Construction of Latency-Bounded Clock Trees*

Rickard Ewetz, Chuan Yean Tan, Cheng-Kok Koh
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-2035
rewetz,tan56,chengkok@purdue.edu

ABSTRACT

Clock trees must be constructed to function even under the influence of on-chip variations (OCV). Bounding the latency of a clock tree, i.e., the maximum delay from the tree root to any sequential element, is important because the latency correlates with the maximum magnitude of the skews caused by OCV. In this paper, a latency constraint graph (LCG) that captures the latencies of a set of subtrees and the skew constraints between the subtrees is introduced. The minimum latency of a clock tree that can be constructed from the corresponding subtrees is equal to the (negative of the) length of a shortest path in the LCG, which can be computed in $O(VE)$. Based on the LCG, we propose a framework that consists of a latency-aware clock tree synthesis (CTS) phase and a clock tree optimization (CTO) phase to construct latency-bounded clock trees. When applied to a set of synthesized circuits, the framework is capable of constructing latency-bounded clock trees that have higher yield compared to clock trees constructed in previous studies.

Keywords

useful skew; latency; CTS; CTO

1. INTRODUCTION

With increasing impacts of on-chip variations (OCV), it is crucial to consider both timing constraints and latency when constructing clock trees for sequential circuits. Clock skew is the difference in the arrival time of the clock signal between a pair of sequential elements (or clock sinks). A clock tree must be constructed such that the clock signal is delivered meeting skew constraints even when the clock tree is under the influence of OCV. Earlier studies have focused on satisfying the skew constraints by providing guardbands to OCV by inserting safety margins in the skew constraints [15, 8]. However, by inserting safety margins in the skew constraints,

*This research was 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 '16, April 3–6, 2016, Santa Rosa, California, USA.

© 2016 ACM. ISBN 978-1-4503-4039-7/16/04...\$15.00

DOI: <http://dx.doi.org/10.1145/2872334.2872349>

Figure 1: (a) Three subtrees, the respective latencies to the clock sinks, and the skew constraints between the clock sinks. (b) The minimum (maximum) latency to the clock sinks is 65.

the construction process becomes more constrained and the maximum delay from the root of the clock tree to any clock sink (maximum latency or simply latency in this paper) may become longer. This makes the constructed clock tree more susceptible to OCV. Typically, the propagation delay of a path in the clock tree correlates with the delay variations along the path. Therefore, by bounding the latency, the maximum magnitude of the skews caused by OCV is also bounded. Consequently, it is important to consider both safety margins and latency bounds when constructing clock trees.

In [15], a clock tree meeting arbitrary skew constraints (with safety margins inserted) was constructed by iteratively joining two smaller subtrees to form a larger subtree. The skew constraints were satisfied by joining each pair of subtrees within a feasible skew range (FSR). In [8], the Greedy-UST/DME algorithm in [15] was extended to enable larger safety margins to be inserted in the skew constraints. However, latency optimization was not considered in [15, 8]. Latency optimization has been considered indirectly during the construction of a clock tree in [3, 5, 11] and directly during clock tree optimization (CTO) in [12, 13].

The limitation of [12, 13] is that the potential latency reductions may be limited because of the structure of the initial clock tree. The drawback of [15, 8, 3, 5, 11] is that latency is not considered explicitly in the tree construction process. It is important to realize that the minimum latency of a clock tree, constructed from a set of subtrees, is dependent on both the latencies of the subtrees and the skew constraints between the subtrees, as illustrated in Figure 1.

Ideally, clock trees with an optimal latency and safety margin should be constructed. In this paper, we propose a

framework to synthesize clock trees meeting a user-specified latency bound L_{user} while providing a user-specified safety margin M_{user} in the skew constraints. A main contribution is the introduction of a latency constraint graph (LCG). An LCG captures the latencies to the sequential elements, the skew constraints, and the skews committed in a tree construction process. The minimum (maximum) latency of a clock tree that is constructed from the sinks captured in the LCG is equal to the negative value of the length of the shortest path from a virtual source to a virtual sink in an LCG, which we refer to as the *latency path*. Both the latency path and the length of the latency path can be computed using the Bellman-Ford algorithm in $O(VE)$ [6].

The proposed framework consists of a latency-aware clock tree synthesis (CTS) phase and a CTO phase. In the latency-aware CTS phase, the construction of a clock tree is viewed as performing a series of delay insertions and skew commitments. By performing delay insertions and skew commitments within feasible delay insertion ranges (FDIRs) and feasible latency ranges (FLRs), respectively, it is ensured that the length of the *latency path* is bounded, which in turn bounds the latency of the clock tree. After the CTS phase is complete, CTO is performed to remove any remaining timing violations by realizing delay adjustments.

To the best of our knowledge, this is the first work that performs latency minimization while providing safety margin in the skew constraints. Moreover, by committing skews within the intersections of FSRs [15] and an FLRs, both arbitrary skew constraints and the latency bound are satisfied. Therefore, compared to [15, 8], our framework is also capable of constructing clock trees with given latency bounds. Furthermore, we observe that CTO is more effective when applied to clock trees with shorter latencies. Hence, we say that our latency-aware CTS phase constructs clock trees that are amenable to CTO. On a set of synthesized circuits with up to 13,216 sequential elements, we improve the yield to 100%.

