Clock Tree Construction based on Arrival Time Constraints

Rickard Ewetz[†] and Cheng-Kok Koh[‡]

[†]Department of Electrical and Computer Engineering, University of Central Florida, Orlando, US^{*}

[‡]School of Electrical and Computer Engineering, Purdue University, West Lafayette, US
rickard.ewetz@ucf.edu, chengkok@purdue.edu

ABSTRACT

There are striking differences between constructing clock trees based on dynamic implied skew constraints and based on static arrival time constraints. Dynamic implied skew constraints allow the full timing margins to be utilized, but the constraints are required to be updated (with high time complexity). In contrast, static arrival time constraints are decoupled and are not required to be updated. Therefore, the constraints can be obtained in constant time, which facilitates the exploration of various tree topologies. On the other hand, arrival time constraints do not allow the full timing margins to be utilized. Consequently, there is a trade-off between topology exploration and timing margin utilization. In this paper, the advantages of static arrival time constraints are leveraged to construct clock trees with useful skew while exploring various tree topologies. Moreover, the constraints are specified and respecified throughout the synthesis process reduce the cost of the constructed clock trees. It is experimentally demonstrated that the proposed approach results in clock trees with 16% lower average capacitive cost compared with clock trees constructed based on dynamic implied skew constraints.

1. INTRODUCTION

Limited routing resources and tight power budgets require clock trees to be constructed with short wire length and small buffer area. Moreover, useful skew is required to meet irregular timing constraints and to improve robustness. Sequential circuits are synchronized by a clock signal, which is delivered using a clock tree, from a clock source to a set of sequential elements (or clock sinks). Clock skew is the difference in the arrival time of the clock signal between a pair of clock sinks. There is an explicit skew constraint between each pair of sinks that are only separated by combinational logic. These explicit skew constraints can be captured in a

^{*}Rickard Ewetz performed part of this research at Purdue University. This research was partially supported by NSF awards CFF-1065318 and CFF-1527562.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ISPD '17, March 19-22, 2017, Portland, OR, USA

© 2017 ACM. ISBN 978-1-4503-4696-2/17/03...\$15.00

DOI: <http://dx.doi.org/10.1145/3036669.3036671>

Figure 1: The tree construction in this paper is based on the bounded useful arrival time constraints in (d), which dominates the arrival time constraints in (a), (b), and (c). The static constraints are decoupled from the SCG in (e), in contrast with the dynamic implied skew constraints in (f). $|V|$ is the number of clock sinks and d_{ij} denotes the length of the shortest path from vertex i to vertex j in the SCG.

skew constraint graph (SCG) [17], as shown in Figure 1(e). In an SCG, each vertex represents a sink and each edge represents an explicit skew constraint between the corresponding two sinks.

Based on the explicit skew constraints, there is a dynamic implied skew constraint between every pair of sinks, as shown in Figure 1(f). The bounds of each constraint are defined by the length of two shortest paths in the SCG. Clock trees meeting the explicit skew constraints can be constructed based on iteratively merging subtrees (or sinks) while considering the *dynamic implied skew constraints* [17, 6, 8]. When a pair of sinks are merged, the skew between the sinks is specified and edge weights are required to be updated in the SCG, which in turn requires every implied skew constraint to be updated (with high run-time complexity).

For example, the implied skew constraint between sink 1 and sink 4 may change when the skew between sink 3 and sink 4 is specified, as illustrated in Figure 1(e) and (f). In [17], the Greedy-UST/DME algorithm was proposed to construct useful skew trees based on implied skew constraints. However, only a limited number of topologies were explored, as it is costly in run-time to update the constraints [17, 6, 8].

An alternative to implied skew constraints is *static arrival time constraints* [13, 11, 5, 2], which consist of a range (or in special cases a point) as the arrival time constraints for each sink, as shown in Figure 1(a)–(d). The constraints are satisfied if the clock signal is delivered within the ranges. The advantage of static arrival time constraints is that the constraints are not required to be updated because they are defined with respect to an arbitrary reference point. Using the reference point, the constraints can be obtained in constant time. However, static constraints are inherently more restrictive than implied skew constraints.

Zero skew trees (ZSTs) and useful skew trees (USTs) can be constructed using static equal [13] and static useful [11] arrival time constraints, both of which are point constraints, as shown in Figure 1(a) and (b), respectively. A ZST is constructed using zero-skew merging by storing the delay of each subtree [13]. The useful arrival time constraints in Figure 1(b) can be obtained using a linear programming (LP) formulation that optimizes clock period or robustness [11]. Next, using virtual delay offsets to account for the non-alignment of the point constraints, a clock tree can be constructed using zero-skew merging [13].

In [5], bounded skew trees (BSTs) were constructed based on static bounded arrival time constraints, which are range constraints, as illustrated in Figure 1(c). The expansion of a point constraint to a range constraint resulted in clock trees with shorter wire lengths. As the skew bound can be obtained in constant time and that the minimum and the maximum delay of each subtree can be stored, subtrees pairs can be merged in constant time [5], which facilitates the exploration of various tree topologies. In [5], the exploration was guided by a rerooting feature that transforms a subtree into subtrees with different tree topologies (further details in Section 2.1 and Figure 2). However, static bounded arrival time constraints do not allow useful skews.

