This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Optimal Generalized H-Tree Topology and Buffering for High-Performance and Low-Power Clock Distribution

Kwangsoo Han Andrew B. Kahng Jiajia Li

***Abstract*—Clock power, skew and maximum latency are three key metrics for clock distribution in low-power and high-performance designs. An H-tree offers minimum clock skew and good robustness against variations, but at the cost of large wirelength and clock power. On the other hand, a "fishbone" clock network with spine-ribs structures has smaller wirelength, latency and clock power, but larger skew, as compared to an H-tree. No previous work enables systematic exploration of the regime between H-tree and spine to achieve an optimal tradeoff among clock power, skew and latency. In this work, we study the concept of a *generalized H-tree* – a topologically balanced tree with an arbitrary sequence of branching factors – and propose a dynamic programming-based method to determine *optimal* clock power, skew and latency, in the space of generalized H-tree solutions. Our method co-optimizes clock tree topology and buffering along branches according to fitted electrical models. We further propose a balanced K-means clustering and a linear programming-guided buffer placement approach to embed the generalized H-tree with respect to a given sink placement. We validate our solutions in commercial clock tree synthesis tool flows, in a commercial foundry's 28LP technology. The results show up to 30% clock power reduction while achieving similar skew and maximum latency as CTS solutions from recent versions of leading commercial place-and-route tools. Our proposed approach also achieves up to 56% clock power reduction while achieving similar skew and maximum latency as compared to CTS solutions from a state-of-the-art academic tool.**

I. Introduction

Physical implementation of clock distribution networks is increasingly critical to the success of high-performance, low-power IC product designs. Clock distribution takes up substantial routing and buffering resources as well as a significant portion of overall power consumption [25]. Power dissipation in the clock network has often been estimated to be one third of total IC power dissipation [21], or even half the total power in some designs. Further, the quality (skew and latency) of clock delivery strongly determines achievable performance of the design, particularly in advanced nodes. Skew is well known to affect datapath area and power, as well as the design schedule needed to achieve timing closure [10] [26] [29] [34] [36] [41]. Maximum clock latency is another key metric of the clock distribution network in advanced nodes since skews are magnified by on-chip variation (OCV) deratings [5].

K. Han, A. B. Kahng and J. Li are with the University of California at San Diego, La Jolla, CA 92093. E-mail: {kwhan, abk, jil150}@ucsd.edu

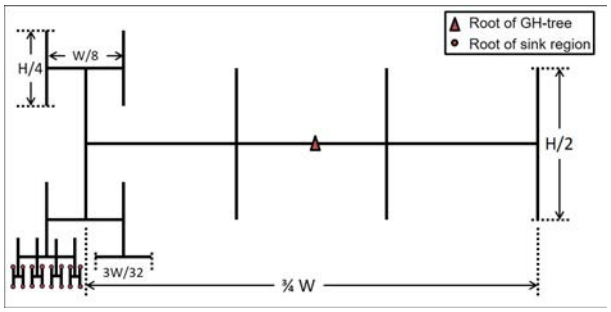

Fig. 1: 8-level GH-tree with branching pattern (4, 2, 2, 2, 4, 2, 2, 2).

As reviewed below, there are several methods of clock distribution. Tree-based constructions are still dominant, and remain the default of commercial clock tree synthesis (CTS) tools. To reduce skew and increase robustness (e.g., in light of manufacturing variability or reliability mechanisms), mesh and other non-tree topologies (e.g., trees + cross-link insertion) have been used. Such non-tree methods typically have large overheads in terms of power, area, wirelength and signoff analysis complexity. Thus, clock trees are still of interest and great practical relevance for reasons of cost efficiency, flexibility and design flow complexity. Increasingly, *structured* approaches to clock tree design are seen in practice, since these offer benefits of predictability in the resource-versus-skew tradeoff, particularly in the upper levels of clock trees.

As a special case of structured clock trees, the highly regular, recursive *H-tree* embedding of a complete binary tree [4] offers minimum skew, but at the cost of larger wirelength and potentially larger clock power and latency. "Fishbone" clock tree topologies (e.g., [2]) with spines and ribs can be more cost-efficient in terms of latency, wirelength, area and clock power – but incur varying propagation delays (that is, of the clock signal to branching points along a given spine) which cause skew. To explore the tradeoff among skew, latency and (clock power, clock buffer area) cost, this work proposes the concept of a *generalized H-tree* (GH-tree), which is a balanced tree topology with arbitrary branching factor at each level. (Like the H-tree, the GH-tree is a multi-level topology; like the fishbone, it can have branching factor greater than two.) Figure 1 shows a GH-tree with depth $P = 8$ and branching factors (4, 2, 2, 2, 4, 2, 2, 2) at levels $p = 1, 2, ..., 8$. In the example, we assume there are 1024 (= $4 \cdot 2 \cdot 2 \cdot 2 \cdot 4 \cdot 2 \cdot 2 \cdot 2$) nodes (sinks) uniformly placed in the region. If the root of the tree is at the region center, each root-to-leaf path will

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

contain horizontal segments of lengths $\frac{3 \cdot W}{4}$, $\frac{W}{8}$, $\frac{3 \cdot W}{32}$ and $\frac{W}{64}$, in that order. The lengths of the successive vertical segments in each root-leaf path are $\frac{H}{2}$, $\frac{H}{4}$, $\frac{H}{8}$ and $\frac{H}{16}$. We note that in our proposed GH-tree, we do not necessarily insert a buffer at each branching point. Further, we allow buffering that is internal to a given branch, that is, at any location along wiring of that given branch.

In this work, we study potential benefits of the generalized H-tree for low-power, low-skew, and low-latency clock distribution. We propose a dynamic programming (DP) algorithm that efficiently finds an *optimal*[1] GH-tree with minimum clock power for given latency and skew targets. This optimization uses calibrated clock buffer library and interconnect timing and power models, and co-optimizes the clock tree topology along with the buffering along branches. Furthermore, we also propose a clustering and linear programming-based heuristic to *embed* the GH-tree with respect to a given placement of clock sinks. Finally, our embedding of the GH-tree is blockage-aware. In a 28LP testbed with multi-corner timing constraints, our embedded GH-tree solutions provide significant clock power benefits (iso-skew and -latency) in comparison to commercial CTS solutions from the place-and-route tools of two leading EDA vendors as well as a state-of-the-art academic CTS tool. Our contributions are summarized as follows.

- We propose the concept of a *generalized H-tree*, which is a balanced tree topology that can have an arbitrary sequence of branching factors.
- We propose a DP-based method to co-optimize clock tree topology and buffering to achieve an optimal GH-tree solution with respect to the tradeoffs among skew, latency and clock power.
- We propose a *balanced K-means clustering* and linear programming-based buffer placement to embed our GH-tree solution with respect to any given sink placement.
- We validate our GH-tree optimization based on sink placements from a leading commercial place-and-route (P&R) tool, which include testcases with high floorplan aspect ratio and existence of blockages.
- Our methodology and optimizations can easily be integrated with commercial P&R tools. Our experimental results in a foundry 28LP technology with multi-corner testcases show up to 30% clock power reductions compared to current CTS tool solutions from two leading EDA vendors and up to 56% clock power reduction compared to a state-of-the-art academic solution [39].

II. Background and Motivation

In this section, we first review previous works on clock distribution, categorizing them as: (i) non-tree methods, and (ii) tree-based methods. We then provide a motivating analysis of the GH-tree solution space.

Non-tree methods. Mesh topologies are commonly understood to provide robustness and small skew. Many works, such as [14] [15] [30] [31], propose clock mesh designs for high-performance circuits that require robust clock networks. To reduce the cost (e.g., wirelength, power), hybrid clock distribution methods integrating both mesh and tree topologies have been proposed [29] [36]. Several works [12] [17] [26] [32] propose to insert cross-links for minimization of clock skew, based on an initial clock tree solution, with small power overhead. Rajaram et al. [26] propose an algorithm to recursively merge subtrees with backward slew propagation. Ewetz and Koh [12] propose systematic cross-link insertion methods to improve the robustness of a clock tree while minimizing its overheads. They propose a vertex reduction method to reduce the amount of redundancy in their non-tree structures. Although non-tree methods reduce clock skew and enhance robustness of clock networks, their intrinsic redundancy incurs additional cost (e.g., wirelength, power, effort of verification) as compared to tree-based methods. Furthermore, non-tree clock distribution topologies such as meshes lack flexibility and tunability; this can block, e.g., useful skew optimizations.

Dolev et al. [11] propose a hexagonal grid-based clock topology (HEX), consisting of a hexagonal grid with intermediate nodes that control the clock signals in the grid and supply the clock signals to nearby functional units. Abdelhadi et al. [1] propose an algorithm to construct a variation-tolerant hybrid clock network based on a combination of non-uniform meshes and unbuffered trees. Their method selectively reduces clock skew variations on critical timing paths. Zhou et al. [42] propose an algorithm to determine tapping points for local buffers that drive a clock mesh with non-uniform load distribution in a tree-driven grid clock network. Their algorithm first calculates load for each node, then clusters the nodes. Tapping points are determined for each cluster based on the minimum and maximum latencies.

Recently, Y. Kim and T. Kim [39] have proposed a synthesis algorithm for clock spine networks that effectively optimizes the tradeoff between clock resource and variation tolerance. The key idea of their algorithm is to treat the clock spine allocation and placement problem as a slicing floorplan optimization problem. Clock tree solutions from the CTS algorithm of [39] are compared to our GH-tree solutions in Section IV.

Tree-based methods. Due to their cost efficiency, clock tree-based methods have been commonly used for clock distribution in low-power designs. Early works [9] [6] [19] [38] propose clock tree constructions based on linear or Elmore delay models to minimize wirelength for a given skew target. However, delay and power impacts of buffers are ignored in these works. Approaches in [8] [13] [20] [23] [27] [28] [40] [41] comprehend buffering impact and co-optimize clock tree construction (i.e., tree topology) with buffering. Vittal and Marek-Sadowska [41] give an early algorithm that co-optimizes tree topology and buffer insertion. Mehta et al. [23] propose a clustering algorithm to obtain approximately load-balanced clusters and construct clock trees so as to minimize skew. These previous approaches typically construct the clock tree in a bottom-up way with a greedy algorithm,

[1]We note that our claim of an *optimal* clock tree solution is in the regime of generalized H-tree solutions (i.e., the continuum between H-tree and spine). We do not claim that our optimal GH-tree is a globally optimal clock tree solution.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

and do not explore the skew vs. cost tradeoff. Furthermore, few works adapt their tree construction approaches to, and validate their solution quality with, commercial P&R tools and realistic design blocks. Other works [5] [16] are ECO-based incremental optimizations based on an initial clock tree solution generated by commercial P&R tools. The objective functions of these works differ from ours. Chan et al. [5] minimize skew at the top-level, whereas Han et al. [16] minimize skew variation across corners.

