, \tilde{c}_{sv})$ to determine a new buffering and wire sizing of T_v. The rest of the clock tree is not aware of these design changes because $(\hat{d}_v, \hat{c}_v)$ exposed to the rest of the clock tree remains the same. Otherwise, we keep updating the DC regions toward the root

TABLE II
DELAY AND POWER BEFORE AND AFTER WIRE-SIZING. THE GAINS ARE MEASURED BY THE INITIAL VALUES DIVIDED BY THE OPTIMIZED VALUES

Input	Initial($w = 1\mu m$)		*ClockTune*($w_m = 0.3\mu m, w_M = 3\mu m, p = q = 256$)								
	delay	load	minimum-delay solution				minimum-power solution				CPU
(#nodes)	(ns)	(pF)	Delay	Gain	Load	Gain	Delay	Gain	Load	Gain	(min.)
r1(267)	1.097	45.2	0.306	3.59	34.5	1.31	1.905	0.58	24.7	1.83	0.34
r2(598)	3.210	93.6	0.762	4.21	67.7	1.38	5.195	0.62	53.0	1.76	0.83
r3(862)	4.590	126.7	1.152	3.99	92.5	1.37	7.838	0.59	73.6	1.72	1.28
r4(1903)	13.184	266.2	3.288	4.01	184.3	1.44	24.802	0.53	156.8	1.70	2.61
r5(3101)	24.883	413.0	6.093	4.08	286.6	1.44	47.958	0.52	248.1	1.66	4.35

TABLE III
DELAY AND POWER BEFORE AND AFTER BUFFER-INSERTION/SIZING AND WIRE-SIZING

Input	Initial($w = 1\mu m$)		*ClockTune*($w_m = 0.3\mu m, w_M = 3\mu m, w_{bm} = 1, w_{bM} = 10, p = q = r = 64$)										
	delay	load	minimum-delay solution					minimum-power solution					CPU
	(ns)	(pF)	Delay	Gain	Load	Gain	Buffers	Delay	Gain	Load	Gain	Buffers	(min.)
r1(267)	1.097	45.2	0.145	7.57	36.6	1.23	34	0.500	2.20	29.8	1.52	25	1.4
r2(598)	3.210	93.6	0.218	14.72	78.9	1.19	61	0.818	3.93	65.0	1.44	67	3.2
r3(862)	4.590	126.7	0.236	19.43	107.9	1.17	106	0.563	8.15	84.6	1.50	64	3.6
r4(1903)	13.184	266.2	0.454	29.02	198.6	1.34	116	1.199	10.99	189.5	1.40	115	10.5
r5(3101)	24.883	413.0	0.545	45.65	331.5	1.25	384	0.999	24.90	317.6	1.30	280	16.1

node until a point of the new DC region satisfies $d_v = \hat{d}_v$ and $c_v = \hat{c}_v$. For example, if the number of leaf nodes or total sink capacitance of a function block increases, we can insert more levels of buffers in T_v to maintain its delay. Adding more buffers will increase the power consumption of T_v. However, only the total downstream capacitance seen at v needs to remain unchanged, and the amount of shielded capacitance can be increased. If, on the contrary, the total sink capacitance of T_v decreases, we can use wider wires or fewer levels of buffers so that d_v and c_v remain unchanged. Since the locations of first-level buffers and wire widths above these buffers can be adjusted, a wide range of local changes can be accommodated through this tuning process.

To enable clock tuning during design cycles, the subtree that are likely to undergo design changes may not be designed at optimal delay. For example, if T_v is designed to have optimal delay and its total sink capacitance decreases, *ClockTune* can be used to find a new embedding with the same delay, where the original delay becomes suboptimal in the new DC region. However, if the total sink capacitance increases, it will not be possible to maintain the delay of T_v. Thus, designers have to trade-off between design flexibility and clock delay.