In [12], USTs were constructed based on the static bounded useful arrival time constraints [2], as shown in Figure 1(d). The constraints allow both range constraints and useful skew to be utilized. Given the explicit skew constraints, alternative sets of range constraints can be specified [2]. In [12], the range constraints were specified to maximize the length of the range constraints, to potentially reduce the cost of the constructed clock trees. The limitation of this approach is that by maximizing the length of the range constraints, the range constraints may become unaligned, which constrains the tree construction. Moreover, while merging subtrees, no routing tree topologies were maintained and no interconnect delays were computed (in contrast with in [13, 11, 5]).

In this paper, we propose a tree construction algorithm based on static bounded useful arrival time constraints. The algorithm allows clock trees with useful skew to be constructed while both exploring various tree topologies and accounting for interconnect delays. Moreover, the range constraints are specified using a LP formulation that aims to reduce the capacitive cost of the constructed clock trees.

The BST construction (in [5]) is extended such that an

UST can be constructed given a set of static bounded useful arrival time constraints. The extension is based on introducing a virtual minimum delay offset and a virtual maximum delay offset for each sink, i.e., combining the construction techniques for the constraints in Figure 1(b) and (c). The values of the offsets are defined by the arrival time constraints. The extension maintains that pairs of subtrees can be merged in constant time, facilitating the exploration of various tree topologies. In contrast with the tree construction in [12], the proposed approach allows routing topologies to be maintained and interconnect delays to be computed while merging subtrees.

Given a set of explicit skew constraints, many alternative sets of static bounded useful arrival time constraints can be specified [2]. Each set of constraints results in a clock tree with a different capacitive cost. We attempt to minimize the capacitive cost by specifying the arrival time constraints while considering both the length and the alignment of the constraints using an LP formulation. In [2, 12], only the length of the range constraints was considered.

Although the static bounded useful arrival time constraints do not allow for full utilization of timing margins, the ability to explore various topologies translates into cost reduction, as a larger solution space is explored. Experimental results show that the proposed approach is capable of constructing clock trees with similar robustness and 16% lower cost.

The remainder of the paper is organized as follows: the constraints and the problem formulation are introduced in Section 2 and in Section 3, respectively. The tree construction is outlined in Section 4. In Section 5, the static constraints are specified using an LP formulation. The synthesis flow and experimental results are presented in Section 6 and Section 7. We conclude in Section 8.

2. STATIC AND DYNAMIC CONSTRAINTS

Setup and hold time constraints are imposed between each pair sequential elements, or flip flops (FFs), that are separated by only combinational logic. The setup and hold time constraints between a launching flip flop FF_i and a capturing flip flop FF_j are formulated as follows:

$$t_i + t_i^{CQ} + t_{ij}^{max} + t_j^S \leq t_j + T, \quad (1)$$

$$t_i + t_i^{CQ} + t_{ij}^{min} \geq t_j + t_j^H, \quad (2)$$

where t_i and t_j are the arrival times of the clock signal to FF_i and FF_j , respectively. t_{ij}^{min} and t_{ij}^{max} are the minimum and maximum propagation delay through the combinational logic; t_i^{CQ} is the clock to output delay of FF_i ; T is the clock period; t_j^S and t_j^H are the setup and hold time of FF_j , respectively. The setup and hold time constraints in Eq (1) and Eq (2) can be reformulated into **explicit skew constraints** as follows:

$$t_h - t_k \leq c_{hk}, \quad (3)$$

where t_h , t_k , and c_{hk} are respectively equal to t_i , t_j , and $T - t_i^{CQ} - t_{ij}^{max} - t_j^S - M_{user}$, for each setup constraint in Eq (1). In addition, t_h , t_k , and c_{hk} are respectively equal to t_j , t_i , and $t_{ij}^{min} + t_i^{CQ} - t_j^H - M_{user}$, for each hold time constraint in Eq (2). M_{user} is a user specified non-negative safety margin that is introduced to account for on-chip variations.

The explicit skew constraints in Eq (3) can be captured in a skew constraint graph (SCG). In an SCG $G = (V, E)$, V is the set of sequential elements and E is the set of skew

Table 1: Comparisons of tree construction techniques. A ‘*’ implies that rerooting was not performed but it would be easy to apply. ‘n/a’ denotes that no tree topology is maintained while merging subtrees.

Tree construction proposed in	Constraints	Update required?	Ease of exploring tree topologies based on rerooting	Useful skews allowed	Degree of utilization of timing margins	Considers interconnect delays during merging
[13]	Static equal arrival time [13]	No	easy*	No	low	Yes
[13]	Static useful arrival time [11]	No	easy*	Yes	low	Yes
[5]	Static bounded arrival time [5]	No	easy	No	medium	Yes
[12]	Static bounded useful arrival time [2]	No	‘n/a’	Yes	high	No
[17]	Dynamic implied skew [17]	Yes	difficult	Yes	full	Yes
This paper	Static bounded useful arrival time [2]	No	easy	Yes	high	Yes

constraints. For each skew constraint in Eq (3), an edge e_{hk} from vertex h to vertex k is added with a weight $w_{hk} = c_{hk}$. Throughout the synthesis process, skews are specified between pairs of sequential elements. If a skew $skew_{ij} = t_i - t_j = a$ is specified between sink i and sink j , the weight of the edges e_{ij} and e_{ji} are updated to $w_{ij} = a$ and $w_{ji} = -a$, respectively, as shown in Figure 1(e).