A Motivating Analysis. To motivate our main studies below, we briefly illustrate the tradeoffs among clock tree wirelength, global skew and maximum clock latency seen across GH-tree topologies with various branching patterns. Before doing so, we summarize in Table I the terminology and notation used in the remaining discussion.

Given layout region area $W \times H$, we analyze the wirelength of a GH-tree with branching pattern (b_1, b_2, b_3, ..., b_P). We assume that (i) at any level, the region area is uniformly split into sink regions (i.e., regions that contain downstream sinks) according to the branching factor; (ii) the root of a sink region is located at the center of the sink region; (iii) branching factor b_p at any level p is always an even number; and (iv) the GH-tree always starts with a horizontal segment at the top level. Based on these assumptions, the wirelength of a horizontal (vertical) wire segment w_p (h_p) at level p is calculated as[2]

$$w_p = \frac{b_p - 1}{\Pi_{i=1}^{(p+1)/2} b_{2i-1}} \cdot W \ , \qquad h_p = \frac{b_p - 1}{\Pi_{i=1}^{p/2} b_{2i}} \cdot H \tag{1}$$

The total wirelength is calculated as

$$L = \sum_{k=1}^{\lceil \frac{P}{2} \rceil} [\frac{b_{2k-1} - 1}{b_{2k-1}} \prod_{i=1}^{k-1} b_{2i}] \cdot W + \sum_{k=1}^{\lfloor \frac{P}{2} \rfloor} [\frac{b_{2k} - 1}{b_{2k}} \prod_{i=1}^{k} b_{2i-1}] \cdot H \tag{2}$$

Assuming a linear wire delay model and ignoring buffering, we derive the maximum and minimum (linear-delay) clock latency from clock source to any sink as

$$t_{max} = \frac{1}{2} \cdot [\sum_{k=1}^{\lceil \frac{P}{2} \rceil} (\frac{b_{2k-1} - 1}{\Pi_{i=1}^{k} b_{2i-1}}) \cdot W + \sum_{k=1}^{\lfloor \frac{P}{2} \rfloor} (\frac{b_{2k} - 1}{\Pi_{i=1}^{k} b_{2i}}) \cdot H] \tag{3}$$

$$t_{min} = \frac{1}{2} \cdot [\sum_{k=1}^{\lceil \frac{P}{2} \rceil} (\frac{1}{\Pi_{i=1}^{k} b_{2i-1}}) \cdot W + \sum_{k=1}^{\lfloor \frac{P}{2} \rfloor} (\frac{1}{\Pi_{i=1}^{k} b_{2i}}) \cdot H] \tag{4}$$

The maximum global skew ω is then defined as the difference between t_{max} and t_{min}.

$$\omega = \frac{1}{2} \cdot [\sum_{k=1}^{\lceil \frac{P}{2} \rceil} (\frac{b_{2k-1} - 2}{\Pi_{i=1}^{k} b_{2i-1}}) \cdot W + \sum_{k=1}^{\lfloor \frac{P}{2} \rfloor} (\frac{b_{2k} - 2}{\Pi_{i=1}^{k} b_{2i}}) \cdot H] \tag{5}$$

Figures 2(a) and 2(b) respectively show skew-wirelength and latency-wirelength tradeoffs in GH-trees with various branching patterns. We calculate skew and latency based on a linear delay model[3] according to Equations (3) and (5). The figures also show the Pareto frontier of non-dominated points of each tradeoff. Figures 2(c) and 2(d) respectively show skew-power and latency-power tradeoffs of *buffered* GH-trees with various branching patterns, as reported by a commercial static timing analysis tool with foundry 28LP technology and library models. The Pareto frontier of each tradeoff is shown as the black curve. We observe from the figures that different branching patterns lead to wide-ranging skew-power and/or latency-power tradeoffs. In this work, we explore branching patterns via dynamic programming to optimize tradeoffs among skew, latency and power. Figure 2(c) shows that with the same branching pattern (e.g., (10, 4, 2, 12)), different buffering solutions can lead to more than 20% skew difference with similar clock power. We therefore perform co-optimization of tree topology (i.e., branching pattern) and buffering to minimize clock power, skew and latency.

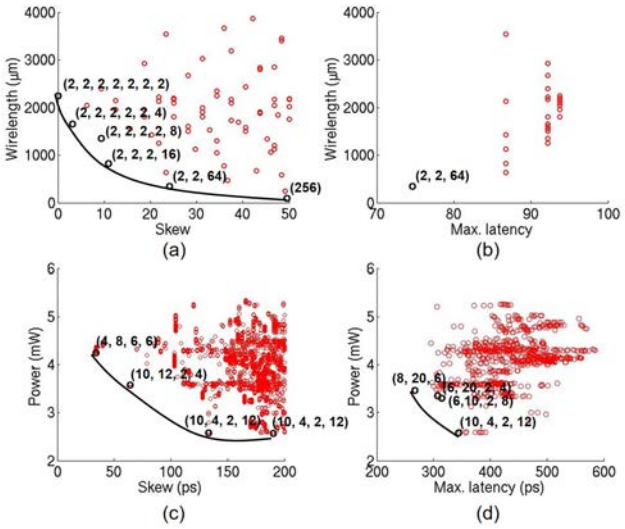

Fig. 2: Study of motivating tradeoffs. (a) Linear delay skew vs. wirelength; and (b) linear delay clock latency vs. wirelength with different branching patterns (256 sinks and region area = 100μm $\times$ 100μm). (c) Skew vs. clock power; and (d) maximum latency vs. clock power, in *buffered* GH-trees for a testcase with 17K sinks and region area = 380μm $\times$ 380μm. In red are dominated points; in black are Pareto frontiers and non-dominated points.

III. Our Approach

We now describe our problem formulation and our approach. Based on the motivating examples shown in Figure 2, we construct GH-trees to explore the tradeoff among skew, latency and clock power. Our construction comprehends the delay and power impact of buffer insertion, sink placement and multiple constraints (e.g., maximum transition time and

[2]Note that $(p+1)/2$ in Equation 1 is always an integer since we create horizontal segments (w_p) and vertical segments (h_p) in alternation, and always start with a horizontal segment from the top. Thus, all horizontal segments are created when p is an odd number.

[3]The linear wire delay model approximates clock latency with relatively low accuracy. However, we give this motivating analysis using the simple linear delay model to more intuitively illustrate the tradeoff among clock power, skew and latency with different GH-tree topologies. Below, we apply comprehensive buffer and wire delay modeling in our optimization to demonstrate the benefits of our proposed approach.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE I: Description of notations used in our work.

Term	Meaning
r (R)	clock tree (set of clock trees)
ω (Ω)	global clock skew (maximum global skew constraint)
t (T)	clock latency (maximum clock latency constraint)
γ (Γ)	clock slew (maximum clock slew constraint)
C	maximum load capacitance constraint
L	clock tree wirelength
O	set of placement blockages
w (W)	width (width of given layout region)
h (H)	height (height of given layout region)
n (N)	number of sink regions (required number of sink regions)
p (P)	level index [p = 1 for clock source] (depth of clock tree)
b_p	branching factor at level p in clock tree
B	branching patterns (i.e., sequences of branching factors $\{b_1,\ b_2,\ ...,\ b_P\}$)
u_k	k^{th} sink cluster ($u_k \in U$)
s_i	i^{th} sink ($s_i \in S$)
$d_{k,i}$	Manhattan distance between sink s_i and root of cluster u_k
$\eta_{k,i}$	binary indicator of whether sink s_i belongs to cluster u_k
$\psi_{j,q}^{llx,lly,urx,ury}$	binary indicator whether j^{th} buffer is located outside the corresponding boundaries of q^{th} blockage
c_i	clock pin capacitance of sink s_i
x_i (y_i)	x-coordinate (y-coordinate) of sink s_i
$((x_k^{ll}, y_k^{ll}),\ (x_k^{ur}, y_k^{ur}))$	bounding box of sink cluster u_k

maximum load capacitance). More specifically, we address the following **GH-tree construction problem:**

Given: a placement solution (i.e., a layout region $W \times H$ and placement of sinks), number of sink regions N such that each region contains <40 sinks (see Footnote 4 below), timing library of clock buffers (.lib), maximum clock skew constraint Ω, maximum clock latency constraint T, maximum transition time constraint Γ (i.e., at both sinks and clock buffer input pins), and maximum load capacitance constraint C.

Perform: GH-tree construction with co-optimization of the clock tree topology (i.e., branching patterns) and buffering to **minimize** clock power, subject to the given constraints.

Figure 3 shows our overall GH-tree construction flow. An instance consists of a post-placement layout, the number of sink regions, and constraints, along with pre-characterized technology- and library-specific lookup tables (LUTs) containing power and delay information of candidate buffering solutions (i.e., segment length and buffer sizes). We perform the GH-tree construction primarily through two main steps: (i) according to the total sink capacitance and layout region area and aspect ratio, we first formulate a dynamic programming problem to co-optimize branching pattern and buffering; and (ii) we then perform balanced K-means clustering and formulate an integer linear programming problem to determine clock buffer placement (i.e., to embed our generalized GH-tree structure into the given sink placement). We note that the key step is the first step (i.e., DP-based co-optimization of clock topology and buffering) that systematically explores the continuum between H-tree and spine to achieve an optimal tradeoff among clock power, skew and latency (within this regime). Last, we realize our GH-tree solution in a commercial P&R tool and report metrics (e.g., skew, latency, power, etc.) to assess solution quality.

Although our approach systematically optimizes the tradeoff among clock power, latency and skew in the GH-tree regime, our method has two limitations that we highlight. First, we do not co-optimize tree topology and buffering together with sink placement, due to large runtime complexity. Second, our approach does not consider clock gating cells in a clock tree. Therefore, an improved resolution of the "chicken-and-egg" loop between (i) placement of roots of sink regions and (ii) top-level tree topology optimization, as well as consideration of clock gating cells during the optimization, remain as open research directions.

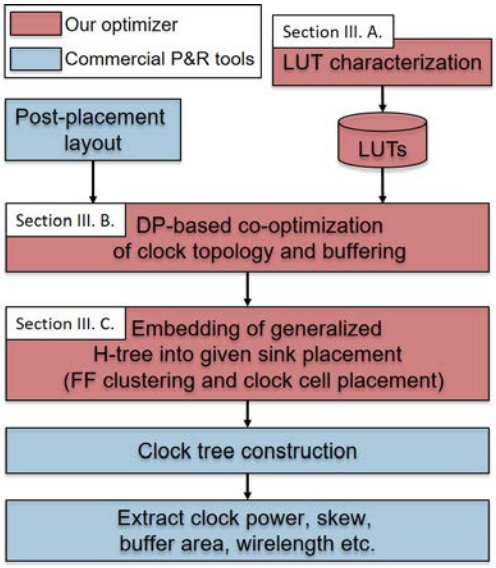

Fig. 3: Overall flow of GH-tree construction.