2. PRELIMINARIES

2.1 Skew constraints

The clock signal delivered to the sequential elements must meet both setup and hold time constraints. 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 at FF_i and FF_j ; t_i^{CQ} is the clock to output delay of FF_i ; T is the clock period; t_j^H and t_j^S are the hold time and setup time of FF_j , respectively; t_{ij}^{max} and t_{ij}^{min} are the maximum and minimum propagation delays through the combinational logic from FF_i to FF_j , respectively. Using $skew_{ij} = t_i - t_j$, and $l_{ij} = t_j - t_i^{CQ} - t_{ij}^{min}$ and $u_{ij} = T - t_i^{CQ} - t_{ij}^{max} - t_j^S$, Eq (1) and Eq (2) are reformulated as follows, and referred to as skew constraints:

$$l_{ij} \leq skew_{ij} \leq u_{ij}. \quad (3)$$

To satisfy the skew constraints under variations, the constraints are tightened with a uniform safety margin M_{user} as follows:

Figure 2: (a) A clock tree with long latency. (b) A clock tree with short latency. The delay variations in the setup and hold time constraint between FF_i and FF_j are estimated to be 34 ps and 17 ps in (a) and (b), respectively.

$$l_{ij} + M_{user} \leq skew_{ij} \leq u_{ij} - M_{user}. \quad (4)$$

The skew constraints are tightened with a uniform safety margin because it is hard to estimate the impact of OCV before an initial clock tree has been constructed in CTS. The skew constraints with safety margins in Eq (4) can be captured in a skew constraint graph (SCG). In an SCG $G = (V, E)$, the vertices V represent the sequential elements and the edges E represent the skew constraints. The skew constraints with safety margins in Eq (4) are represented with an edge e_{ij} from vertex i to vertex j , with the weight $w_{ij} = -(l_{ij} + M_{user})$ and with an edge e_{ji} from vertex j to vertex i with a weight of $w_{ji} = u_{ij} - M_{user}$. The SCG of the skew constraints in Figure 1(a) is shown with red dashed lines and circles in the middle part of Figure 3.

2.2 On-chip variations

After CTS has been performed, the negative effects of the OCV can be estimated using the timing and topology of the initial clock tree. Next, the estimates are used to evaluate performance and to guide CTO. The constraints in Eq (1) and Eq (2) are extended to include OCV as follows:

$$t_i + t_i^{CQ} + t_{ij}^{max} + t_j^S + \delta_i \leq t_j + T - \delta_j, \quad (5)$$

$$t_i + t_i^{CQ} + t_{ij}^{min} - \delta_i \geq t_j + t_j^H + \delta_j, \quad (6)$$

where δ_i and δ_j are the delay variations caused by OCV. Let the closest common ancestor (CCA) of FF_i and FF_j be denoted as $CCA(i, j)$. The delay variations δ_i and δ_j are only accumulated on the paths from $CCA(i, j)$ to FF_i and FF_j , respectively.

By assuming that the delay variations on a path are proportional to the propagation delay of the path, the delay variations δ_i and δ_j are estimates as follows:

$$\delta_i = c_{ocv} \cdot t_{CCA(i,j),i}, \quad (7)$$

$$\delta_j = c_{ocv} \cdot t_{CCA(i,j),j}, \quad (8)$$

where $t_{CCA(i,j),i}$ and $t_{CCA(i,j),j}$ are the propagation delays from $CCA(i, j)$ to FF_i and FF_j , respectively; c_{ocv} is a user-specified parameter. Consequently, a clock tree with a long latency is more susceptible to OCV variations compared with a clock tree with short latency, as illustrated in Figure 2.

Using the estimates, the total negative slack (TNS) and the worst negative slack (WNS) in the constraints in Eq (5) and Eq (6) can be computed. Note that if the latency of the clock tree L is less than or equal to a latency bound $L \leq \frac{M_{user}}{2 \cdot c_{ocv}}$, it is ensured that $\delta_i + \delta_j \leq M_{user}$ is satisfied

for each pair of flip flops FF_i and FF_j , i.e., $TNS = 0$ and $WNS = 0$. This holds because the delay variations $\delta_i + \delta_j$ can be estimated with $c_{ocv} \cdot t_{CCA(i,j),i} + c_{ocv} \cdot t_{CCA(i,j),j}$ and bounded by $2 \cdot c_{ocv} \cdot L$ because $t_{CCA(i,j),i} \leq L$, and $t_{CCA(i,j),j} \leq L$.

However, it is typically impossible, or very costly, to construct a clock tree meeting such stringent latency bounds. Therefore, we set the user-specified latency bound L_{user} to be a fraction of the latency of a clock tree constructed with no latency bound.

2.3 Problem definition

This paper considers a clock tree synthesis problem, i.e., constructing a clock tree that deliver synchronizing clock signals to sequential elements while satisfying skew constraints and transition time constraints under the influence of OCV, in order to obtain high yield. The objective is to minimize the power consumption of these clock trees.

Two key factors that influence yield are latency and safety margins. Large safety margins implies tighter skew constraints, which restrict the tree construction process and typically results in clock trees that have longer latencies; such clock trees are more susceptible to OCV. Therefore, we approach the synthesis problem by constructing latency-bounded clock trees with uniform safety margin provided in the skew constraints. This problem has two inter-related components: (1) Determining the optimal values for the latency bound and the uniform safety margin, which are denoted as L_{opt} and M_{opt} , respectively. (2) Constructing a clock tree for some L_{opt} and M_{opt} .