Dynamic implied skew constraints are imposed between each pair of sinks by the explicit skew constraints. In [17], it was shown that the implied skew constraints between a pair of sinks is defined as follows:

$$-d_{ji} \leq t_i - t_j \leq d_{ij}, \quad (4)$$

where d_{ij} and d_{ji} denotes the shortest path from vertex i to vertex j and from vertex j to vertex i , respectively, in the SCG. As the implied skew constraints are defined based on the SCG, they are required to be updated when any skew is specified in the SCG. The time complexity to compute or update an implied skew constraint is $O(V \log V + E)$ [6].

A **static arrival time constraint** is a range of arrival time constraints, denoted r_i , for each sink i , with respect to an arbitrary reference point. The arrival time constraints are satisfied if the clock signal is delivered to the sinks within the range constraints [2]. A set of arrival time constraints are defined to be *valid* if they guarantee that the explicit skew constraints in the SCG are satisfied, which can be ensured as follows:

$$x_i^{lb} \leq x_i^{ub}, \quad \forall i \in V \quad (5)$$

$$x_i^{ub} - x_j^{lb} \leq w_{ij}, \quad \forall (i, j) \in E \quad (6)$$

where x_i^{lb} and x_i^{ub} respectively denote the lower and the upper bound of the range r_i . V and E are the vertices and the edges in an SCG, respectively. (The x_i^{lb} and x_i^{ub} notation is illustrated in Figure 3 in Section 4.2.)

As the arrival time constraints are specified with respect to an arbitrary reference point, they are not required to be updated when skews are specified in the SCG. Moreover, the reference point is not required to be specified.

2.1 Arrival time vs. implied skew constraints

In Table 1, it can be observed that the proposed tree construction is advantageous to the earlier tree construction approaches based on **static arrival time constraints** [2].

The proposed tree construction dominates the tree construction approaches in [13, 5] because the tree construction is based on the static useful bounded arrival time constraints illustrated in Figure 1(d), which is a dominating generalization of the constraints used in [13, 11, 5], shown in Figure 1(a), (b), and (c), respectively.

Both the proposed tree construction and the tree construction in [12] are based on the static bounded useful arrival time constraints. Consequently, both approaches allow useful skew and have a high degree of timing margins utilization, as unaligned range constraints are used. The difference is that the proposed approach (in similar to in [13, 5]) maintains the routing tree topology of each subtree, which is not performed in [12]. Without a tree topology, interconnect delays cannot be computed, as indicated in Table 1. However, it should be noted that after sufficiently large subtrees have been formed in [12], a tree topology is generated for each subtree and interconnect delays are computed. Nevertheless, these generated tree topologies may violate the skew constraints as the interconnect delays were not considered during the tree generation.

In tree construction based on static bounded useful arrival time constraints, a set of range constraints are required to be specified based on the explicit skew constraints. The specification is coupled with the tree construction problem, as alternative sets of range constraints translate into clock trees with different capacitive cost. In [2], it was observed that the longer range constraints correspond to less constrained tree construction. Therefore, the lengths of the range constraints were lexicographically maximized, i.e., the minimum length range constraint was iteratively maximized up to a threshold. However, for two sinks that are located physically close to be able to be merged meeting the timing constraints, the range constraints have to intersect. (Minor misalignments can be compensated by the interconnect delays in the routing tree topology.) The formation of intersecting range constraints is not directly captured in [2]. Therefore, we propose to specify the range constraints while considering both alignment and length using an LP formulation, which is further discussed in Section 5.

Compared with using **dynamic implied skew constraints** [17], the advantage of performing tree construction based on static arrival time constraints is that the constraints are not required to be updated, they are decoupled from the SCG. Consequently, a pair of subtrees can be merged in constant time. Therefore, it is run-time feasible to evaluate merging two subtrees while exploring various tree topologies. In [5], the topology exploration is performed using a rerooting feature. A subtree with $n \geq 2$ leaf nodes can be rerooted into $2n - 3$ subtrees with different tree topologies, which is illustrated in Figure 2. Consequently, two subtrees with $n \geq 2$ and $m \geq 2$ respective leaf nodes can be merged to a subtree with $(n+m)$ leaf nodes while considering $(2n-3) \cdot (2m-3)$ tree topologies. The two drawbacks of tree construction based on static arrival time constraints are: (i) Arrival time constraints are inherently more restrictive than

the explicit skew constraints stored in the SCG. This can be understood because the explicit skew constraints between a pair of sinks have to be satisfied for any pair of arrival times within the respective ranges, see Eq (6); (ii) The static approach does not leverage that the SCG is updated with skew information throughout the tree construction process, which may expose additional timing margins.

Figure 2: For a subtree with n leaf nodes, $2n-3$ tree topologies are explored by rerooting [5].

The advantage of tree construction based on using dynamic implied skew constraints is that the full timing margins can be utilized. However, it is costly in terms of run-time to explore various topologies, as the implied skew constraints have to be updated after each skew in a topology is specified. The update of each implied skew constraints is $O(V \log V + E)$ [6].

In Table 1, it can be observed that there exists a trade-off between using static arrival time constraints and dynamic implied skew constraints, i.e., ease of topology exploration versus degree of utilization of timing margins. To allow the static approach to expose additional timing margins, we propose to re-specify the static bounded useful arrival time constraints periodically throughout the tree construction process, i.e., mitigating the shortcomings of static arrival time constraints. Further details are provided in Section 6.1.