V. COMPLEXITY

Assuming a clock tree T_v has n nodes, the number of delay samples is p, and the number of wire-width samples is q. In the min-ZSWS problem, it takes $O(1)$ time to construct the DC regions for leaf and level-1 nodes. Level-2 nodes require $O(pq)$ time due to delay and wire-width sampling. The other nodes need $O(p^2)$ time to combine p range for each of the p delay samples. Note that a level-2 node can have more than one capacitance ranges with each delay. However, the gaps between the ranges tend to be filled up quickly as we move upward toward the root node. For example, multiple ranges can overlap and become a single range when we create the branch DC regions or merge the branch DC regions with the $\curlywedge$ operator. In practice, the number of ranges with each delay is always less than four and we exclude it in the complexity analyses. Thus, the complexity for the bottom-up phase is $O(\max(p, q)pn)$. In the top-down phase, each wire width can be determined in $O(p)$ time and the complexity is $O(pn)$. The overall runtime complexity is $O(\max(p, q)pn)$. Since we only need to store the maximum and minimum values of the capacitance load of each delay sample, the memory requirement is $O(pn)$.

In the min-ZSBWS problem, the complexity to construct the DC regions for leaf and level-1 nodes is $O(1)$. Let p be the number of delay samples, let q be the number of wire-width samples for wire-sizing transformation and the number of buffer-width samples for buffer insertion/sizing and wire-sizing transformation, and let r be the number

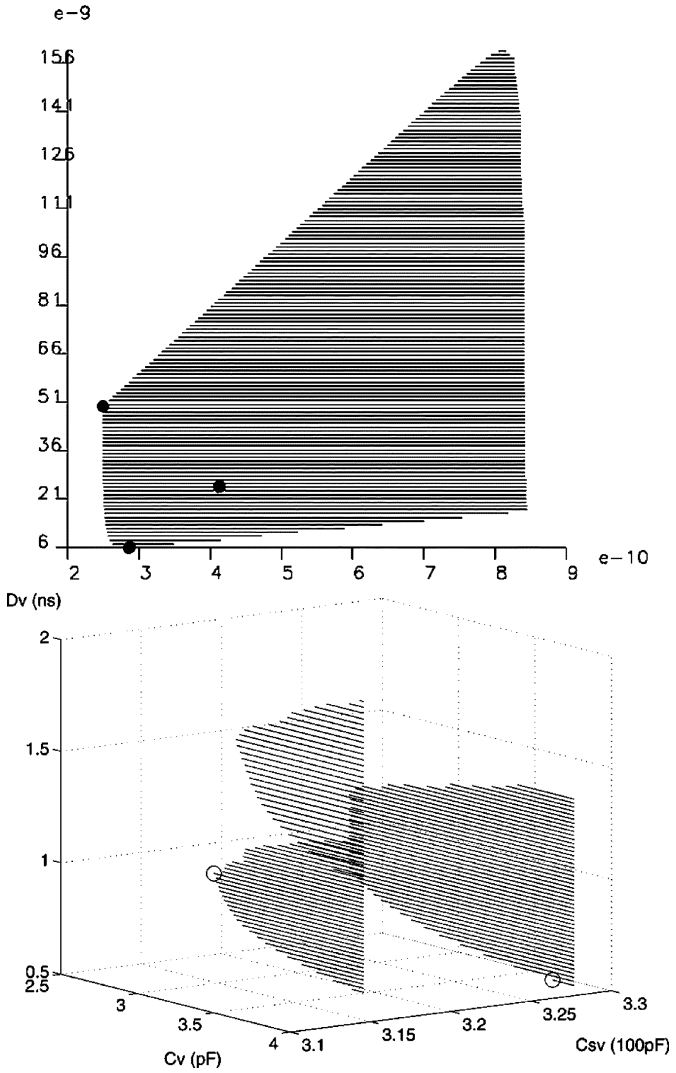

Fig. 5. DC regions of the root node of $r5$ in (top) min-ZSWS and (bottom) min-ZSBWS problems. The circles indicate the minimum-delay and minimum-power solutions.

TABLE IV
COMPARISON BETWEEN DISCRETIZATION-INDUCED SKEW AND PROCESS-VARIATION-INDUCED SKEW

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

of shield-capacitance samples. The transformations perform q sampling with each of the $p \cdot r$ line samples in the children DC regions in order to define one of the $p \cdot r$ line samples in the current DC region. Thus, the straightforward implementation requires $O(p^2qr^2n)$ runtime. By exploring the properties of (10) and (11), the runtime can be reduced to $\sim O(pqr^2n)$, and the memory requirement is $O(prn)$.