We limit the scope of this paper to component (2) of this problem, i.e., constructing a clock tree for some given user-specified latency bounds L_{user} and uniform safety margins M_{user} . We rely on the user to provide appropriate values of L_{user} and M_{user} . In our future work, we will also investigate component (1) of the problem, i.e., how to determine M_{opt} and L_{opt} , and the problem as a whole.

2.4 The Greedy-UST/DME algorithm

Using the Greedy-UST/DME algorithm [15], a clock tree can be constructed meeting the constraints captured in an SCG, i.e., the skew constraints with safety margins in Eq (4). The algorithm constructs a clock tree meeting arbitrary skew constraints by iteratively merging a pair of smaller subtrees to form a larger subtree. In [15], it was shown that if two subtrees were merged within a feasible skew range (FSR), all the skew constraints would be satisfied. An FSR is computed by finding two shortest path in an SCG. Specifically, the FSR between two subtrees i and j is:

$$FSR_{ij} = [-d_{ij}, d_{ji}], \quad (9)$$

where, d_{ij} and d_{ji} are the shortest paths from vertex i to vertex j and the shortest path from vertex j to vertex i in the SCG, respectively. Later, when the merging location is decided and the two subtrees are joined physically, the FSR is narrowed to a specific skew value, committing the skew. The skew commitment requires two edges to be added to the SCG, an edge e_{ij} with a weight of $w_{ij} = -a$ and an edge e_{ji} with a edge weight of $w_{ji} = a$, are added.

3. LATENCY CONSTRAINT GRAPH

We introduce the concept of a latency constraint graph (LCG) as follows. First, we define the term *sink latency*.

Figure 3: The latency path, i.e., the shortest path from the virtual source to the virtual sink has a length of $0 + (-10) + (-15) + (-40) = -65$.

For each respective sink, the *sink latency* is equal to the delay from the root of a subtree where the sink resides to the respective sink. An LCG $G = (V, E)$, is an extension of an SCG with the addition of a virtual source and a virtual sink. An LCG for the subtrees in Figure 1(a) is shown in Figure 3. We refer to the original vertices of an SCG as sink vertices. As illustrated in Figure 3, the virtual source is connected to each of the sink vertices with a directed edge with a weight of 0. Moreover, each sink vertex is connected to the virtual sink with an directed edge with a weight equal to the negative value of the respective *sink latency*.

Latency path: We call the shortest path from the virtual source to the virtual sink the *latency path* and the length of the latency path is denoted as s^p . The minimum (maximum) latency of a clock tree that is constructed from the sinks captured in the LCG is equal to $-s^p$. Both the latency path and the length of the latency path can be found in $O(VE)$ using the Bellman-Ford algorithm [6]. Consequently, the subtrees in Figure 1 can be joined to a clock tree with a latency of 65 because $-s^p = 65$ in Figure 3.

To find a clock tree with the latency of $-s^p$, we introduce the reversed graph $\bar{G}$ of the LCG G . In the reversed graph $\bar{G}$, all the edges in G are reversed, i.e., an edge from vertex i to vertex j in G is equivalent to an edge from vertex j to vertex i in $\bar{G}$. Moreover, the top-down shortest path from the virtual source to a vertex i in G is denoted s_i^t and the bottom-up shortest path from the virtual sink to a vertex i in $\bar{G}$ is denoted s_i^b . Interestingly, a clock tree constructed such that the latency to each sink is equal to the bottom-up shortest path to the respective sink vertex satisfies both the latency requirement and the skew constraints. Consequently, in the example, the latency to the sequential elements in such a clock tree would be $[-s_1^b, -s_2^b, -s_3^b] = [65, 55, 40]$.

3.1 Updating edge weights in LCG

The construction of a clock tree can be viewed as a series of delay insertion and skew commitment operations that correspond to both physical commitments in a tree construction process and an equivalent modifications of edge weights in the LCG and its reversed graph. The two operations are outlined as follows:

Delay insertion: This is the process of inserting a piece of wire or a buffer at the root of a subtree. In the LCG, the delay insertion increases the latency to all the sinks of residing in the subtree, which necessitates a reduction of the corresponding latency edge weights. (Without loss of generality, every subtree can be captured using a single rep-

representative sink. Therefore, a delay insertion is equivalent to the reduction of a single edge weight.)

Skew commitment: This is the process of joining two subtrees i and j to a single subtree. As in an SCG, if a $skew_{ij} = a$ is committed, two edges are required to be added to the LCG, an edge e_{ij} with a weight of $w_{ij} = -a$ and an edge e_{ji} with a edge weight of $w_{ji} = a$. Again, the added edge weights always correspond to a reduction of the edge weights.

In summary, both the delay insertion and skew commitment operations result in reductions of edge weights in the LCG. Moreover, the reductions may reduce the length of the latency path. To ensure that all subtrees can be joined to clock tree meeting a user-specified latency bound L_{user} , we introduce a feasible delay insertion range (FDIR) and a feasible latency range (FLR). If each delay insertion or skew commitment is performed within an FDIR or an FLR, respectively, $-s^p \leq L_{user}$ is ensured, and consequently the latency bound is satisfied.