3. PROBLEM FORMULATION

This paper considers a useful skew clock tree synthesis problem. The problem consists of constructing a clock tree that delivers a clock signal from a clock source to a set of sequential elements while meeting the skew constraints in Eq (3) and transition time constraints. The source to sink connections are realized using wires and buffers from a wire and buffer library, respectively. The objective is to construct clock trees using the least amount of wire and buffer resources. The resource utilization is measured in capacitive cost, which is known to correlate closely with power consumption.

We approach the problem by extending the BST construction such that an UST can be constructed given a set of static bounded useful arrival time constraints (see Section 4). Given an SCG, many alternative sets of static arrival time constraints can be specified, each resulting in a clock tree with a different capacitive cost. In Section 5, we specify the static bounded useful arrival time constraints with the goal of minimizing the capacitive cost of a clock tree constructed using the constraints.

4. BST AND UST TREE CONSTRUCTION

In Section 4.1, we review the BST construction in [5]. In Section 4.2, the BST construction is extended such that USTs can be constructed based on static bounded useful arrival time constraints while considering interconnect delays.

4.1 BST tree construction in [5]

In [5], the BST construction is based on the observation that if the maximum skew between any pair of sinks is less than B , the clock signal will be delivered within the range constraints, illustrated in Figure 1(c). Here, B is equal to the range, i.e., the difference of upper and lower bounds, of each arrival time constraints.

To facilitate the construction of such a BST, the minimum and maximum delay of each subtree i are stored and denoted min_t_i and max_t_i , respectively. Initially, min_t and max_t are set to 0 for a subtree (or sink). Next, a clock tree is constructed by iteratively merging subtrees while ensuring that $max_t_k - min_t_k \leq B$ of each formed subtree k .

A pair of subtrees i and j are merged into a larger subtree k with $max_t_k - min_t_k \leq B$ as follows: the subtrees i and j are connected with a wire and the length of the wire is equal to the Manhattan distance between the subtrees. (For certain pairs of delay imbalanced subtrees, detour wiring is required [13].) Next, the alternative locations for the root of subtree k are determined on the wire. This can be performed in constant time, as the skew bound B can be obtained in constant time and min_t_k and max_t_k can be computed incrementally as follows:

$$min_t_k = \min\{min_t_i + w(k, i), min_t_j + w(k, j)\}, \quad (7)$$

$$max_t_k = \max\{max_t_i + w(k, i), max_t_j + w(k, j)\}, \quad (8)$$

where $w(k, i)$ and $w(k, j)$ denotes the interconnect delay of the wire between the root of the subtree k and the root of the subtrees i and j , respectively.

Before merging a pair of subtrees, each subtree can be rerooted into multiple subtrees with different topologies, as illustrated in Figure 2. During rerooting, it is utilized that min_t_p and max_t_p are computed and stored for each partial subtree p of a larger subtree. Moreover, each rerooted subtree can be obtained by pairwise merging three partial subtrees of the initial subtree (or a previously rerooted subtree). Therefore, each rerooted subtree can be formed in constant time. The run-time is linear with respect to the number of rerooted topologies that are explored.

As no routing tree topologies are generated in [12], no interconnect delays can be computed, which is equivalent to setting $w(k, i) = 0$ and $w(k, j) = 0$ in both Eq (7) and Eq (8).

It can be understood that the reference point to which the range constraints are defined is arbitrary, because only the relative delay between pairs of sinks is required to meet a skew bound. With a non-arbitrary reference point, the skew bound $B = 50$ could for example mean that the clock signal must be delivered to each sink with a delay in [200, 250] ps.

In the next section, we extend the BST construction such that an UST can be constructed based on static bounded useful arrival time constraints, as shown in Figure 3.

4.2 Proposed UST construction

The extension is based on using a maximum skew bound B^v (in similar to B in [5]) and virtual minimum and virtual maximum delay offsets to account for the non-alignment and

the range of the arrival time constraints, similar to using single delay offsets to handle the constraints in Figure 1(b) based on ZST construction [13].

Based on the arrival time constraints, B^v is set to an arbitrary value that satisfies $\frac{B^v}{2} \geq x_i^{ub}$ and $\frac{B^v}{2} \geq -x_i^{lb}$ for all $i \in V$, which is illustrated in Figure 3. The virtual minimum delay offset off_i^{min} and virtual maximum delay offset off_i^{max} for a sink i are specified by the arrival time constraints and B^v as follows:

$$off_i^{min} = -\frac{B^v}{2} - x_i^{lb}, \quad (9)$$

$$off_i^{max} = \frac{B^v}{2} - x_i^{ub}, \quad (10)$$

Finally, an UST can be constructed in a similar fashion as an BST in [5], by setting $B = B^v$ and $min_t_i = off_i^{min}$ and $max_t_i = off_i^{max}$ for each sink i , respectively.

The skew bound B^v can be obtained in constant time and min_t and max_t can still be incrementally computed for each subtree. Therefore, it is possible to merge subtrees in constant time and explore various topologies. Note that the reference point is arbitrary and not specified and that B^v can in fact be defined to an arbitrary value by the offsets.

Figure 3: Tree construction based on static bounded useful skew constraints.

Now that we have explained how a UST can be constructed given a set of static bounded useful arrival time constraints, we focus on specifying the constraints with the goal of minimizing the capacitive cost of a clock tree constructed using the constraints.