A. LUT Characterization

We characterize LUTs based on simulations using *Synopsys PrimeTime* [49] as inputs for our DP-based optimization. These LUTs contain power, input capacitance, slew propagation, and delay information of buffered and unbuffered wire segments. We use four types of buffers (X50, X67, X100, X134 from the 28LP libraries). In this technology, the gate area of a X134 buffer is $\sim$7$\times$ the gate area of a minimum-size (X2) buffer. We also use "ganged buffers" (i.e., two, four or six X134 buffers with shorted inputs and shorted outputs) to achieve higher driving strengths. We create wire segments of lengths 15μm, 30μm, 45μm, 60μm, 75μm,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

and $90\mu m$.[4] Along these wire segments, we enumerate all possible buffering solutions with the minimum granularity of $15\mu m$ (i.e., the minimum distance between two buffers is $15\mu m$). The granularity of LUTs (e.g., number of buffer candidates, minimum wire segment length) will determine the tradeoff between optimization runtime and solution quality. In this work, we empirically select the granularity of our LUTs to achieve improved solution quality compared to two commercial tools, while using comparable runtime. For example, with a $45\mu m$ segment length, a minimum buffering distance of $15\mu m$ and seven buffer sizes, there are 8^3 (i.e., no buffer or exactly one of the seven buffer sizes, at each of the three buffering locations) distinct buffering solutions. Note that our LUTs include unbuffered solutions, i.e., pure-wire solutions. We consider both 1W1S (single-width, single-spacing) and 2W2S (double-width, double-spacing – which we understand to be the common non-default routing rule (NDR) for clock distribution) wire segments in our characterization. We vary the input slew from $5ps$ to $60ps$ in steps of $5ps$ and we vary output load from $1fF$ to $150fF$ in steps of $1fF$ from $1fF$ to $5fF$, and in steps of $5fF$ from $5fF$ to $150fF$. For each 3-tuple of (distance, input slew, output load) we obtain the buffering solution (including input capacitance) and the output slew. From the large number of possible solutions, we prune solutions as follows. For each (distance, input capacitance, output slew, output load) 4-tuple, we select three delay values at the 10^{th}, 50^{th} and 90^{th} percentiles of the delay range, and then select the minimum-power solution for each of these three delay values. Figure 4 shows an example of our pruning on LUTs, in which we select minimum-power solutions with different output load values. Red (resp. blue) dots are the buffering solutions with output load = $75fF$ (resp. $35fF$). Cross (x) points are the selected buffering solutions.

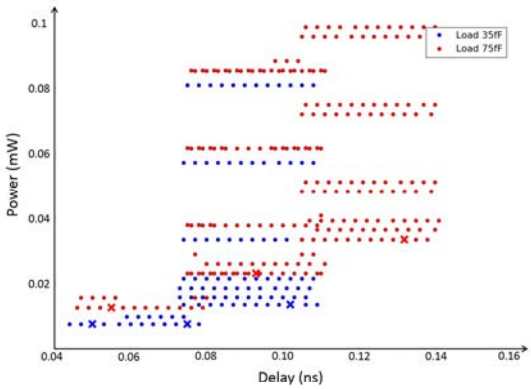

Fig. 4: Example of pruning for buffering solutions; distance = $45\mu m$, output slew = $35ps$.

In practice, we have found that this pruning reduces the number of buffering solutions by ∼94% at the cost of only ∼5% solution quality (i.e., in terms of skew or latency) loss.

[4]We use LUTs based on multiple short wire segments to estimate the delay and slew propagation of a long wire segment. Since we match the output load, input capacitance as well as output and input slew values of two consecutive short segments to form a long segment, the estimation error is negligible. The small estimation error comes from discreteness of capacitance and slew values.

B. DP-Based Co-optimization of Clock Topology and Buffering

Based on the characterized LUTs, we determine the optimal branching pattern along with the buffering solution for GH-tree using dynamic programming (DP). Other inputs to our optimization are layout region, placement of sinks, the number of sink regions, and the maximum skew and maximum latency constraints. A sink region typically contains <40 sinks in our optimization. Our objective is to minimize the clock power while satisfying the given maximum skew and maximum latency constraints. As discussed, in this step, we assume that the sink regions are induced from a uniform placement of sinks, and that branching points are always located at the center of the corresponding sink region. We understand that the sink regions are typically not uniform for a real placement solution. However, due to high runtime complexity, it is practically infeasible for our current approach to consider sink placement during our DP-based optimization. We therefore assume uniform sink regions during our DP-based GH-tree construction. We then embed our DP-based solution (without solution quality degradation) into the given (real) sink placement based on sink clustering and branching point displacement. We formulate our DP in a high-dimensional solution space with seven dimensions (i.e., with respect to seven essential parameters of a clock tree optimization) and construct our GH-tree in a bottom-up way. The seven dimensions are {clock tree depth (P), region area (width (w) and height (h)), number of sink regions (n), maximum and minimum clock latencies (t_{max} and t_{min}), and input capacitance (i.e., the load capacitance seen from the root)}.

Algorithm 1 describes our optimization procedure; see also the illustration in Figure 5. We first construct GH-trees for the base case, that is, trees with depth $= 1$ over all different region areas (i.e., $w \times h$) and numbers of sink regions (i.e., n) (Line 1). As an example, Figure 5 shows the solutions at level p (i.e., subtree with depth $= 1$) with different region areas (i.e., 5×5 (red), 10×10 (green) and 15×15 (purple)). Procedure $build\_base\_trees(w, h, n, \emptyset)$ constructs GH-trees with depths of one (i.e., *spines* with different buffering solutions) within a $w \times h$ region and with a branching factor of b_p. Following [37], we use the term *spine* to denote one horizontal or vertical wire segment in the clock tree. Note that at the bottom level, $b_p = n$. As illustrated in Figure 5, we optimize the buffering solution along the spine based on characterized LUTs. Optimization of each tree segment along the spine generates a minimum-power Pareto surface in the high-dimensional space indexed by the LUT input parameters (e.g., maximum and minimum latencies, and input capacitance). The optimization eventually results in multiple subtree solutions. We store these subtree solutions in a set R indexed by tree depths, region area, number of sink regions, maximum and minimum clock latencies, and input capacitance. In other words, R is a set of subtree solutions along with their depth, region area, number of sink regions, clock latency, input capacitance and power information corresponding to the minimum-power Pareto surface.

Next, we recursively search for the optimal (i.e., minimum-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

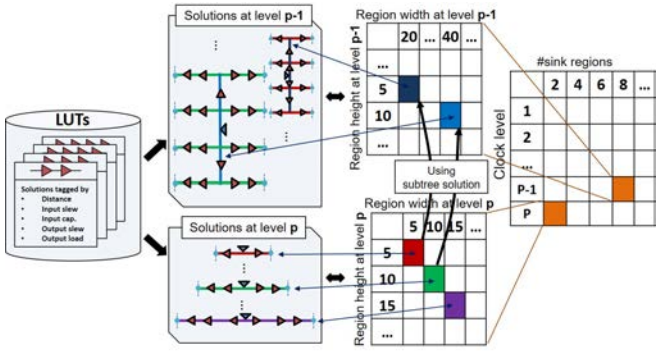

Fig. 5: Co-optimization of GH-tree topology and buffering. The example illustrates construction of trees with depth $= 2$ and eight sink regions, based on subtrees with depth $= 1$ and two sink regions.

Algorithm 1 DP-based GH-tree construction.

```
1: R ← build_base_trees(w, h, n, ∅),  ∀ w, h, n
        s.t. 0 < w ≤ W, 0 < h ≤ H, 2 ≤ n ≤ N, n is an even number
2: for P := 2 to P_max do
3:   for w := 0 to W do
4:     for h := 0 to H do
5:       for n := 2 to (N · w · h)/(W · H) do
6:         R ← retrieve_subtrees(P_l, w_l, h_l, n_l)
7:         for all (r_l) ∈ R do
8:           r ← build_tree(w, h, n, r_l)
9:           r' ← retrieve_tree(R, P, w, h, n, r.t_max, r.t_min)
10:          if r' = null then
11:            R ← R ∪ {r}
12:          else if r.power < r'.power then
13:            remove r' from R
14:            R ← R ∪ {r}
15:          end if
16:        end for
17:      end for
18:    end for
19:  end for
20: end for
21: r_opt.power ← ∞
22: for all r' ∈ R s.t. r'.w = W, r'.h = H, r'.n ≥ N do
23:   if r'.t_max − r'.t_min ≤ Ω && r'.t_max ≤ T && r'.power <
      r_opt.power then
24:     r_opt ← r'
25:   end if
26: end for
27: return r_opt
```

power) GH-tree solutions with depth $P > 1$, region area $w \times h$, and number of sink regions n. We increase P by one per iteration during the optimization, until P = P_{max} (Lines 2–20). The maximum depth P_{max} for a given N is estimated based on the conventional H-tree (which has branching factor of two for all levels), i.e., $P_{max} = \lceil \log_2 N \rceil$. For each depth P, we perform buffering optimization along the tree segments at the topmost level, based on the stored subtree solutions at depth $(P - 1)$. In other words, we use existing solutions (i.e., subtree solutions from R) and add one more (topmost) level with optimized buffering to construct a new tree with depth increased by one. Figure 5 illustrates how our optimizer constructs solutions at level $(p - 1)$ (i.e., subtrees with depth $= 2$) based on the solutions at level p (i.e., subtrees with depth $= 1$). In this example, solutions for region area 20×5 (resp. 40×10) at level $(p - 1)$ are constructed based on four instantiated solutions for region area 5×5 (resp. 10×10) at level p.

For each (P, w, h, n) tuple, we construct our optimization solutions based on the set of all stored subtrees (r_l) (from R) that satisfy

$$P_l = P - 1;\ w_l = h;\ h_l = w/b_t;\ n_l = n/b_t \qquad (6)$$

where P_l is the depth of r_l; $w \times h$ is the dimension of the layout region; $w_l \times h_l$ is the dimension of a sink region (i.e., layout region for subtree r_l); b_t is the branching factor at the topmost level of the current tree; and n_l is the number of sink regions of r_l. Lines 3-4 and Line 5 respectively enumerate possible dimensions and numbers of sink regions for subtree solutions. In Line 6 of Algorithm 1, we find all subtree solutions, which we have optimized in previous iterations, with branching factor b_t such that $2 \leq b_t \leq n/2$.