VI. EXPERIMENTAL RESULTS

We implement our algorithm in C++ and run the program on a 1-GB 1.2-GHz Pentium IV PC. The benchmarks $r1$-$r5$ are taken from [18]. All simulations use $r_0 = 0.03$, $c_0 = 2 \times 10^{-16}/\mu\text{m}^2$, $w_m = 0.3$ μm, $w_M = 3$ μm. The parameters of the buffers are $c_b = 40$ fF, $r_b = 100\,\Omega$, $t_c = 30$ ps, and $w_{\text{bm}} = 1$, $w_{\text{bM}} = 10$. The maximum load of a buffer is 4 pF. The initial routings are generated by the BB+DME [13] algorithm. The numbers of samples used in the min-ZSWS problem are $p = q = 256$. The numbers of samples used in the min-ZSBWS problem are $p = q = r = 64$.

Table II shows the minimum-delay and minimum-power solutions for the min-ZSWS problem. If the initial routing does not use the minimum wire width, then both the delay and power can be reduced by performing wire sizing. Table III shows the minimum-delay and minimum-power solutions for the min-ZSBWS problem. The delay is dramatically lower than that of the initial routing even for the minimum-power solution, and the minimum-delay solutions have more than $2\times$ speedup compared to the minimum-power solution. However, the power saving from moving minimum-delay solutions to minimum-power solutions is less than 5% for $r5$. Since the process-variation-induced skew is roughly proportional to the clock delay, it is not worthwhile to go for minimum-power solutions. As shown in Table III, the power consumptions of initial solutions and minimum-delay solutions are all roughly proportional to the sizes of the clock trees. Therefore, buffer insertion/sizing and wire-sizing techniques cannot alleviate the linear growth of the power consumption for large clock trees. Thus, clock gating or other design techniques need to be investigated for low-power applications. We also use different initial wire widths to generate different initial routings and the solutions found by *ClockTune* do not change much because most of the delay reductions come from buffer insertion/sizing. Using smaller initial widths results in higher initial delay and lower initial load, thus, the delay gains become higher and load gains are lower than those listed in Tables II and III (and *vice versa*). Note that the delays shown in the figures and tables are the Elmore delays multiplied by $ln2$. Fig. 5 shows the DC regions of the root node in $r5$ for the min-ZSWS and min-ZSBWS problems.

In industrial applications, wire and buffer widths usually take discrete values. We can discretize the widths to make the embeddings generated by *ClockTune* comply with layout restrictions. After discretization, the embeddings are no longer zero skew. Fortunately, discretization introduces random variations to the clock tree and their effects tend to cancel each other out. Process variation is usually systematic and affects buffer channel widths as well. Thus, discretization-induced skew is much less significant than process-variation-induced skew and we can obtain near-zero-skew embeddings from *ClockTune*. Table IV shows the discretization-induced skews and process-variation-induced skews of minimum-delay embeddings from Tables II and III. Upon discretization, all wire and buffer widths are rounded to the nearest multiples of unit widths ΔW. For process variation, we use a simple linear model, such that the variations on all wire widths, buffer widths, and buffer channel widths increase linearly across the whole chip with maximum variation $\Delta W_{\max}$. Results show that discretization-induced skew is within tolerable range and *ClockTune* is suitable for industrial applications.