5. PROPOSED SPECIFICATION OF ARRIVAL TIME CONSTRAINTS

In this section, valid static bounded useful arrival time constraints are specified based on the explicit skew constraints. It is not difficult to specify a set of valid arrival time constraints. Every feasible solution of an LP formulated with the constraints in Eq (6) and Eq (5) forms a set of valid arrival time constraints. The challenge is how to define a suitable objective function, such that the solution to the LP formulation results in arrival time constraints that help to minimize the capacitive cost of the clock tree constructed.

We approach this challenge by observing the following property of arrival time constraints: let r^I be the intersection of the arrival time constraints of all sinks and let $|r^I|$ be the range of r^I (if the intersection is non-empty). *All subtree(s) constructed from the sinks satisfying a skew bound $B = |r^I|$ will satisfy the arrival time constraints.*

It can be easily understood that the larger the $|r^I|$ is, the less constrained the tree construction is and therefore, the more likely the clock tree will have lower capacitive cost. Suppose we construct the bottom k stages of a clock tree without considering any skew constraints, where a stage consists of subtrees, each driven by a buffer. Let $skew^{(k)}$ denote the maximum skew between any pair of sinks in the subtrees of these bottom k stages constructed in such a fashion. We attempt to specify the arrival time constraints with $|r^I| \geq skew^{(k)}$. This would imply that the k bottom-most stages could be constructed in an unconstrained fashion, which probably would result in clock trees with small capacitive cost, as it is well known that a majority of the capacitive cost of a clock tree is located in the bottom most stages [7, 3].

The limitation of the proposed approach is that if any explicit skew constraints require useful skew to be satisfied, i.e., $t_i - t_j \leq -b$, where $b > 0$. No common intersection r^I exists, which is the main limitation of the bounded arrival time constraints in [5].

It can also be understood that by lexicographically maximizing the length of the range constraints as in [2], intersecting range constraints may be formed. However, it may be more effective to both consider length and alignment in the specification process, as in the approach proposed in the next section. The capacitive cost of the clock trees constructed based on the two approaches are compared in Section 6.

5.1 Proposed LP formulation

We propose to specify the arrival time constraints with the following goals: (1) The range constraints have to be valid, i.e., the constraints in Eq. (6) and Eq. (5) have to be satisfied. (2) The lower and upper bounds of each range constraint should be minimized and maximized, respectively. (3) The arrival time constraints should be aligned although they are allowed to be unaligned (to allow insertion of useful skews). (4) Arrival time constraints of similar range are preferred. The motivation for this preference is that a subtree is always more constrained timing wise than the subtrees from which it was constructed. Tree construction is constrained by the arrival time constraints with the smallest range.

With these goals, we propose the following LP formulation:

$$\min \sum_{i \in V} f(x_i^{lb})^{lb} + f(x_i^{ub})^{ub} \quad (11)$$

$$x_i^{lb} \leq x_i^{ub}, \quad \forall i \in V \quad (12)$$

$$x_i^{ub} - x_j^{lb} \leq w_{ij}, \quad \forall (i, j) \in E \quad (13)$$

where, $f(x)^{lb}$ and $f(x)^{ub}$ are convex p -part piecewise linear functions shown in Figure 4. In Figure 4, $c_1, \dots, c_p$ are user specified weights and $\frac{skew^{(1)}}{2}, \dots, \frac{skew^{(p-1)}}{2}$ are stage skews.

It is evident that the formulation achieves the goals (1) and (2) by the constraints in Eq (12) and Eq (13) and the objective function. The formulation achieves the goals (3) and (4) by setting the slope of the piecewise linear functions $f(x)^{lb}$ and $f(x)^{ub}$ as illustrated in Figure 4(a) and (b). In the figure, it can be observed that there is a heavy penalty if the lower bound (the upper bound) of a range is not set to be lesser (greater) than $-\frac{skew^{(1)}}{2}$ ($\frac{skew^{(1)}}{2}$). Moreover, the slopes of $f(x)^{lb}$ and $f(x)^{ub}$ are changed at certain multiples of $-\frac{skew^{(1)}}{2}$ and $\frac{skew^{(1)}}{2}$, to encourage sim-

ilar ranges and arrival constraints that are aligned and centered in $[-\frac{skew^{(1)}}{2}, \frac{skew^{(1)}}{2}]$ to be formed. Empirically, we find that it is important to set the slope, of the different parts to be drastically different, to avoid having constraints with disproportionately small ranges. In our implementation, $c_i = 200^i/20000$.

Figure 4: Convex piecewise linear functions $f(x)^{lb}$ and $f(x)^{ub}$.

6. CTS AND ITS EVALUATION

6.1 Flow for tree construction

Clock trees with buffers are constructed by integrating the proposed constraints into a classical bottom-up tree construction framework, which is based on algorithms in [17, 5, 4, 8, 10]. An overview of the framework is shown in Figure 5.

Figure 5: Flow for tree construction.

A clock tree is constructed buffer stage by buffer stage. A buffer stage consists of a set of subtrees, each driven by a buffer. The input to the construction of the bottom most stage is the clock sinks, and the input to the construction of the consecutive stages are the input pins of the driving buffers of the previous buffer stage. Each buffer stage is constructed by specifying (or re-specifying) the static arrival time constraints (see Section 5). Next, subtrees are iteratively pairwise merged to form larger subtrees while satisfying the arrival time constraints (see Section 4.2). Lastly, buffers are inserted to drive the constructed subtrees such that the transition time constraints are satisfied [4, 8, 10]. The iterative buffer stage construction process continues until only a tree remains.