Procedure $build\_tree(w, h, n, r_l)$ then builds trees r with depth = P, using copies of the collected subtrees $r_l \in R$ (which have depth = $(P - 1)$) as its lower-level subtrees (Line 8). In other words, we build the tree segment with optimized buffering at the topmost level, and at each sink of the topmost segment, we use (i.e., instantiate) the same subtree r_l to build lower levels of the tree r. To reduce the runtime complexity, our current approach assumes that at any level, the subtrees are identical. As shown in Section IV below, this does not preclude strong final solution quality. Among all the constructed trees with depth = P and the same maximum and minimum latency, we select the solution with minimum power and add it to the solution set R (Lines 7–16). Procedure $retrieve\_tree(R, P, w, h, n, r.t_{max}, r.t_{min})$ in Line 9 retrieves a previously stored solution r' from the set R that satisfies the conditions depth = P, width = w, height = h and number of sink regions = n, and has maximum and minimum latencies equal to specified t_{max} and t_{min}, where t is clock latency. Finally, we select the solution with minimum power that satisfies the maximum skew constraint (Ω) and the maximum latency constraint (T) from set R with number of sinks N and region area $W \times H$ (Lines 21–27).

We note that the slews are propagated from top to bottom in a tree. However, our optimization performs bottom-to-top GH-tree construction. We propagate slew bottom-up to accurately capture the slew degradation and avoid maximum transition violations. We first assume several slew values (e.g., $25ps$, $30ps$, $35ps$, $40ps$) at the root of each sink region. For each of the assumed slew values, we propagate slew bottom-up, based on LUTs (note that our LUTs contain output and input slews for each buffering and/or wiring solution). During the DP-based optimization, we only select solutions which ensure that slew values throughout the slew propagation are always within the range of [$5ps$, $60ps$], where $60ps$ is the maximum slew constraint in our experiments, and $5ps$ is the minimum achievable slew in practice in our experiments. We also note that buffer locations are determined by the selected LUT solution with bottom-up slew propagation. Thus, buffers are not necessarily inserted in all branching points.

The runtime complexity of proposed algorithm is $O(P^{max} \cdot \frac{W \cdot H}{w_{int} \cdot h_{int}} \cdot N^2 \cdot \frac{\gamma_{max}}{\gamma_{int}})$, where w_{int}, h_{int} and γ_{int} are respectively the minimum distance and timing intervals for discretization of the original continuous solution space to formulate the dynamic programming; γ_{max} is the specified

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

maximum clock latency constraint. We set w_{int}, $h_{int} < 5\mu$m and $\gamma_{int} = 1ps$ in our experiments. To reduce the runtime, we apply the following pruning techniques.

- **Pruning with number of leaf regions.** For a given sub-region of size $w \times h$ we prune solutions with number of leaf regions greater than $\frac{N \cdot w \cdot h}{W \cdot H}$.
- **Pruning with skew/latency constraints.** We prune solutions that have skew larger than the maximum skew constraint or maximum latency larger than the latency upper bound.
- **Pruning with maximum fanout constraint.** We prune solutions that have branching factor larger than the maximum fanout constraint.[5]

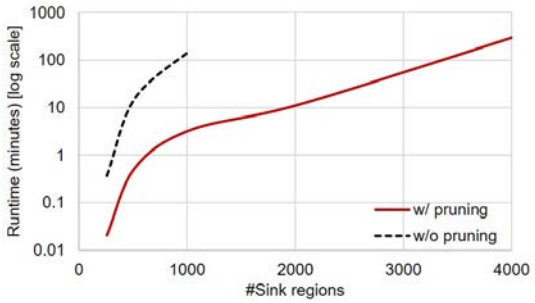

Fig. 6: Runtime of our DP-based optimization method with and without pruning techniques across different numbers of sink regions.

Figure 6 shows DP-based optimization runtime with the number of sink regions ranging from 200 to 4000, where each sink region contains ∼25 flops. The maximum skew constraint used in the experiment is 30ps. Results show that with pruning, the DP-based optimization can optimize a design with more than 4K sink regions (or 100K flops) within six hours. Assuming that the flop count to total instance count ratio is typically 10% to 25%, our approach can optimize a design with 1M instances within six hours. Our studies show that runtime and memory usage increase significantly if we do not apply the proposed pruning techniques, due to the large number of intermediate solutions (such that we are not able to optimize beyond a design with 1K sink regions due to excessive memory usage). Moreover, we observe same solution quality between the runs with and without pruning.

C. Embedding of GH-Tree Into a Sink Placement

The clock tree topology and buffering solution from our DP-based optimization assumes a uniform sink (region) distribution (whereby branching points are at the centers of regions). However, given a (realistic) non-uniform sink (flip-flop) placement, we must cluster flip-flops with balanced load across different clusters to avoid skew and latency increase. In other words, we should assign clusters of flip-flops to sinks of the GH-tree, based on the actual sink flip-flop placement. To adapt our optimized GH-tree to the given sink (i.e., flip-flop) placement, we perform a balanced K-means clustering of sinks and adapt buffer placements based on the clustering solution.[6] We note that our approach is different from conventional top-down clock tree construction methods (e.g., Planar-DME [19]), in that (i) we embed an optimized clock tree topology with buffering solutions to a layout region with given sink placements, and (ii) we balance load capacitance among sink regions. By maintaining the distances between consecutive branching points and buffers at each clock level as well as balancing the load capacitance among sink regions, we preserve the solution quality (i.e., skew and latency) of the GH-tree solution obtained by the DP-based optimization. Furthermore, we understand that the optimal GH-tree topology and buffering solution can vary across different sink placements. With this in mind, we keep the best M solutions from our DP-based optimization and select the minimum-power solution for the given (actual) sink placement as our final solution. Based on our preliminary experiments, we empirically use $M = 5$ to generate the results reported below, where increasing M beyond 5 will not improve the solution quality significantly.

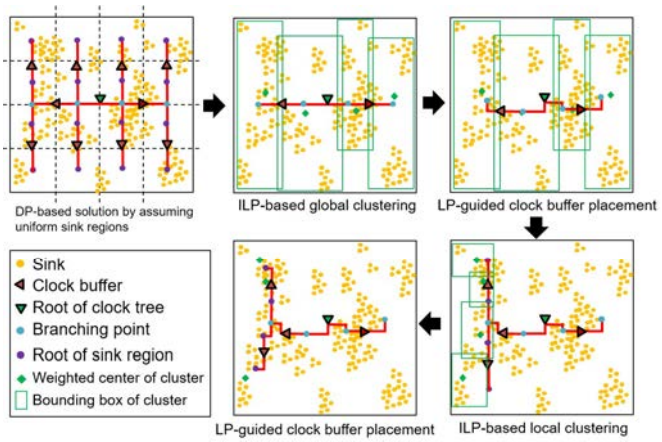

Fig. 7: Example of sink clustering and clock buffer placement. In this example, we only show the local clustering for the leftmost branch of the first-level clock tree.

The remainder of this subsection describes two mathematical programs (ILPs) that perform sink clustering (i.e., to assign flip-flops to sinks of the constructed GH-tree) and place buffers of our GH-tree (one example is shown in Figure 7). The two ILPs respectively act at global and local clustering, as we describe below. We perform the clustering and branching point placement top-down, level-by-level. Our clustering optimization balances the load capacitance across different clusters (each cluster is assigned to a sink of our GH-tree) to minimize the discrepancy between our DP solution (which assumes a uniform sink placement) and the final solution (with the given actual sink placement). Note that this implies that we must consider wire capacitance at the bottom-most level, where the routing is achieved by a commercial P&R tool.[7] However, any constructive approach

[5]In our experiments, we set the maximum fanout constraint to 40 based on guidance from an industrial collaborator [18].

[6]Conventional K-means clock tree synthesis [10] cannot be applied to our problem as it does not comprehend the load capacitance balancing criterion.

[7]Different routing tools can have different clock routing solutions for the bottom-level clock tree. However, the difference is very small. Based on our experimental results, skew from bottom-level clock routing is $<1ps$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Algorithm 2 Embedding of GH-Tree into a sink placement.

```
for p := 1 to P do
  if |S|/|U| ≤ Q_th then
    global_clustering()
  else if r.power < r'.power then
    local_clustering()
  end if
  branching_point_and_buffer_placement()
end for
```

to this wire capacitance estimation is inaccurate and can dramatically increase runtime complexity during top-level optimization where each branching point can have many sink flip-flops. We therefore divide the GH-tree into global and local clustering, based on the total number of downstream sinks (i.e., flip-flops), and only consider wire capacitance during local clustering.

Algorithm 2 describes our clustering procedure. For each level p, we iteratively apply either global or local clustering followed by LP-based branching point and clock buffer placement. Starting from level 1 (the topmost level) of the tree, we cluster sinks based on initial locations of the branching points (i.e., the light blue dots in the top-left figure of Figure 7, which are at the centers of uniform sink regions) of the DP-based GH-tree solution (Algorithm 1). The number of global clusters is the same as the number of branching points. For an example in Figure 7, since $p = 1$ and $b_1 = 4$, our ILP will generate four global clusters that have similar load capacitance. We then formulate an LP to determine the exact branching point locations as well as buffer locations in each global cluster. When the number of sinks in each cluster is smaller than a threshold value (i.e., $|S|/|U| < Q_{th}$), we apply our ILP-based local clustering, and refine the branching point locations in each local cluster using our LP. Note that the "K" in the K-means clustering is determined by the number of branching points in the GH-tree.

ILP formulation for global clustering. We pre-calculate distance $d_{k,i}$ between the branching point of cluster u_k and the sink s_i within the bounding box of the region that will be clustered. The initial locations of branching points are assumed to be at the centers of uniformly-sized regions corresponding to the branching factor. The blue dots in the top-left figure of Figure 7 show an example of initial branching point locations when the branching factor $b_1 = 4$.

Minimize: $$\sum_{s_i \in S} d_i + \alpha \cdot d^{max}$$

Subject to:

$$\sum_{u_k \in U} \eta_{k,i} = 1 \qquad \forall s_i \in S \qquad (7)$$

$$d_i = \sum_{u_k \in U} d_{k,i} \cdot \eta_{k,i} \qquad \forall s_i \in S \qquad (8)$$

$$d^{max} \geq d_i \qquad \forall s_i \in S \qquad (9)$$

$$C^{pin} \cdot (1 - \Delta) \leq \sum_{s_i \in S} c_i \cdot \eta_{k,i} \leq C^{pin} \cdot (1 + \Delta) \qquad \forall u_k \in U \qquad (10)$$

We define d_i as the distance between the sink s_i and the branching point of the cluster that includes the sink s_i; d^{max} denotes the maximum distance among the distances d_i; and α is a weighting factor.[8] Our objective is to minimize the sum of all distances d_i and weighted d^{max}. Constraint (7) ensures that each sink can only belong to exactly one cluster. In Constraints (8) and (9), we obtain d_i for each sink and d^{max}, respectively. Constraint (10) ensures that the total pin capacitance of each cluster satisfies specified lower and upper bounds. The lower and upper bound capacitances are determined by C^{pin}, which is estimated as the total pin capacitance covered by the current region divided by the number of clusters, along with the margin Δ. Since the capacitance cannot be always balanced between clusters, we add margin Δ to ensure that there is a feasible solution of the ILP.