3.2 Derivation of FDIRs and FLRs

Consider a delay insertion or a skew commitment operation that reduces the edge weight w_{ij} to $\hat{w}_{ij}$. Assume that before the operation, $-s^p \leq L_{user}$ is satisfied. By reducing the edge weight w_{ij} , the latency path can only be reduced if the edge e_{ij} is part of the latency path. The shortest path from the virtual source to the virtual sink using the edge e_{ij} is denoted s_{ij}^p and equal to $s_{ij}^p = s_i^t + w_{ij} + s_j^b$. To meet the latency bound constraint, after the edge weight reduction, $-s_i^t - \hat{w}_{ij} - s_j^b \leq L_{user}$ is required to be satisfied. This implies a constraint on $\hat{w}_{ij}$, which is:

$$-\hat{w}_{ij} \leq L_{user} + s_i^t + s_j^b. \quad (10)$$

The maximum delay insertion Δ_i at the root of a subtree i is equal to the slack in Eq (10) before the edge weight reduction, i.e., $L_{user} + s_i^t + s_j^b + w_{ij}$. Consequently, the $FDIR_i$ for a delay insertion Δ_i is formulated as follows:

$$FDIR_i = [0, L_{user} + s_i^t + w_{i,vsink}], \quad (11)$$

where $w_{i,vsink}$ is the weight of the edge from the sink vertex i to the virtual sink before the delay insertion, and s_i^t is the top-down shortest path to the sink vertex i and $s_j^b = 0$ since vertex j corresponds to the virtual sink.

Next, we formulate the FLR_{ij} for a skew commitment $skew_{ij}$. Recall that a skew commitment results in the addition of two edges e_{ij} and e_{ji} with weights $\hat{w}_{ij} = -skew_{ij}$ and $\hat{w}_{ji} = skew_{ij}$, respectively. Using the constraints in Eq (10), FLR_{ij} is obtained as follows:

$$FLR_{ij} = [-L_{user} - s_j^t - s_i^b, L_{user} + s_i^t + s_j^b]. \quad (12)$$

In Eq (12) s_i^t and s_j^b are the shortest paths to the sink vertices i and j with the edges e_{ij} and e_{ji} removed from G and $\bar{G}$. This modification is required because both edge weights w_{ij} and w_{ji} are reduced with the same skew commitment.

Finally, to satisfy both skew constraints and latency bound, a concept of a feasible skew-latency range (FSLR) is introduced, which is the intersection of an FSR in Eq (9) and an FLR in Eq (12) as follows:

$$FSLR_{ij} = FSR_{ij} \cap FLR_{ij}. \quad (13)$$

It can be proved that if both FSR and FLR are non-empty, FSLR is also non-empty. In short, the statement holds be-

Figure 4: Flow for latency-aware CTS.

cause the bottom-up shortest path to each sink vertex defines a latency assignment to the sinks that is by definition a part of both the FLR and FSR.

Note that the use of FDIRs and FLRs is orthogonal to the use of FSR in [15]. Consequently, if the latency bound is loose, our proposed approach is equivalent to the Greedy-UST/DME algorithm [15].

4. PROPOSED SYNTHESIS FLOW

In this section, we present the proposed framework for synthesizing latency-bounded clock trees. The framework consists of a latency-aware CTS phase and a CTO phase. The innovations proposed in the paper focus on the latency-aware CTS phase. The CTO phase is an adaptation of algorithms proposed in previous studies.

4.1 Latency-aware CTS phase

The latency-aware CTS is based on a traditional bottom-up tree construction flow that constructs a clock tree buffer stage by buffer stage, alternating between merging subtrees [15] and inserting buffers [4, 2, 8]. This “core flow” is illustrated with solid boxes in Figure 4 and consists of “merging” and “buffer insertion”.

Merging: The process is based on the Greedy-UST/DME algorithm [15] using a nearest neighbour graph (NNG). In an NNG, each subtree is represented with a vertex and each edge represents the wiring cost of merging two subtrees. Iteratively the two subtrees connected with the least cost edge are attempted to be merged within an FSR. If the two subtrees connected with the least cost edge can be merged within an FSR while meeting a transition time constraint, the two subtrees are replaced by a larger subtree, which is reinserted into the NNG. Otherwise, the two subtrees are locked from further merging. After all subtrees are locked from further merging, a buffer is inserted at the root of each of the subtrees in the buffer insertion step.

Buffer insertion: A minimally sized buffer that can drive each respective locked subtree while meeting the transition time constraint is inserted at the root of the respective subtree, as in [4, 2, 8].

Assume that a set of subtrees has been constructed from the “merging” and “buffer insertion” processes. In addition, assume that the roots of these subtrees are located at the same spatial location. In such a situation, the LCG can be utilized to merge subtrees into a clock tree with a latency of $-s^p$ and the latency to each of the sequential elements would be equal to (the negative of) the bottom-up shortest path to

each respective sink vertex in the LCG. Consider subtrees in Figure 1(a) with the LCG in Figure 3, one way to construct such a clock tree would be to insert the delay difference between the final latencies [65, 55, 40] and the current latencies [5, 20, 40], i.e., $[65, 55, 40] - [5, 20, 40] = [60, 35, 0]$, at the root of the respective subtrees. Note the required delay insertions can be shared among the subtrees as illustrated in Figure 1(b). This method of joining the subtrees, based on the LCG, is referred to as *root construction*; it is labeled as “root construction” in Figure 4 and is further explained in Section 4.1.1.