Specify or re-specify static arrival time constraints: In the construction of the bottom buffered stage, the arrival time constraints are specified with respect to the sinks, as described in Section 5. In the construction of a higher-level buffer stage, each subtree can be viewed as a sink and the arrival time constraints are re-specified, i.e., a single range constraint is specified for each subtree. The re-specification exposes additional timing margins by including the skew information in the SCG obtained from the construction of lower-level buffer stages.

6.2 Experimental evaluation

In the remainder of this section, we present experimental results to demonstrate the effectiveness of the proposed constraints and algorithms in reducing capacitive cost of clock trees. (We demonstrate that the proposed techniques can be used to construct clock trees that are robust to OCV in Section 7.) The algorithms are implemented in C++ and the experiments are performed on a 10 core 5.0 GHz Linux machine with 64 GB of memory.

Using the proposed tree construction framework, various different tree structures are constructed. (1) The D-UST structure is a tree structure that is constructed using dynamic implied skew constraints, i.e., the Greedy-UST/DME algorithm in [17]. (2) The PS-UST structure is a tree structure constructed using static useful arrival time constraints where the arrival time constraints for each sink are in the form of a point. (3) The LS-UST structure is a tree structure that is constructed based on static bounded useful arrival time constraints. The range constraints are specified to maximize the lengths lexicographically as in [2, 12]. (4) The S-UST structure is a tree structure that is constructed based on static bounded useful arrival time constraints. The range constraints are specified using the LP formulation in Section 5. (5) The TS-UST structure is the S-UST structure with the additional feature of using rerooting to explore topologies. (6) The RTS-UST structure is equal to the TS-UST structure with the additional feature that the arrival time constraints are re-specified after the synthesis of each buffer stage, as described in Section 6.1.

6.3 Evaluation of various trees structures

In Table 2, we present the results of the various tree structures constructed on the twelve circuits in Table 3, which are available online [7]. The top seven circuits have been used in earlier studies. We compare the performance in terms of the capacitive cost in the column labeled “Cap cost” and the run-time in the column labeled “Run-time”.

No direct comparison is provided with [12]. However, using the LS-UST structure, a direct comparison is provided with the method of specifying the constraints in [2, 12].

The S-UST structures have 5% lower average capacitive cost when compared with the D-UST structures. The lower capacitive costs may stem from that the S-UST structures have relatively aligned arrival time constraints (specified before the tree construction). In the D-UST structure, the arrival times to the sinks may be significantly skewed, as skews are incrementally specified within implied skew constraints.

Compared with the S-UST structures, the PS-UST structures have 48% higher cost on the average, as the arrival time constraints in PS-UST are in the form of points. The LS-UST structures have 37% higher average cost compared with the S-UST structures. This is because the range constraints are specified to maximize the lengths lexicographically [2, 12], instead of considering both length and alignment. The TS-UST structures have 8% lower average capacitive cost when compared with the S-UST structures. This can be understood because the TS-UST structures allows subtrees to be rerooted, facilitating the exploration of various tree topologies. As a greater solution space is explored, clock trees with lower capacitive costs are obtained. Even though the TS-UST structures have a lower utilization of timing margins compared with the D-UST structures, the average capacitive cost is lower because of the topology exploration.

Table 2: Comparisons of various tree structures.

Circuits (name)	Cap cost (pF)						Run-time (min)					
	D-UST	PS-UST	LS-UST	S-UST	TS-UST	RTS-UST	D-UST	PS-UST	LS-UST	S-UST	TS-UST	RTS-UST
scaled_s1423	3.3	4.4	9.9	3.9	3.2	3.2	1	1	1	1	1	1
scaled_s5378	5.7	10.7	9.6	6.3	6.2	5.8	1	3	2	2	20	2
scaled_s15850	18.3	20.5	28.3	20.0	20.0	17.5	16	18	11	20	20	9
msp	1.7	2.5	1.8	1.8	1.5	1.5	1	2	5	1	4	4
fpu	2.1	2.9	2.0	2.0	1.9	1.9	1	1	1	1	4	4
ecg	34.5	50.3	76.4	30.4	28.3	26.9	26	30	64	23	53	63
aes	207.5	372.0	204.4	202.4	207.5	200.7	186	324	114	127	214	155
usbf	8.0	9.9	8.0	5.2	4.5	4.5	4	9	5	3	9	10
dma	7.3	11.9	6.4	5.8	5.3	5.3	4	11	5	3	14	14
pci_bridge32	15.1	15.5	11.2	8.9	7.8	7.7	10	8	10	5	24	24
des_peft	19.2	29.8	44.1	22.7	19.7	18.9	8	14	20	16	36	32
eht	23.6	44.7	23.7	23.3	21.2	21.2	16	25	16	15	72	78
Norm.	1.00	1.48	1.30	0.95	0.87	0.84						

Table 3: Properties of synthesized circuits.

Circuit (name)	Used in	Sinks (num)	Skew constraints (num)
scaled_s1423	[8]	74	78
scaled_s5378	[8]	179	175
scaled_s15850	[8, 10]	597	318
msp	[8]	683	44990
fpu	[8]	715	16263
ecg	[8, 10]	7674	63440
aes	[10]	13216	53382
usbf		1765	33438
dma		2092	132834
pci_bridge32		3578	141074
des_peft		8808	17152
eht		10544	450762

The RTS-UST structures have 3% lower average capacitive cost when compared with the TS-UST structures, as the arrival time constraints are re-specified during the synthesis process to expose additional timing margins after each buffer stage. However, capacitive reductions are only obtained on six out of the twelve circuits. The explanation for this is that the skew constraints have to be relatively stringent for the exposed timing margins to translate into reduction in capacitive cost.