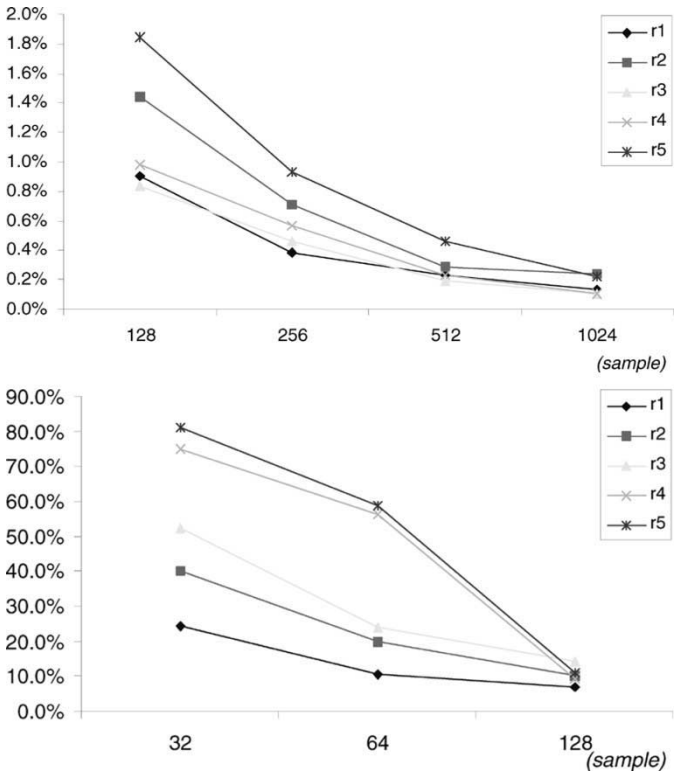

Fig. 6. Relative distances from minimum delays obtained by *ClockTune* to optimal delays in (top) min-ZSWS and (bottom) min-ZSBWS problems. Optimal delays are approximated by nonlinear curve fitting.

Theoretically, *ClockTune* requires infinite samples in order for the minimum-delay solutions to converge to optimal delay. Since the runtime complexity of *ClockTune* is polynomial, the convergence rate affects the scalability of *ClockTune*. Fig. 6 shows the relative distances from minimum delays obtained by *ClockTune* to optimal delays in which optimal delays are approximated by nonlinear curve fitting. The results show that it takes reasonable samples in finding good solutions. If we fix the number of samples, the runtime is linear with respect to the size of the clock tree. Thus, *ClockTune* scales well for large clock trees.

VII. Conclusion

We present a simultaneous buffer insertion/sizing and wire sizing zero skew clock-tree optimization algorithm, *ClockTune*. The algorithm takes polynomial runtime and memory usage and finds minimum-delay and minimum-power embeddings efficiently. For wire widths from 0.3 to 3 μm and buffer widths from $1\times$ to $10\times$, the algorithm achieves $45\times$ delay improvement and $1.25\times$ power-saving over $r5$'s initial routing with 1 μm wires generated by the BB+DME algorithm. *ClockTune* can also be applied for clock tuning to speedup design convergence.

References

[1] ClockTune [Online]. Available: http://vlsi.ece.wisc.edu/Tools.htm

[2] J. G. Xi and W. W.-M Dai, "Useful-skew clock routing with gate sizing for low power design," in *Proc. 33rd Annu. Design Automation Conf.*, 1996, pp. 383–388.

[3] J. Cong, Z. Pan, L. He, C.-K. Koh, and K.-Y. Khoo, "Interconnect design for deep submicron ICs," in *Proc. 1997 IEEE/ACM Int. Conf. Computer-Aided Design*, 1997, pp. 478–485.

[4] X. Zeng, D. Zhou, and W. Li, "Buffer insertion for clock delay and skew minimization," in *Proc. 1999 Int. Symp. Physical Design*, 1999, pp. 36–41.

[5] S. S. Sapatnekar, "RC interconnect optimization under the elmore delay model," in *Proc. 31st Annu. Design Automation Conf.*, 1994, pp. 387–391.

[6] R. Kay, G. Bucheuv, and L. T. Pileggi, "EWA: Exact wiring-sizing algorithm," in *Proc. Int. Symp. Physical Design*, 1997, pp. 178–185.

[7] J. Cong, C. Koh, and K. Leung, "Simultaneous buffer and wire sizing for performance and power optimization," in *Proc. Int. Symp. Low Power Electron. Design*, 1996, pp. 271–276.

[8] C.-P Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1998, pp. 617–624.

[9] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1996, pp. 44–49.

[10] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion with accurate gate and interconnect delay computation," in *Proc. 36th ACM/IEEE Design Automation Conf.*, 1999, pp. 479–484.

[11] L. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 865–868.

[12] I.-M Liu, T.-L Chou, A. Aziz, and D. F. Wong, "Zero-skew clock tree construction by simultaneous routing, wire sizing, and buffer insertion," in *Proc. Int. Symp. Physical Design*, 2000, pp. 33–38.

[13] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, and A. Kahng, "Zero skew clock routing with minimum wirelength," *IEEE Trans. Circuits Syst. II*, vol. 39, pp. 799–814, Nov. 1992.