However, to utilize the root construction, all subtrees are required to be located at the same spatial location. To satisfy this requirement, a root container is introduced. (In our implementation, the root container is located at the center of the circuit.) After the “buffer insertion” step in the core flow, it is checked, for every subtree, whether the subtree can reach the root container by inserting a stem wire below the newly inserted buffer. (A stem wire is a wire that is connected between the root of a subtree and the respective driving buffer [4]). If so, the subtree is routed to and placed in the root container and removed from the core flow. Otherwise, the subtree continues to be part of the iterative merging and buffer insertion process. After all subtrees in the core flow are located in the root container, the proposed root construction is applied.

Next, we propose to supplement the core flow with a latency-aware subtree construction technique. This technique is to overcome the limitation that $-s^p \leq L_{user}$ may not be satisfied at the beginning of the root construction. Therefore, to ensure that $-s^p \leq L_{user}$ holds at the beginning of the root construction, we propose to incorporate techniques called “Merging based on FSLRs”, “Latency locking based on FDIRs”, and “Subtree dragging based on FDIRs”, which are illustrated in Figure 4 and explained in Section 4.1.2.

4.1.1 Root construction

The input to the root construction is a set of subtrees located at the same spatial location (in a root container). The output is a clock tree with the latency to each sequential element equal to the (negative value of) the bottom-up shortest path to each sink vertex. We explain the root construction in Figure 5, by illustrating how the subtrees in Figure 1(a), with the LCG in Figure 3, are joined to form a clock tree in Figure 1(b), with a latency of $-s^p = 65$.

In Figure 3, the (negative of the) bottom-up shortest paths are [65, 55, 40]. To realize these latencies from the current latencies [5, 20, 40] of the subtrees, delay insertions $\Delta = [65, 55, 40] - [5, 20, 40] = [60, 35, 0]$ are required. A naive solution would be to realize each of the delay insertions separately, i.e., a delay $\Delta_1 = 60$ and $\Delta_2 = 35$ could be added to the subtrees 1 and 2 respectively, before joining the three subtrees at the root.

Instead, we propose a method of maximal sharing of the required delay insertions. The delay insertions Δ are sorted in a decreasing order, i.e., [60, 35, 0]. This sorted order defines both the merging order and the sharing of the delay insertions. The subtrees are merged from left to right corresponding to the sorted order of the delay insertion, i.e., subtree 1 is merged with subtree 2 and the resulting subtree is merged with subtree 3. The delay insertions are shared with all subtrees located to the left in the sorted delay inser-

Figure 5: Maximal sharing of delay insertions.

tion array, i.e., the delay 35 is shared among both subtree 1 and 2. By using the maximal sharing of the delay insertions, we expect the cost to be reduced compared with the case when root construction is not applied. Moreover, yield may also improve since the sinks will be closer in the tree topology.

Furthermore, the delay insertions Δ are realized *imperfectly* to simplify root construction process. We restrict the delay realization to be performed by buffer insertion. Therefore, the delay insertion granularity is limited by the buffer library. Here, instead of realizing each delay insertion precisely, we attempt to realize each delay insertion as close to but less than or equal to, the specified delay insertion as possible. The motivation to realizing delays imperfectly is that the CTO process can realize more precise delay adjustments after CTS is complete. Since the timing and the topology of the initial clock tree is available during CTO, delay adjustments can be realized with a finer granularity while considering more accurate OCV estimates.

It is expected that the imperfect root construction may affect the yield after CTS. If inadequate safety margins are provided in a single skew constraint, the constructed clock tree may suffer substantial yield loss. However, after CTO, the yield should be recovered.

4.1.2 Latency-aware subtree construction

The root construction step is capable of joining the subtrees to a clock tree with a latency of $-s^p$. The techniques in this section are designed to ensure that $-s^p \leq L_{user}$ is satisfied at the beginning of the root construction.

It is straightforward to only allow subtrees to be merged within FSLRs introduced in Eq (13), ensuring that $-s^p \leq L_{user}$ is satisfied after two subtrees are merged. However, with only that extension of the core construction flow, subtrees with long latencies that are spatially distant from the root container may be created. Moreover, when these subtrees are routed to the root container, the constraint $-s^p \leq L_{user}$ may be violated.

To account for the spatial location of a subtree, the concept of *virtual subtree latency* is introduced. The virtual sink latency of a subtree is equal to the delay that is inserted if a subtree is routed (or dragged) to the root container by connecting a wire from the root of the subtree to the location of the root container (with buffers inserted on the wire to control the transition time). Next, the latency edge weights are modified to be the negative value of the sum of the sink latency and the virtual subtree latency.

With the inclusion of virtual latencies, a clock tree with a latency of L_{user} can be constructed by dragging each subtree to the root container and then joining them at the root

container. When dragging the subtrees to the location of the root container, the virtual latency would be replaced with “real” latency and the weight of each latency edge would remain the same. Consequently, the length of the latency path s^p would also remain the same, and a clock tree with a latency of $-s^p \leq L_{user}$ can be constructed. However, routing each of the subtrees (or sinks) to the location of the root container directly would be very costly, in terms of wire and buffer resources. Therefore, we do not drag subtrees to the root container directly but instead continue to merge subtrees while ensuring that $-s^p \leq L_{user}$. Only after a subtree has grown beyond a threshold, it is dragged to the root container. The core flow is modified as follows:

Merging based on FSLRs: The “Merging” in the core flow is modified to merge subtrees within FSLRs in Eq (13) instead of FSRs. By merging subtrees within FSLRs it is ensured that both skew constraints and the latency bound are satisfied.