As the arrival time constraints do not have to be updated, the S-UST structures are expected to have shorter run-times when compared with the D-UST structures, the construction of which requires updates of the dynamic implied skew constraints. This is particularly true for larger circuits. Compared with the S-UST structures, the TS-UST structures are expected to have a longer run-times because rerooting is applied to explore various tree topologies. The RTS-UST structures are expected to have shorter run-times compared with the TS-UST structures, as the re-specification of the arrival time constraints may make the tree construction less constrained. Ideally, we would like to control the run-time of the topology exploration more closely. Nevertheless, there are many second order effects that influences the run-time. In particular, a big component of the run-times is related to NGSPICE circuit simulations for guiding the synthesis process.

7. EVALUATION OF ROBUSTNESS TO OCV

In this section, the trees structures constructed by our framework are compared with the clock trees in [8, 10], in terms of timing yield and capacitive cost. The comparison is performed using the Monte Carlo framework proposed in [8], which is an extension of the ISPD 2010 clock contest formulation in [16].

In [10], the clock trees were constructed using a CTS phase and a clock tree optimization (CTO) phase. After an initial

clock tree has been constructed in the CTS phase, some timing violations may still exist. The CTO phase is employed to remove these violations. The optimization is performed by realizing delay adjustments in the tree by inserting buffers and detour wires. The delay adjustments are specified using an LP formulation [14, 15, 9]. For further technical details of the CTO phase, please refer to [14, 15, 9]. To facilitate a fair comparison with [8, 10], we apply CTO [14, 15, 9] to the RTS-UST structures constructed by our framework. Before we present the experimental results, we first describe the evaluation framework.

7.1 Monte Carlo evaluation framework

Each clock tree is evaluated in terms of timing yield and capacitive cost. The timing yield of a clock tree is determined by simulating the clock tree with 500 Monte Carlo simulations. In each simulation, the clock tree is subject to wire width variations ($\pm 5.0\%$), supply voltage variations ($\pm 7.5\%$), temperature variations ($\pm 15.0\%$), and channel length variations ($\pm 5.0\%$) around the nominal values.

Each simulation represents the testing of a chip, if all the skew and transition times are satisfied, the chip is classified as good. If any timing constraint is violated, the chip is classified as defective. The timing yield is defined to be the number of good chips divided by the number of tested chips.

7.2 Evaluation of timing yield and cost

In Table 4, we compare the RTS-UST structures constructed in this work with the D-UST structures constructed in [8], which reported results on six of the twelve benchmark circuits. We also compare against the D-UST structures and LD-UST structures in [10], which reported results on three circuits. The LD-UST structure in [10] is an extension of the D-UST structure in that the structure can meet both skew constraints and a user-specified latency bound at the expense of increased capacitive cost. The normalized capacitive results (labeled “Norm.” in Table 4) are obtained with respect to the capacitive cost of the RTS-UST structures after CTO.

First, we compare the results after CTS (and before CTO). Compared with the RTS-UST structures, the D-UST structures in [8], the D-UST structures in [10], and the LD-UST structures in [10] have 35%, 15%, 16% higher capacitive cost, respectively, which is similar to the results reported for the D-UST structures in Table 2.

After CTO, we observe that the capacitive cost of the RTS-UST structures have only increased by 0.4% on the average (by comparing RTS-UST structures obtained after CTS and after CTO). Therefore, it can be understood that the RTS-UST structures have 43%, 13%, and 16% lower

Table 4: Evaluation of clock trees in timing yield and capacitive cost. A ‘-’ in the CTO run-time column means that CTO is not required to achieve 100% yield.

Circuit	Work	Structure	M_{user}	After CTS				After CTO			
				Cap (pF)	Latency (ps)	Yield (%)	Run-time (min)	Cap (pF)	Latency (ps)	Yield (%)	Run-time (min)
scaled_s1423	[8]	D-UST	0	3.4	140	100.0	1	3.4	140	100.0	-
	this work	RTS-UST	0	3.2	128	100.0	1	3.2	128	100.0	-
scaled_s5378	[8]	D-UST	0	5.7	130	100.0	1	5.7	130	100.0	-
	this work	RTS-UST	0	5.8	205	57.8	2	5.8	205	100.0	1
scaled_s15850	[8]	D-UST	27	20.2	405	96.6	5	20.7	425	99.4	13
	[10]	D-UST	25	17.3	328	81.4	4	17.9	424	81.4	14
	[10]	LD-UST	25	17.7	291	99.2	5	18.1	313	100.0	11
	this work	RTS-UST	25	17.5	244	99.8	9	17.7	256	100.0	14
msp	[8]	D-UST	0	1.9	98	100.0	4	1.9	98	100.0	-
	this work	RTS-UST	0	1.5	89	100.0	4	1.5	89	100.0	-
fpu	[8]	D-UST	0	2.3	87	100.0	2	2.3	87	100.0	-
	this work	RTS-UST	0	1.9	109	93.2	4	1.9	109	100.0	1
ecg	[8]	D-UST	30	66.8	417	98.8	39	75.7	474	91.6	34
	[10]	D-UST	30	35.8	382	99.4	20	36.3	401	99.4	33
	[10]	LD-UST	30	35.0	318	94.6	29	35.2	345	100.0	51
	this work	RTS-UST	30	26.9	234	99.6	63	27.0	247	100.0	32
aes	[10]	D-UST	50	207.5	2207	82.8	245	208.3	2320	97.6	180
	[10]	LD-UST	50	233.9	1863	100.0	133	234.7	1933	99.0	152
	this work	RTS-UST	50	200.7	1172	86.8	155	202.0	1242	96.6	103
usbf	this work	RTS-UST	30	4.5	135	100.0	10	4.5	135	100.0	-
dma	this work	RTS-UST	30	5.3	118	100.0	14	5.3	118	100.0	-
pci_bridge32	this work	RTS-UST	30	7.7	150	100.0	24	7.7	150	100.0	-
des_peft	this work	RTS-UST	30	18.9	148	100.0	32	18.9	148	100.0	-
eht	this work	RTS-UST	30	21.2	144	100.0	78	21.2	144	100.0	-
Norm.	[8]	D-UST		1.36				1.43			
	[10]	D-UST		1.15				1.13			
	[10]	LD-UST		1.16				1.16			
	this work	RTS-UST		0.996				1.00			