[14] J.-L. Tsai, T.-H. Chen, and C. C.-P. Chen, "Epsilon-optimal minimum-delay/area zero-skew clock-tree wire-sizing in pseudo-polynomial time," in *Proc. Int. Symp. Physical Design*, 2003, pp. 166–173.

[15] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage, "Skew and delay optimization for reliable buffered clock trees," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1993, pp. 556–562.

[16] P. R. O'Brien and T. L. Savarino, "Modeling the driving-point characteristic of resistive interconnect for accurate delay estimation," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1989, pp. 512–515.

[17] X. Tang, R. Tian, H. Xiang, and D. F. Wong, "A new algorithm for routing tree construction with buffer insertion and wire sizing under obstacle constraints," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2001, pp. 49–56.

[18] R.-S. Tsay, "Exact zero skew," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1991, pp. 336–339.

Constrained Floorplanning Using Network Flows

Yan Feng, Dinesh P. Mehta, and Hannah Yang

***Abstract*—This paper presents algorithms for a constrained version of the "modern" floorplanning problem proposed by Kahng in "Classical Floorplanning Harmful?" (Kahng, 2000). Specifically, the constrained modern floorplanning problem (CMFP) is suitable when die-size is fixed, modules are permitted to have rectilinear shapes and, in addition, the approximate relative positions of the modules are known. This formulation is particularly useful in two scenarios: 1) assisting an expert floorplan architect in a semiautomated floorplan methodology and 2) in incremental floorplanning. CMFP is shown to be negative–positive hard. An algorithm based on a max-flow network formulation quickly identifies input constraints that are impossible to meet, thus permitting the floorplan architect to modify these constraints. Three algorithms [Breadth First Search (BFS), Improved BFS (IBFS), Compromise BFS (CBFS)] based on using BFS numbers to assign costs in a min-cost max-flow network formulation are presented. Experiments on standard benchmarks demonstrate that IBFS is fast and effective in practice.**

***Index Terms*—Algorithms, design automation, flow graphs.**

I. Introduction

In classical floorplanning, the input consists of a set of (typically rectangular) modules. A set of realizations providing height and width information is associated with each module. In addition, a connectivity matrix that contains the number of interconnections between pairs of modules is provided. The objective is to minimize some combination of the area, estimated wire length, and other criteria that have emerged recently such as critical-path wire length, length of parallel-running wires, clock skew, etc. Much research in floorplanning is concerned with finding a good representation that can be used efficiently within the context of simulated annealing [2]–[9].

Kahng [1] critiques the classical floorplanning problem and proposes a modern formulation that is more consistent with the needs of current design methodologies. Some of the attributes of the modern formulation are: 1) the dimensions of the bounding rectangle must be fixed because floorplanning is carried out after the die size and the package have been chosen in most design methodologies; 2) the modules' shapes should not be restricted to rectangles, L-shapes, and T-shapes; and 3) "round" blocks with an aspect ratio near 1 are desirable.

Several aspects of this problem had been previously addressed by Mehta and Sherwani [10]. Their algorithm assumes a fixed outline and obtains a provable zero whitespace solution by relaxing the requirement on module shapes. Further, it also tries to make blocks as "round" as possible and to minimize the number of sides. Their methodology differs from that proposed by Kahng in that they assume that an approximate location for each module was included in the input. This is a realistic formulation in several design scenarios where the designer already has a fairly good idea as to the approximate locations of the modules. This claim is supported by an excerpt reproduced from a discussion among designers in an electrical design automation (EDA) newsgroup:

Manuscript received May 29, 2003; revised September 19, 2003. This work was supported by the National Science Foundation under Grant CCR-9988338. This paper was recommended by Guest Editor C. J. Alpert.

Y. Feng and D. P. Mehta are with the Department of Mathematical and Computer Science, Colorado School of Mines, Golden, CO 80401 USA (email: yfeng@mines.edu; e-mail:dmehta@mines.edu).

H. Yang is with the Strategic Computer-Aided Design Labs, Intel, Hillsboro, OR 97124 USA (e-mail:hyang@ichips.intel.com).

Digital Object Identifier 10.1109/TCAD.2004.825877