Subtree locking based on FDIRs: If a subtree has long latency or is distant from the root container, it may be costly to merge it with another subtree. Therefore, we lock such subtrees from further merging and drag them towards the root container in the subtree dragging step. A subtree is locked if the upper bound of its FDIR (see Eq (11)) is less than a parameter $p_{lock} = 20$ ps. This condition is checked in the merging process after each new subtree is formed.

Subtree dragging based on FDIRs: The subtree dragging is applied after the buffer insertion step. It is applied to every subtree whose upper bound of its FDIR (see Eq (11)) is less than a parameter $p_{drag} = 20$ ps. The driving buffer is sized up to the next driver size and a piece of stem wire is inserted between the buffer and the subtree. The stem wire is elongated to the maximum length while satisfying a transition time constraint. Next, the buffer is dragged as close to the root container as possible using the stem wire, transferring virtual latency to real latency.

4.2 Clock tree optimization (CTO) phase

The CTO phase is based on the techniques proposed in [9]. The optimization aims to remove timing violations (or negative slacks, i.e., TNS and WNS) in a clock tree by realizing non-negative delay adjustments. The location and the magnitude of the delay adjustments are determined by solving an LP formulation. Next, the delay adjustments are realized by inserting buffers and detouring wires [10].

Note that delay insertions that are realized imperfectly during the root construction can be realized as delay adjustments during CTO process, if required. Moreover, it is important to observe that if the latency of the initial clock tree is long, large delay variations are present in many skew constraints and there may be little room for CTO to improve performance. Therefore, we say that by bounding the latency of a clock tree during latency-aware CTS, the clock tree is more amenable to CTO.

5. EXPERIMENTAL EVALUATION

The proposed algorithms are implemented in C++ and the experiments are performed on a quad core 3.10 GHz Linux machine with 7.7 GB of memory.

To evaluate our proposed framework, the extension [8] of the problem formulation in the ISPD 2010 contest [14] is used. A summary of the properties of the synthesized circuits that are used in the evaluation are shown in Table 1 [7].

Table 1: Properties of synthesized circuits.

Name	Clock period (ns)	Number of nets	Number of cells	Number of sequential elements	Number of skew constraints
scaled_s15850	0.32	-	-	597	318
ecg	1.00	62164	61491	7674	63440
aes	1.00	262884	262497	13216	53382

In [8], a Monte Carlo framework is used to evaluate the robustness of a clock tree subject to voltage (15%), wire width (10%), temperature (30%), and channel length variations (10%). The variations are generated using a quad-tree model [1] to exhibit spatial correlation. The robustness of a clock tree is measured by simulating it with 500 Monte Carlo simulations. In each simulation, it is checked whether all the skew constraints (see Eq (5) and Eq (6)) and a transition time constraint are satisfied. The transition time states that the 10% to 90% rise or fall time of the clock signal at any point in the clock tree must be below 100 ps. The quality of a clock tree is measured in terms of yield. The yield is defined to be the number of simulations with no constraint violations divided by the total number of simulations. The power consumption is estimated with capacitive cost.

To clearly demonstrate the impact of each optimization step we show the performance and cost of the constructed clock trees after both CTS and CTO. Even though the evaluation is performed in terms of yield, we show the TNS and WNS estimates that are used to guide the CTO process. It is expected that if the TNS and WNS are zero, or small, the yield of the clock tree will be high when evaluated by the Monte Carlo framework. However, there is no guarantee that a clock tree with zero TNS and WNS will have 100% yield or that a clock tree with non-zero TNS and WNS will suffer yield loss. The parameter $c_{ocv} = 0.085$ is used based on statistics obtained through circuit simulations.

We construct and evaluate several different tree structures to show the impact of our proposed optimization techniques. The structure “Tree” is a clock tree constructed using only the traditional core flow, i.e., using merging and buffer insertion. The structure “R-Tree” is the core flow with the addition of the root construction. The structure “L-R-Tree”, is the structure obtained by the complete flow, i.e., the core flow with the latency-aware subtree construction techniques and the root construction. After the different tree constructions have been performed, CTO is performed. We also report the results of the most competitive clock trees constructed in [8]. The construction process of clock trees in [8] is similar to that of our “Tree” structure. However, CTO is not performed in [8]. To enable a more fair comparison, we perform CTO on the “Tree” structures in [8].

5.1 Evaluation of root construction

Table 2 shows the experimental results on the circuits scaled_s15850, ecg, and aes. We construct the “Tree” structures and “R-Tree” structures with different safety margins M_{user} and compare the performance after CTS and after CTO. We have also included relevant results from [8].

We observe the same trends as in [15, 8] for the “Tree” structures constructed with different safety margins M_{user} after CTS. The performance in TNS, WNS, and yield is improved when larger safety margins are inserted in the skew constraints. For example, on ecg, a yield of 92.8% (97.4% and 99.6%) is obtained when a safety margin of 20 ps (30 ps and 40 ps) is inserted. However, as larger safety margins are

Table 2: Evaluation of root construction after CTS and after CTO.