ILP formulation for local clustering. Although the ILP for global clustering finds a balanced pin capacitance solution over all sink regions, it ignores wire capacitance. Therefore, we formulate a second ILP and apply it to local clusters that have smaller regions.

Minimize: $$\sum_{s_i \in S} d_i + \sum_{u_k \in U} \frac{\alpha}{|U|} \cdot (x_k^{ur} - x_k^{ll} + y_k^{ur} - y_k^{ll})$$

Subject to:

$$x_k^{ur} \geq x_i \cdot \eta_{k,i} \qquad \forall s_i \in S,\ u_k \in U \qquad (11)$$

$$y_k^{ur} \geq y_i \cdot \eta_{k,i} \qquad \forall s_i \in S,\ u_k \in U \qquad (12)$$

$$x_k^{ll} \leq x_i + \lambda \cdot (1 - \eta_{k,i}) \qquad \forall s_i \in S,\ u_k \in U \qquad (13)$$

$$y_k^{ll} \leq y_i + \lambda \cdot (1 - \eta_{k,i}) \qquad \forall s_i \in S,\ u_k \in U \qquad (14)$$

$$C^{pin+wire} \cdot (1 - \Delta) \leq \sum_{s_i \in S} (c_i + \zeta) \cdot \eta_{k,i} + \beta \cdot (x_k^{ur} - x_k^{ll} + y_k^{ur} - y_k^{ll}) \qquad (15)$$

$$C^{pin+wire} \cdot (1 + \Delta) \geq \sum_{s_i \in S} (c_i + \zeta) \cdot \eta_{k,i} + \beta \cdot (x_k^{ur} - x_k^{ll} + y_k^{ur} - y_k^{ll}) \qquad (16)$$

+ Constraints (7) and (8)

The objective of this second ILP is to minimize the sum of all distances d_i, of which the definition is the same as above, plus the weighted sum of half-perimeter wirelength (HPWL) of all clusters' bounding boxes. We use HPWL and the number of sinks within a cluster to model the wire capacitance of the sink region. In Constraints (11)–(14), we obtain the lower-left and upper-right corner locations of each cluster's bounding box. λ is a large positive integer. Constraints (15) and (16) ensure that total (i.e., pin and wire) capacitance of each cluster satisfy given lower and upper bounds. ζ and β are respectively coefficients for the number of sinks and the cluster's bounding

[8]Based on our preliminary studies, we empirically use $\alpha = 8$ in our experiments.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

box HPWL in our linear wire capacitance estimation model.[9]

LP formulation for branching point location. Based on the two ILPs described above, we obtain the clustering solution at a given clock level. However, the initial assumption of branching locations to be at the center of a region may cause large skew if the sinks within a cluster are placed non-uniformly. To address this issue, we formulate a linear program (LP) to place each branching point close to the weighted center of each cluster, as follows.

Minimize: d^{Δ}_{max}

Subject to:

$$|x^b_{k+1} - x^b_k| + |y^b_{k+1} - y^b_k| = d^b \quad (17)$$

$$x^{\Delta}_k + x^b_k - x^w_k \geq 0, \quad x^{\Delta}_k - x^b_k + x^w_k \geq 0 \quad (18)$$

$$y^{\Delta}_k + y^b_k - y^w_k \geq 0, \quad y^{\Delta}_k - y^b_k + y^w_k \geq 0 \quad (19)$$

$$x^{\Delta}_{max} \geq x^{\Delta}_k, \quad y^{\Delta}_{max} \geq y^{\Delta}_k \quad (20)$$

$$d^{\Delta}_{max} = x^{\Delta}_{max} + y^{\Delta}_{max} \quad (21)$$

Here, x^b_k and y^b_k denote the x-/y-coordinates of the k^{th} branching point, and x^w_k and y^w_k denote the x-/y-coordinates of the weighted center of the k^{th} cluster.[10] The objective is to minimize the sum of the maximum x- and y-distances between any branching point and its corresponding weighted center. Constraint (17) ensures a fixed distance d^b between any two consecutive branching points. In Constraints (18) and (19), variables x^{Δ}_k and y^{Δ}_k respectively denote the x- and y-distances between the branching point and the weighted center of the k^{th} cluster. We then obtain the sum of the maximum x- and y-distances from Constraints (20) and (21).

Placement blockage. We further show a simple extension of our LP formulation to an ILP formulation to comprehend *blockage-aware* buffer placement. In this extension, we adjust buffer placement to address the existence of blockages. A caveat is that this method may not work well for designs with a large number of blockages and/or a complex floorplan, since our DP-based tree topology and buffering solutions are not aware of blockages. We assume that there are O rectangular blockages. The index of a blockage is denoted by q. Each blockage is defined by its lower-left corner (x^{ll}_q, y^{ll}_q) and upper-right corner (x^{ur}_q, y^{ur}_q). When there are placement blockages in the floorplan, we define the following constraints.

$$\begin{aligned}
&\psi^{llx}_{j,q} = 1 \Leftrightarrow x^f_j < x^{ll}_q \\
&\psi^{urx}_{j,q} = 1 \Leftrightarrow x^f_j > x^{ur}_q \\
&\psi^{lly}_{j,q} = 1 \Leftrightarrow y^f_j < y^{ll}_q \\
&\psi^{ury}_{j,q} = 1 \Leftrightarrow y^f_j > y^{ur}_q \\
&\psi^{llx}_{j,q} + \psi^{urx}_{j,q} + \psi^{lly}_{j,q} + \psi^{ury}_{j,q} \geq 1
\end{aligned} \quad (22)$$

[9]We determine ζ and β by fitting a linear model to wire capacitances extracted for all our testcases. We perform least-squares regression as follows: wire capacitance = $\zeta\cdot$ #sinks + $\beta\cdot$ HPWL. In our 28nm foundry enablement, $\zeta = 0.28$ and $\beta = 2.93$.

[10]We calculate the x- and y-coordinates of each weighted center as the respective means of all x- and y-coordinates of placed sinks in the corresponding cluster.

Here, (x^f_j, y^f_j) is the location of the j^{th} buffer at a given clock level. $\psi^{\{llx,lly,urx,ury\}}_{j,q}$ are binary indicator variables which indicate whether the j^{th} buffer is located outside the corresponding boundaries of the blockages. The last inequality in (22) defines the constraint that at least one of the indicator variables must be true. Satisfying this constraint implies that the j^{th} buffer is not in the bounding box of the q^{th} blockage. Note that with the Constraints (22), the problem becomes an integer linear program (ILP).

IV. Experimental Setup and Results

We conduct our experiments in a commercial foundry's $28nm$ LP technology, with a dual-Vt, 12-track standard-cell library. The input placement solutions (including clock sink placements) are generated using *Cadence Innovus Implementation System v15.2* [44]. Our optimization flow is implemented using C++ and Tcl scripts. We use *CPLEX v12.6* [45] as both ILP solver and LP solver, along with *OpenMP* [48] to enable multi-threaded execution. We execute all our experiments by using up to 40 threads on a 2.6GHz Intel Xeon E5-2690 server. We construct reference clock tree solutions using the latest releases available to us of two leading-edge commercial P&R tools (i.e., Tool1 and Tool2) as well as a state-of-the-art academic tool [39], and report attributes of solutions from these tools along with those of our GH-tree solutions.[11]

TABLE II: Description of our testcases.

Testcases	#Insts.	#FFs	Util. (%)	Max tran. (ps)	Max cap. (fF)
B19	39788	3086	73	60	80
JPEG	46937	4712	73	60	80
VGA	66226	17057	76	60	80
LEON3MP	463104	108817	74	60	80
VGA_blockage	65891	17057	61	60	80
VGV_high_AR	65124	17057	75	60	80

TABLE III: MCMM settings and clock periods (ns) for our testcases.

	C1 = {ss, 0.9V, -40C}	*C2 = {ff, 1.1V, -40C}*
B19	1.5	1.0
JPEG	1.2	1.0
VGA	1.4	1.0
LEON3MP	1.6	1.0
VGA_blockage	1.4	1.0
VGA_high_AR	1.4	1.0

We evaluate our optimizer using four designs: *JPEG* from OpenCores [47], and *B19*, *VGA* and *LEON3MP* from the ISPD-2012 contest [24]. We use the real design (JPEG) and the testcases from the ISPD-2012 contest (B19, VGA, LEON3MP) since they contain datapaths (in contrast to testcases from the ISPD-2010 contest, which do not contain datapath information), thus enabling comparison versus commercial tools. We synthesize these testcases using

[11]The tool flows used in our work are based on latest versions of leading commercial EDA tools available through the respective vendors' university programs. More specific identification of tools and vendors is not permitted by the vendors.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Synopsys Design Compiler H-2013.03-SP3 [51]. Table II summarizes the instance count, number of clock sinks, placement utilization and timing constraints for our testcases. To study the impact of maximum skew and latency constraints on power, we run our optimizer with different maximum skew and latency constraints and collect discrete solution points showing skew-power and latency-power tradeoff. We obtain reference clock solutions with multi-corner multi-mode (MCMM) optimization; we define our mode and corner settings in Table III. We also apply late and early deratings of 1.1 and 0.9 to model OCV effects. We use *Synopsys HSPICE G-2012.06-SP1* [50] to perform timing and power analysis.

A. Comparison with Trees from State-of-the-art CTS

Figure 8 and Figure 9 respectively compare skew and clock power, and maximum latency and clock power, of GH-tree solutions to those from the two commercial tools and one academic flow [39]. Table IV compares #buffers, buffer area, max (insertion) delay across corners, and wirelength among clock tree solutions as well as optimization runtime. All flows use the same sink placement solution as input. We apply the same setups (i.e., clock buffer cells (X50, X67, X100, X134 and ganged buffers), BEOL layers (M3 and M4), maximum transition constraints ($60ps$)) to our GH-tree construction and to both commercial and academic tool flows. We sweep the maximum skew and maximum latency constraints on the four designs for the GH-tree constructions. Blue curves in the figures are skew-power and latency-power Pareto curves of our GH-tree solutions.[12]

Overall analysis. Our results show that our GH-tree solutions achieve significant clock power reduction with similar or reduced skew and latency values as compared to the solutions from both commercial and academic tools. We also include the conventional "strict" H-tree solution as a comparison. Note that we use the same methodology to determine the buffer locations and sizes in "strict" H-tree and GH-tree constructions. In other words, it is unnecessary to add a buffer at each branching point. For example, we achieve power reductions of 30% on B19 and 20% on LEON3MP in Table IV compared to commercial tools' results. Moreover, we observe that due to the symmetric topology of our GH-tree, our GH-tree solution is typically more robust against skew variation across different corners as compared to the clock trees from other tools. As an example, skew of the clock tree solution from Tool2 increases by 137% on design VGA_blockage between two corners; by contrast, our solutions generally have similar skew values across corners. The conventional "strict" H-tree achieves the minimum clock skew but at the cost of larger power, buffer area and wirelength. As an example, for LEON3MP, H-tree has depth $P = 12$, but GH-tree has $P = 10$ (i.e., branching factor = (2, 2, 2, 2, 2, 4, 4, 2, 2, 2)). The shallower depth of GH-tree significantly reduces the number of buffers and wirelength. We also validate our GH-tree optimization on design VGA with high floorplan aspect ratio and existence of placement blockages (shown in Figure 15), where we observe similar clock power and latency, but larger skew, compared to the case without blockage and high floorplan aspect ratio. Compared to the results of [39], our GH-tree achieves much smaller power (i.e., up to 55% on B19) and skew, but at the cost of larger maximum latency.