capacitive costs compared with the D-UST structures in [8], the D-UST structures in [10], and the LD-UST structures in [10], respectively, after CTO. In addition, even though we do not apply any form of latency optimization, the latencies of the RTS-UST structures are 28% lower compared with the LD-UST structures. We believe that this stems from smaller RTS-UST structures being constructed.

The RTS-UST structures have slightly worse results in yield after CTS (and before CTO). However, in terms of timing yield after CTO, the RTS-UST obtains a 100% yield on all circuits except aes, where a yield of 96.6% is obtained. As mentioned earlier, this improvement is achieved with a 0.4% overhead. Compared with the D-UST structures in [8], the RTS-UST structures obtain better or equal timing yield on all six considered circuits. Compared with the D-UST structures in [10], the RTS-UST structures obtain better timing yield on scaled_s15850 and ecg but slightly worse timing yield on aes. Compared with the LD-UST in [10], the RTS-UST obtains slightly worse timing yield on aes.

Clearly, the RTS-UST structures demonstrate better quality in terms of both capacitive cost and timing yield when compared to the D-UST and LD-UST structures on all circuits except for aes and scaled_s5378. On these two circuits, the RTS-UST structures are only marginally worse.

8. SUMMARY AND FUTURE WORK

In this paper, it is demonstrated that static bounded useful arrival time constraints can be used to construct clock trees meeting useful skew constraints while exploring various topologies. In the future, we plan to extend our framework to consider latency minimization techniques as in [10, 12].

9. REFERENCES

- [1] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing

- analysis for intra-die process variations with spatial correlations. ICCAD’03, pages 900–907, 2003.
- [2] C. Albrecht, B. Korte, J. Schietke, and J. Vygen. Maximum mean weight cycle in a digraph and minimizing cycle time of a logic chip. *Discrete Applied Math.*, 123(1-3):103–127, 2002.
- [3] Y.-C. Chang, C.-K. Wang, and H.-M. Chen. On construction low power and robust clock tree via slew budgeting. ISPD ’12, pages 129–136, 2012.
- [4] Y. P. Chen and D. F. Wong. An algorithm for zero-skew clock tree routing with buffer insertion. EDTC’96, pages 230–237, 1996.
- [5] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao. Bounded-skew clock and steiner routing. *ACM Trans. Des. Autom. Electron. Syst.*, 3(3):341–388, July 1998.
- [6] R. Ewetz, S. Janarthanan, and C.-K. Koh. Fast clock skew scheduling based on sparse-graph algorithms. ASP-DAC ’14, pages 472–477, 2014.
- [7] R. Ewetz, S. Janarthanan, and C.-K. Koh. Benchmark circuits for clock scheduling and synthesis. <https://purr.purdue.edu/publications/1759>, 2015.
- [8] R. Ewetz and C.-K. Koh. A useful skew tree framework for inserting large safety margins. ISPD ’15, pages 85–92, 2015.
- [9] R. Ewetz and C.-K. Koh. MCMC clock tree optimization based on slack redistribution using a reduced slack graph. ASP-DAC ’16, pages 366 – 371, 2016.
- [10] R. Ewetz, C. Tan, and C.-K. Koh. Construction of latency-bounded clock trees. ISPD ’16, 2016.
- [11] J. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, pages 945–951, 1990.
- [12] S. Held, B. Korte, J. Massberg, M. Ringe, and J. Vygen. Clock scheduling and clocktree construction for high performance asics. ICCAD’03, pages 232–239, 2003.
- [13] R.-S. Tsay. Exact zero skew. In *ICCAD’91*, 1991.
- [14] V. Ramachandran. Construction of minimal functional skew clock trees. ISPD’12, pages 119–120, 2012.
- [15] S. Roy, P. M. Mattheakis, L. Masse-Navette, and D. Z. Pan. Clock tree resynthesis for multi-corner multi-mode timing closure. ISPD’14, pages 69–76, 2014.
- [16] C. N. Sze. ISPD 2010 high performance clock synthesis contest: Benchmark suite and results. ISPD’10, pages 143–143, 2010.
- [17] C.-W. A. Tsao and C.-K. Koh. UST/DME: a clock tree router for general skew constraints. *TODAES*, pages 359–379, 2002.