Circuit	Structure	M_{user} (ps)	After CTS						After CTO					
			Latency (ps)	Cap (pF)	TNS (ps)	WNS (ps)	Yield (%)	Run time (min)	Latency (ps)	Cap (pF)	TNS (ps)	WNS (ps)	Yield (%)	Run time (min)
scaled_s15850	Tree	20	299	16.3	-196	-20	95.0	1	324	16.7	-156	-20	95.0	1
	R-Tree	20	273	16.1	-262	-26	97.2	4	311	16.5	-218	-29	63.4	10
	Tree	25	424	17.7	-329	-25	96.4	4	424	18.4	-239	-25	34.0	13
	R-Tree	25	303	17.3	-291	-30	80.4	1	313	17.7	-212	-19	97.6	1
	Tree in [8]	27	405	20.2	-150	-19	96.6	5	425	20.7	-72	-13	99.4	13
	Tree	27	328	19.0	-329	-18	99.8	5	376	20.2	-187	-17	99.6	20
	R-Tree	27	291	18.1	-200	-21	94.0	5	329	19.0	-93	-13	100.0	17
ecg	Tree in [8]	20	420	57.0	-4577	-24	82.6	29	427	57.6	-9342	-46	0.0	272
	Tree	20	344	32.4	-7061	-24	92.8	18	392	36.4	-9443	-31	23.2	201
	R-Tree	20	317	30.9	-1983	-16	75.6	28	348	33.4	-3085	-18	63.0	119
	Tree in [8]	30	417	66.8	-1569	-19	98.8	39	474	75.7	-1874	-20	91.6	341
	Tree	30	454	40.2	-2849	-21	97.4	15	447	44.8	-4122	-37	86.0	233
	R-Tree	30	382	35.8	-205	-6	99.4	20	401	36.3	-24	-3	99.4	33
	Tree in [8]	40	716	96.8	-245	-15	96.2	106	742	98.8	-158	-10	98.6	109
	Tree	40	850	62.4	-1890	-18	99.6	33	862	68.2	-2140	-26	99.0	293
	R-Tree	40	811	58.2	-419	-16	94.2	69	854	58.4	-3	-2	99.8	45
aes	Tree	40	1954	180.4	-24073	-57	93.4	139	2003	212.1	-52775	-56	0.0	895
	R-Tree	40	1756	160.8	-5116	-29	75.4	116	1880	164.0	-561	-12	96.2	254
	Tree	50	2468	228.8	-2036	-43	91.4	189	2474	264.2	-15317	-38	78.2	883
	R-Tree	50	2207	207.5	-2877	-27	82.8	245	2320	208.3	-13	-3	97.6	180
	Tree	60	2706	270.2	-823	-20	96.4	193	2708	273.1	-117	-7	97.6	137
	R-Tree	60	2397	245.9	-1779	-32	71.4	220	2482	246.3	0	0	98.0	199
Norm.	Tree in [8]		1.17	1.62	2.83	1.68	1.03	1.40	1.14	1.65	46.58	5.03	0.73	3.96
	Tree		1.15	1.07	3.49	1.44	1.13	1.44	1.11	1.14	243.52	5.56	0.76	4.81
	R-Tree		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

inserted, the construction process becomes more constrained and it can be observed that the capacitive cost and latency increase from 32.4 pF and 344 ps to 62.4 pF and 850 ps, respectively.

Next, to demonstrate the importance of considering latency during CTS, we discuss the difference in performance between the “Tree” and the “R-Tree” structures. Both structures are constructed based on a user-specified safety margin M_{user} . However, the “R-Tree” structures also try to minimize latency (although no latency bound is required).

After CTS, it can be observed that the “R-Tree” structures have 15% shorter latency and 7% lower capacitive cost compared with the “Tree” structures on the average. The latency improvements are a consequence of using the LCG to join the subtrees in the root container to a single clock tree. Moreover, it is likely that the reductions in capacitive cost mostly stem from the the sharing of delay insertions during root construction, and because of the imperfect root construction.

The “R-Tree” structures also outperform the “Tree” structures in terms of TNS and WNS. This is expected because the delay variations introduced by OCV are smaller in clock trees with shorter latencies. However, on a few circuits, the “R-Tree” structures obtain worse TNS and WNS; we believe this is because of the imperfect delay insertion during root construction. For example, on circuit aes, with $M_{user} = 60$ ps, the “R-Tree” structure has worse TNS and WNS.

Even though the “Tree” structures have worse performance in TNS and WNS, it is expected that the yield in the Monte Carlo evaluation is better compared with the “R-Tree” structures. This is because if a single skew constraint is violated as a result of the imperfect root construction for the “R-Tree” structures, yield loss is suffered. (However, we expect this yield loss to be recovered after CTO.)

Next, we focus our attention on the experimental results obtained after CTO. For the “Tree” structures, it can be observed that on many circuits, CTO is unable to improve the performance in TNS and WNS. As mentioned, this may be because the “Tree” structures have long latencies and there is no room for further optimization. On circuits where CTO is unable to improve or close timing in terms of TNS and WNS, it can be understood that the yield may be worse after CTO because the safety margins in the skew constraints may have been redistributed unevenly, and inadequate safety margins in a single skew constraint may result in substantial yield loss.

On the other hand, the CTO process is capable of removing or reducing TNS and WNS when applied to the “R-Tree” structures. Therefore, we say that our framework constructs clock trees that are more amenable to CTO. Moreover, the capacitive overhead of performing CTO is small on the average; only an 2.4% increase in capacitive cost is observed. Since significant reductions in TNS and WNS are obtained, it is not surprising that the yield of the “R-Tree” structures is higher after CTO compared with after CTS. Moreover, we also note that the “R-Tree” structures have higher yield after CTO compared with the “Tree” structures after both CTS and CTO. This confirms the importance of considering both latency and safety margins during CTS. Nevertheless, if the initial clock tree is constructed with a too small safety margin M_{user} , CTO may not be able to achieve timing closure, and the yield may be inadequate (See $M_{user} = 20$ ps and 20 ps for scaled_s15850 and ecg, respectively.)