Power analysis. We also observe that our GH-tree solutions have smaller number of buffers and clock wirelength as compared to solutions from commercial and academic tools. Figure 10 further shows a histogram of clock buffer power (i.e., sum of internal, leakage and dynamic power) values of our GH-tree versus corresponding values from a commercial tool's solution. We observe that our DP-based optimization, which can select its buffering solution "optimally" based on the characterized LUTs, achieves smaller buffer power values for most of the clock buffers. In other words, our GH-tree optimization is able to achieve optimized load capacitance of buffers as well as slew propagation for reduced clock power. The power information from our LUTs enables power-awareness in our DP-based GH-tree construction.

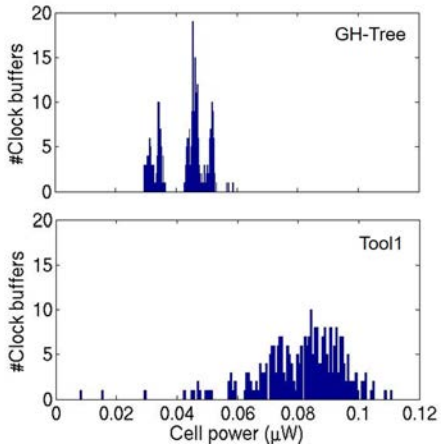

Fig. 10: Distribution of clock buffer power values from GH-tree and commercial tool's solution. Design: VGA.

Runtime analysis. Results in Table IV show that although the naive worst-case time complexity of our (DP- and ILP-based) optimization is high (cf. the nested **for** loops in Algorithm 1), the pruning techniques and empirically selected granularity of our LUTs (e.g., $15\mu m$ for distance, $5ps$ for slew, and $5fF$ for capacitance) make the runtime of our optimization comparable to those of commercial tools. The large runtime of design LEON3MP mainly comes from bottom-level tree construction ($\sim$40 minutes) and ECO routing according to our GH-tree solution using OpenAccess [46] ($\sim$15 minutes). The actual GH-tree construction runtime (i.e., DP + ILP runtime) for design LEON3MP is only $\sim$25 minutes. We understand that such runtime is very acceptable in light of the potential clock power benefits from our approach.

Robustness analysis. We further perform Monte Carlo simulation on our GH-tree solution and those from commercial tools, and compare the resultant variation in clock skew and power. Figure 11 shows that our GH-tree solution exhibits relatively smaller variation in clock skew (i.e., $\sim$35ps) compared to commercial tools' solutions (i.e., $\sim$40ps).

[12]We estimate the Pareto curve based on discrete solution points due to limited computing resources. In addition, since the tradeoff between skew/max latency versus clock power is monotone, we feel that three solution points can provide a useful estimation of the tradeoff.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

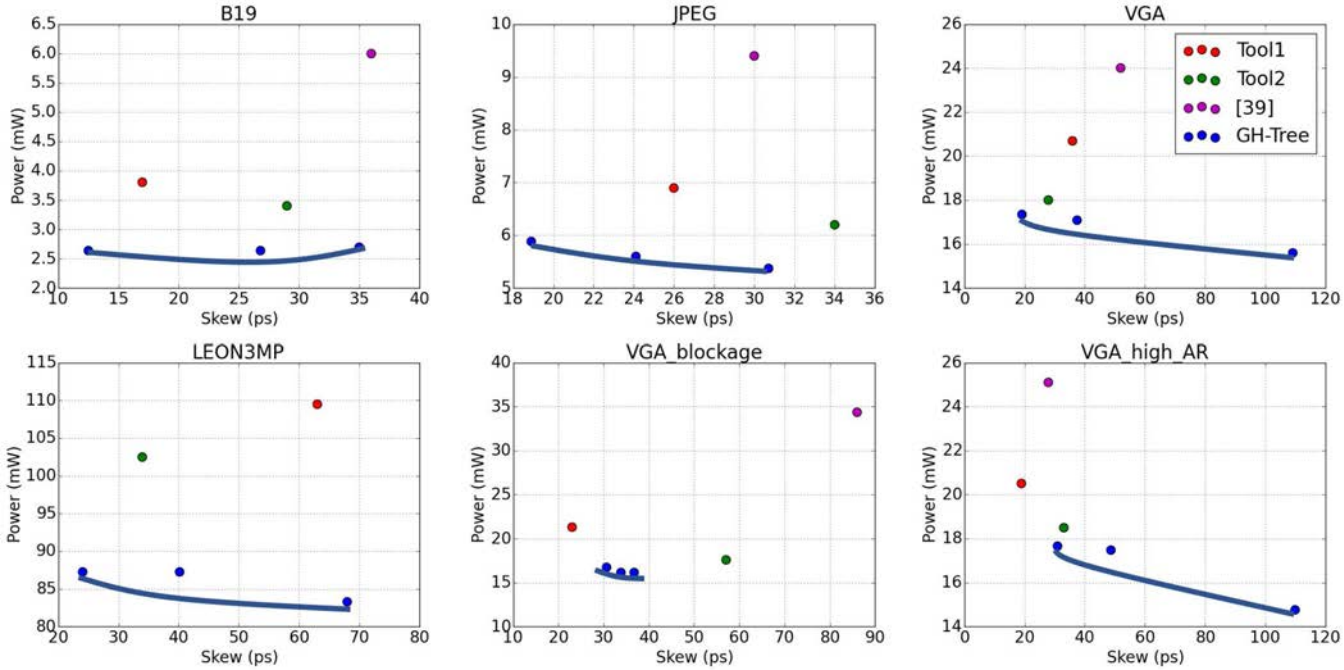

Fig. 8: Power and skew comparisons among GH-tree, Tool1, Tool2 and [39] for four testcases.

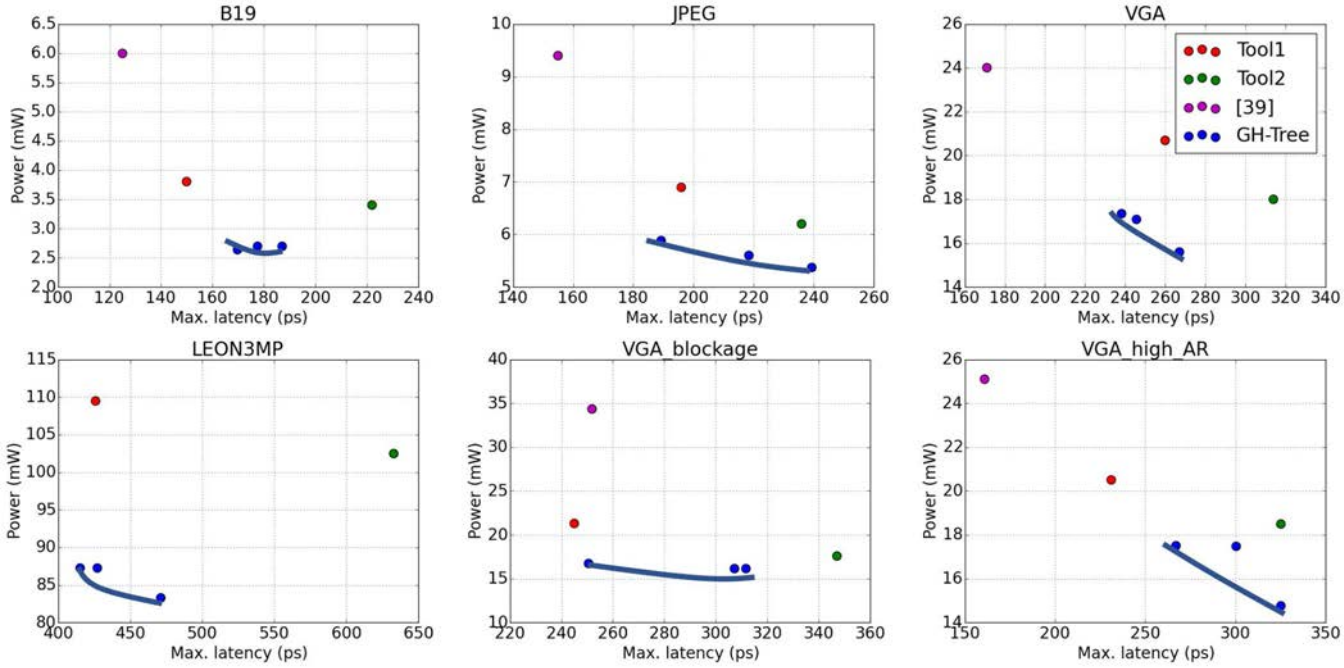

Fig. 9: Power and maximum latency comparisons among GH-tree, Tool1, Tool2 and [39] for four testcases.

Furthermore, all of Tool1, Tool2 and GH-tree solutions have small power variation.

B. Impact of NDR

We now summarize observed impacts of non-default rules (NDRs) on clock tree solution quality. We generate GH-trees with various NDR options: (i) 1W1S only, (ii) 2W2S only, and (iii) the combination of 1W1S and 2W2S. We set the maximum skew constraint to 200ps and compare Pareto curves of the latency versus clock power tradeoffs. Figure 12 shows the comparison for the JPEG testcase at 28LP technology. Due to better slew propagation, solutions with 2W2S have fewer clock buffers as compared to the 1W1S solutions (i.e., the average numbers of clock buffers are respectively 83 and 111 in 2W2S- and 1W1S-only solutions). Further, solutions that permit either 1W1S or 2W2S at each level (of clock subnets) are able to achieve a better tradeoff between latency and power.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE IV: Comparison between clock tree solutions from [39], Tool1 and Tool2 versus our GH-trees. Technology: 28LP.

Testcase	Flow	Corner = *C1*			Corner = *C2*			#Buffers	Buf area	Clk WL	Runtime
		Max laten. (ps)	Skew (ps)	Clk power (mW)	Max laten. (ps)	Skew (ps)	Clk power (mW)		(μm^2)	(mm)	(min)
B19	Tool1	**150**	**17**	**3.8**	110	**27**	9.3	104	227	15688	15
	Tool2	**222**	**29**	**3.4**	129	**5**	8.3	84	245	12996	11
	[39] (min_pwr)	**111**	**40**	**5.6**	63	**40**	12.1	211	338	N/A	N/A
	[39] (min_skew)	**125**	**36**	**6.0**	78	**38**	13.0	228	413	N/A	N/A
	GH-tree	**170**	**12.5**	**2.6**	116	**25.8**	6.4	41	106	12242	15
	H-tree	**166**	**7**	**3.1**	106	**11**	7.6	147	227	13941	16
JPEG	Tool1	**196**	**26**	**6.9**	131	**26**	16.8	160	345	20967	16
	Tool2	**236**	**34**	**6.2**	141	**18**	15.2	120	352	18432	14
	[39] (min_pwr)	**179**	**65**	**9.2**	103	**73**	19.7	340	651	N/A	N/A
	[39] (min_skew)	**155**	**30**	**9.4**	92	**36**	20.4	353	676	N/A	N/A
	GH-tree	**201**	**19**	**5.9**	129	**17**	14.5	147	296	20009	17
	H-tree	**229**	**12**	**6.6**	150	**16**	16.3	169	456	20064	14
VGA	Tool1	**260**	**36**	**20.7**	152	**7**	55.3	464	1119	57678	16
	Tool2	**314**	**28**	**18.0**	201	**11**	48.5	369	1047	56305	22
	[39] (min_pwr)	**171**	**52**	**24.0**	114	**73**	52.1	911	1651	N/A	N/A
	[39] (min_skew)	**171**	**52**	**24.0**	114	**73**	52.1	911	1651	N/A	N/A
	GH-tree	**238**	**19**	**17.4**	174	**41**	44.2	331	1036	57404	21
	H-tree	**253**	**16**	**20.4**	162	**19**	55.0	597	1682	62957	16
LEON3MP	Tool1	**426**	**63**	**109.5**	276	**25**	195.9	2661	6654	369737	54
	Tool2	**633**	**34**	**102.5**	421	**58**	184.2	2509	7225	367854	37
	[39] (min_pwr)	**N/A**	**N/A**	**N/A**	N/A	**N/A**	N/A	N/A	N/A	N/A	N/A
	[39] (min_skew)	**N/A**	**N/A**	**N/A**	N/A	**N/A**	N/A	N/A	N/A	N/A	N/A
	GH-tree	**415**	**24**	**87.2**	285	**35**	157.3	1331	4154	374568	101
	H-tree	**415**	**22**	**96.0**	287	**24**	173.0	2399	6741	393582	99
VGA_blockage	Tool1	**245**	**23**	**21.3**	174	**36**	56.7	475	1148	66323	14
	Tool2	**347**	**57**	**17.6**	212	**24**	47.5	401	1127	64640	24
	[39] (min_pwr)	**298**	**133**	**29.9**	208	**145**	54.5	1118	2154	N/A	N/A
	[39] (min_skew)	**252**	**86**	**34.4**	157	**93**	74.3	1291	2387	N/A	N/A
	GH-tree	**239**	**36**	**16.9**	163	**31**	45.4	293	815	68635	19
	H-tree	**282**	**22**	**20.8**	174	**21**	56.0	599	1685	72636	16
VGA_high_AR	Tool1	**231**	**19**	**20.5**	164	**27**	54.7	456	1094	59506	14
	Tool2	**325**	**33**	**18.5**	211	**43**	49.8	395	1120	58114	21
	[39] (min_pwr)	**161**	**28**	**25.1**	105	**74**	54.5	956	1679	N/A	N/A
	[39] (min_pwr)	**161**	**28**	**25.1**	157	**93**	54.5	956	1679	N/A	N/A
	GH-tree	**265**	**33**	**17.5**	187	**30**	47.3	299	1039	57855	21
	H-tree	**271**	**15**	**20.4**	169	**12**	54.8	661	1669	65181	22

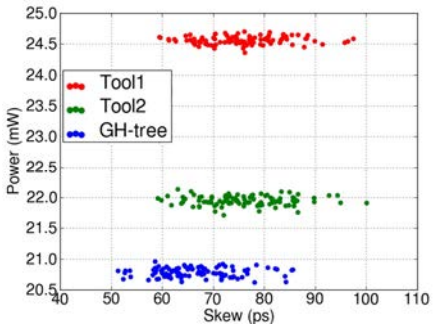

Fig. 11: Clock skew and power comparison among GH-tree, Tool1 and Tool2 through Monte Carlo simulation. Design: VGA.

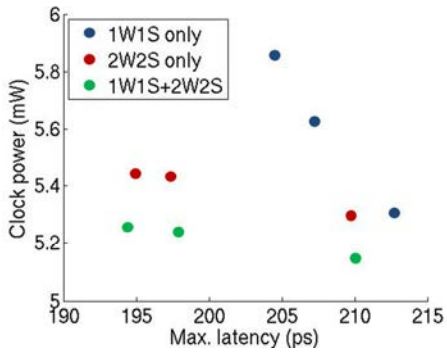

Fig. 12: GH-tree optimization with various NDR options.

C. Study of "Skew Budgeting" across Clock Levels

How to optimally budget skew across clock tree levels has been an open problem for over two decades. Interestingly, our DP approach may provide new insights into how to budget skew for minimum clock power. To study the skew budgeting across clock levels, we run our optimizer multiple times with target skews from $5ps$ to $40ps$ in steps of $5ps$ on testcase *VGA*. In each run, we find the minimum-power GH-tree solution that satisfies the given target skew. Figure 13 shows the normalized skew at each clock level of the minimum-power GH-tree solutions across different target skews . When the skew constraint is tight (i.e., $\leq 15ps$), most of the skew occurs in the bottom levels (i.e., levels 5, 6 and 7) of the clock tree. However, when the skew constraint is relaxed (i.e., $\geq 20ps$), most of the skew occurs in the top levels (i.e., levels 1 and 2) of the clock tree. We show the normalized clock power for each target skew at the top of each bar in Figure 13. For example, minimum-power GH-tree solution for target skew $30ps$ consumes 74% power of minimum-power GH-tree solution for target skew $5ps$. To achieve $\sim$26% clock power reduction by changing target skew from $5ps$ to $30ps$, the clock tree must have $\sim$80% of the skew in levels 1 and 2. We

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2889756, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

emphasize that, as noted above, in our GH-tree a level does not necessarily imply the insertion of a buffer. In other words, a higher number of levels does not necessarily result in larger latency. Rather, we observe from our GH-tree solutions (which are on Pareto frontiers with respect to tradeoffs among skew, latency and clock power) that skew and latency are typically correlated with each other.

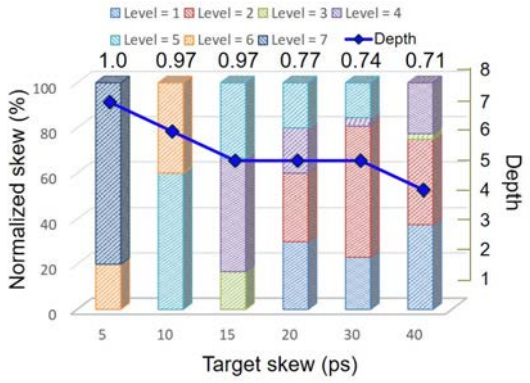

Fig. 13: Skew budgeting, normalized clock power and the number of levels (depth) of the clock tree for different target skews. According to our optimal DP solutions, nearly 80% of skew occurs in the bottom levels of the tree when the target skew is $\leq 15ps$, but this shifts to the top levels of the tree when the target skew is $\geq 20ps$.

V. Conclusion

In this work, we propose the concept of a *generalized H-tree*, which is a balanced tree topology with an arbitrary sequence of branching factors at each level. Our DP-based method provides an *optimal* GH-tree that has minimum clock power for a given skew and maximum latency targets. Our DP solutions are constructed using clock buffers (with ganging) along with interconnect timing and power models from a 28LP foundry design enablement; we co-optimize the clock tree topology along with the buffering along branches. We furthermore propose a clustering- and linear programming-based heuristic to embed the GH-tree with respect to the given placement of clock sinks. We validate our solutions in commercial P&R tool flows in a 28LP foundry technology. The results show up to 30% clock power reduction while achieving similar skew and latency as CTS solutions from recent versions of leading commercial P&R tools. Our proposed approach also achieves up to 56% clock power reduction compared to a state-of-the-art academic tool [39]. Compared to "strict" H-tree, our results achieve better tradeoffs such that power is significantly reduced at the cost of small skew increase. Our ongoing and future work includes (i) co-optimization of sink placement and clock tree construction; (ii) budgeting of skew and latency across levels; (iii) application of useful skew in GH-tree; (iv) application of GH-tree construction in hierarchical designs (that require hierarchical CTS); (v) co-optimization of datapath placement and GH-tree construction; (vi) clock gate- and logic cells-aware GH-tree construction; and (vii) blockage-aware DP-based clock tree topology and buffering.

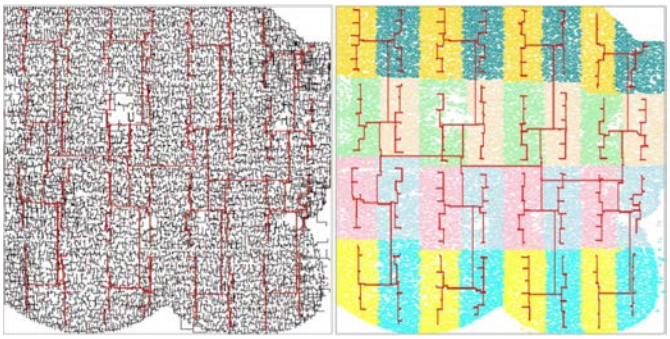

Fig. 14: Layout example of GH-tree on VGA. In red is clock routing of the top four levels in the GH-tree with branching pattern (4, 4, 2, 6). The left figure shows clock routing (top and bottom levels) and the right figure shows the sink clustering solution.

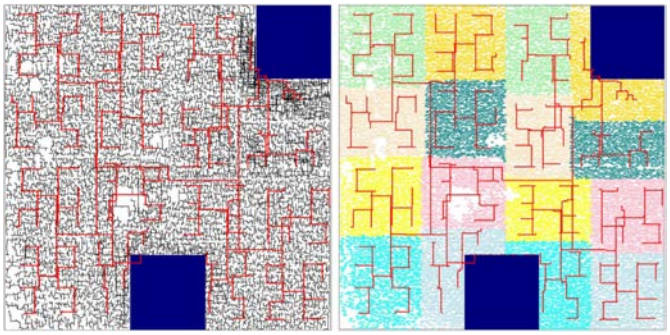

Fig. 15: Layout example of GH-tree on VGA_blockage. In red is clock routing of the top six levels in the GH-tree with branching pattern (2, 2, 2, 2, 2, 4). The left figure shows clock routing (top and bottom levels) and the right figure shows the sink clustering solution.

Acknowledgments

We would like to deeply thank Y. Kim and T. Kim for running their tool on our testcases and providing the results. We also thank Soonbok Jang of Samsung Electronics for her participation in this project while a Visiting Scholar at UCSD and thank Yaping Sun for valuable early discussions.

References

[1] A. Abdelhadi, R. Ginosar, A. Kolodny and E. G. Friedman, "Timing-Driven Variation-Aware Synthesis of Hybrid Mesh/Tree Clock Distribution Networks", *Integration, the VLSI Journal* 46(4) (2013), pp. 382-391.

[2] A. Andreev, A. Nikishin, S. Gribok, P.-C. Tan and C.-H. Choo, "Clock Network Fishbone Architecture for a Structured ASIC Manufactured on a 28 NM CMOS Process Lithographic Node", *U.S. Patent* 8,629,548, January 2014.

[3] C. J. Alpert, Z. Li, G.-J. Nam, S. Ramji, C. N. Sze, P. G. Villarubia and N. Viswanathan, "Structured Placement of Latches/Flip-Flops to Minimize Clock Power in High-Performance Designs", *U.S. Patent* 8,954,912, May 2014.

[4] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*, Reading, MA, Addison-Wesley, 1990.

[5] T.-B. Chan, K. Han, A. B. Kahng, J.-G. Lee and S. Nath, "OCV-Aware Top-Level Clock Tree Optimization", *Proc. GLSVLSI*, 2014, pp. 33-38.

[6] T. H. Chao, Y. C. Hsu, J. M. Ho, K. D. Boese and A. B. Kahng, "Zero Skew Clock Routing With Minimum Wirelength", *IEEE TCAS* 39(11) (1992), pp. 799-814.

[7] M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai and A. Tomkins, "Minimizing Wirelength in Zero and Bounded Skew Clock Trees", *SIAM J. Discrete Mathematics* 17(4) (2004), pp. 582-595.

[8] Y.-Y. Chen, C. Dong and D. Chen, "Clock Tree Synthesis Under Aggressive Buffer Insertion", *Proc. DAC*, 2010, pp. 86-89.

[9] J. Cong, A. B. Kahng, C. K. Koh and C.-W. A. Tsao, "Bounded-Skew Clock and Steiner Routing", *ACM TODAES* 3(3) (1998), pp. 341-388.

[10] C. Deng, Y.-C. Cai and Q. Zhou, "Register Clustering Methodology for Low Power Clock Tree Synthesis", *J. Computer Science and Technology* 30(2) (2015), pp. 391-403.

[11] D. Dolev, M. Fugger, C. Lenzen, M. Perner and U. Schmid, "HEX: Scaling Honeycombs is Easier Than Scaling Clock Trees", *Proc. SPAA*, 2013, pp. 164-175.

[12] R. Ewetz and C.-K. Koh, "Cost-Effective Robustness in Clock Networks Using Near-Tree Structures", *IEEE TCAD* 34(4) (2015), pp. 515-528.

[13] M. R. Guthaus, D. Sylvester and R. B. Brown, "Clock Buffer and Wire Sizing using Sequential Programming", *Proc. DAC*, 2006, pp. 1041-1046.

[14] M. R. Guthaus, G. Wilke and R. Reis, "Non-uniform Clock Mesh Optimization with Linear Programming Buffer Insertion", *Proc. DAC*, 2010, pp. 74-79.

[15] M. R. Guthaus, G. Wilke and R. Reis, "Revisiting Automated Physical Synthesis of High-performance Clock Networks", *ACM TODAES* 18(2) (2013), pp. 31:1-31:27.

[16] K. Han, A. B. Kahng, J. Lee, J. Li and S. Nath, "A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Skew Variation Reduction", *Proc. DAC*, 2015, pp. 26:1-26:6.

[17] J. Hu, A. B. Kahng, B. Liu, G. Venkataraman and X. Xu, "A Global Minimum Clock Distribution Network Augmentation Algorithm for Guaranteed Clock Skew Yield", *Proc. ASP-DAC*, 2007, pp. 24-31.

[18] S. Jang, Samsung Electronics, *personal communication*, May 2015.

[19] A. B. Kahng and C.-W. A. Tsao, "Planar-DME: Improved Planar Zero-Skew Clock Routing with Minimum Pathlength Delay", *Proc. Euro DAC*, 1994, pp. 440-445.

[20] 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", *Proc. ISPD*, 2000, pp. 33-38.

[21] D. Liu and C. Svensson, "Power Consumption Estimation in CMOS VLSI Circuits", *IEEE J. Solid-State Circuits* 29(6) (1994), pp. 663-670.

[22] F. Minami and M. Takano, "Clock Tree Synthesis Based on RC Delay Balancing", *Proc. CICC*, 1992, pp. 28-3.1-28.3.4.

[23] A. D. Mehta, Y.-P. Chen, N. Menezes, D. F. Wong and L. T. Pileggi, "Clustering and Load Balancing for Buffered Clock Tree Synthesis", *Proc. ICCD*, 1997, pp. 217-223.

[24] M. M. Ozdal, C. Amin, A. Ayupov, S. M. Burns, G. R. Wilke and C. Zhuo, "ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite", *Proc. ISPD*, 2012, pp. 161–164, http://archive.sigda.org/ispd/contests/12/ispd2012_contest.html.

[25] M. Pedram, "Power Minimization in IC Design: Principles and Applications" , *ACM TODAES* 1(1) (1996), pp. 3-56.

[26] A. Rajaram and D. Z. Pan, "Variation Tolerant Buffered Clock Network Synthesis with Cross Links", *Proc. ISPD*, 2006, pp. 157-164.

[27] L. Rakai, A. Farshidi, L. Behjat and D. Westwick, "Buffer Sizing for Clock Networks using Robust Geometric Programming Considering Variations in Buffer Sizes", *Proc. ISPD*, 2013, pp. 154-161.

[28] R. R. Rao, D. Blaauw, D. Sylvester, C. J. Alpert and S. Nassif, "An Efficient Surface-Based Low-Power Buffer Insertion Algorithm", *Proc. ISPD*, 2005, pp. 86-93.

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

[30] J. Reuben, H. M. Kittur and M. Shoaib, "A Novel Clock Generation Algorithm for System-on-Chip Based on Least Common Multiple", *Computers and Electrical Engineering* 40(7) (2014), pp. 2113-2125.

[31] J. Reuben, V. M. Zackriya, S. Nashit and H. M. Kittur, "Capacitance Driven Clock Mesh Synthesis to Minimize Skew and Power Dissipation", *IEICE Electronics Express* 10(24) (2013), pp. 1-12.

[32] R. Samanta, J. Hu and P. Li, "Discrete Buffer and Wire Sizing for Link-Based Non-Tree Clock Networks", *IEEE TVLSI* 18(7) (2010), pp. 1025-1035.

[33] V. Sathe, S. Arekapudi, A. Ishii, C. Ouyang, M. Papaefthymiou and S. Naffziger, "Resonant Clock Design for a Power-efficient, High-volume x86-64 Microprocessor", *Proc. ISSCC*, 2012, pp. 140-149.

[34] H. Seo, J. Kim, M. Kang and T. Kim, "Synthesis for Power-Aware Clock Spines", *Proc. ICCAD*, 2015, pp. 126-131.

[35] C. N. Sze, "ISPD 2010 High Performance Clock Network Synthesis Contest: Benchmark Suite and Results", *Proc. ISPD*, 2010, pp. 143-143.

[36] H. Su and S. S. Sapatnekar, "Hybrid Structured Clock Network Construction", *Proc. ICCAD*, 2001, pp. 333-336.

[37] S. Tam, "Modern Clock Distribution Systems" in *Clocking in Modern VLSI Systems*, New York, Springer, 2009.

[38] C.-W. A. Tsao and C.-K. Koh, "UST/DME: A Clock Tree Router for General Skew Constraints", *ACM TODAES*, 7(3) (2002), pp. 359-379.

[39] Y. Kim and T. Kim, "Algorithm for Synthesis and Exploration of Clock Spines", *Proc. ASP-DAC*, 2017, pp. 263-268.

[40] J.-L. Tsai, T.-H. Chen and C. C.-P. Chen, "Zero Skew Clock-Tree Optimization with Buffer Insertion/Sizing and Wire Sizing", *IEEE TCAD*, 23(4) (2004), pp. 565-572.

[41] A. Vittal and M. Marek-Sadowska, "Low-Power Buffered Clock Tree Design", *IEEE TCAD* 16(9) (1997), pp. 965-975.

[42] N. Y. Zhou, P. Restle, J. Palumbo, J. Kozhaya, H. Qian, Z. Li, C. J. Alpert and C. Sze, "PACMAN: Driving Nonuniform Clock Grid Loads for Low-Skew Robust Clock Network", *Proc. SLIP*, 2014.

[43] CAD/CAM/CAE Wallchart. *http://www.garysmitheda.com/wp-content/uploads/2015/05/All_WC-15.pdf*

[44] Cadence Innovus User Guide, http://www.cadence.com

[45] IBM ILOG CPLEX. www.ilog.com/products/cplex/

[46] Si2 OpenAccess. http://www.si2.org/?page=69

[47] OpenCores: Open Source IP-Cores, http://www.opencores.org

[48] OpenMP Architecture Review Board.

[49] Synopsys PrimTime User Guide, http://www.synopsys.com

[50] Synopsys HSPICE User Guide, http://www.synopsys.com

[51] Synopsys Design Compiler User Guide, http://www.synopsys.com

Kwangsoo Han received B.S. and M.S. degrees in electrical engineering from Hanyang University, Seoul, Korea. He joined the VLSI CAD Laboratory, University of California at San Diego, as a Ph.D. student in September 2013. His current research interests include design for manufacturability and VLSI physical design optimization.

Andrew B. Kahng is a professor in the Computer Science Engineering Department and the Electrical and Computer Engineering Department of the University of California at San Diego. His interests include IC physical design, the design-manufacturing interface, combinatorial optimization, and technology roadmapping. He received the Ph.D. degree in Computer Science from the University of California at San Diego.

Jiajia Li received the B.S. degree in software engineering from Shenzhen University, China, in 2011; and the M.S. degree in electrical engineering from the University of California at San Diego, La Jolla, in 2013. He is currently pursuing the Ph.D. degree at the University of California at San Diego. He joined the VLSI CAD Laboratory, University of California at San Diego, in April 2012. His current research interests include physical design and signoff optimization, margin reduction and low-power design.