5.2 Evaluation of latency-bounded clock trees

We are interested in improving yield on the circuits, ecg, aes, and scaled_s15850. As observed in Table 2, the “R-Tree” structures performed better compared with the “Tree” structures mainly because of their shorter latencies. There-

Table 3: Evaluation of latency-bounded clock trees after CTS and CTO.

Circuit	Structure	L_{user} (ps)	After CTS						After CTO					
			Latency (ps)	Cap (pF)	TNS (ps)	WNS (ps)	Yield (%)	Run time (min)	Latency (ps)	Cap (pF)	TNS (ps)	WNS (ps)	Yield (%)	Run time (min)
scaled_s15850 $M_{user} = 25$ ps	R-Tree	∞	303	17.3	-291	-30	80.4	4	321	17.9	-182	-20	81.4	14
	L-R-Tree	270	248	17.7	-96	-15	99.2	5	267	18.1	-45	-6	100.0	11
	L-R-Tree	250	214	18.3	-78	-10	100.0	3	229	18.8	-30	-7	100.0	14
	L-R-Tree	230	194	22.3	-85	-13	99.8	6	227	22.7	-48	-10	100.0	12
ecg $M_{user} = 30$ ps	R-Tree	∞	382	35.8	-205	-6	99.4	20	401	36.3	-24	-3	99.4	33
	L-R-Tree	380	318	35.0	-345	-16	94.6	29	345	35.2	-4	-1	100.0	51
	L-R-Tree	360	325	39.1	-823	-12	99.8	26	343	40.4	-260	-10	100.0	67
	L-R-Tree	340	265	39.3	-741	-16	100.0	34	303	40.7	-245	-12	100.0	64
aes $M_{user} = 50$ ps	R-Tree	∞	2207	207.5	-2877	-27	82.8	245	2320	208.3	-13	-3	97.6	180
	L-R-Tree	2000	1863	233.9	-1492	-18	100.0	133	1933	234.7	-13	-2	99.0	152
	L-R-Tree	1800	1638	234.1	-2064	-29	100.0	281	1706	236.6	-119	-4	98.2	294
	L-R-Tree	1600	1483	237.5	-1347	-18	100.0	129	1513	240.0	-187	-5	99.6	175

fore, we speculate that we may be able to improve performance by combining the use of safety margins and latency bounds. Moreover, potentially, by imposing a latency bound, a smaller safety margin M_{user} can be used, which may translate into savings in capacitive cost. In Table 3, we construct three versions of the “L-R-Tree” structure on each circuit, each with the same safety margin M_{user} but a different latency bound L_{user} . The safety margin M_{user} is set to 25 ps, 30 ps, and 50 ps because these safety margins seem to provide a reasonable starting point in yield and capacitive cost (see the relevant results for “R-Tree” in Table 2, which are repeated in Table 3).

In Table 3, we observe that the framework is capable of constructing clock trees meeting specified latency bounds. For example, on ecg and aes, the “R-Tree” structures have latencies of 382 ps and 2207 ps, respectively; the “L-R-Tree” structures have latency ranges of 265 ps to 318 ps and 1483 ps to 1863 ps, respectively. Moreover, it is observed that it is not too costly to reduce the latency quite substantially. The capacitive overheads are at most 15%.

We observe that the performance in TNS and WNS after CTS is inconclusive. On some circuits and for certain latency bounds, the performance of the “R-Tree” structures is better and for others, “L-R-Tree” structures have better performance. This may be because of the imperfect root construction that is applied in the construction of both the “R-Tree” and the “L-R-Tree”. However, after CTO, the “L-R-Tree” structures seem to obtain better performance in TNS and WNS, if a L_{user} is set to an appropriate value.

Moreover, the yield of the “L-R-Tree” structures is better compared with the “R-Tree” structures. On scaled_s15850 and ecg, the yield is better after CTO compared with after CTS, as expected. However, on aes, it is observed that the yield is actually better after CTS compared with CTO, even though the performance is better in TNS and WNS after CTO. We believe that this may be because the performance in TNS and WNS does not correlate perfectly with yield performance. However, we still believe that TNS and WNS are useful metrics for optimization.

Based on the results in Table 2 and Table 3, it seems essential to combine the use of safety margins and latency bounds to be able to construct clock trees with high yield. Only the “L-R-Tree” is capable of obtaining a yield of 100% (after CTS or CTO) on all the circuits. Moreover, the “L-R-Tree” structures are cheaper in terms of capacitance compared to

the clock trees constructed with larger safety margins M_{user} in Table 2.

6. SUMMARY AND FUTURE WORK

We have introduced the concept of an LCG that captures, skew constraints, latencies, and skews committed in a synthesis process. Based on the LCG we have proposed a framework that constructs latency-bounded clock trees given a user-specified latency bound L_{user} and a user specified uniform safety margin M_{user} . In the future, we will solve component (1) of the problem defined in Section 2.3, i.e., determining appropriate M_{user} and L_{user} , and the problem as a whole.

7. 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] S. Bujimalla and C.-K. Koh. Synthesis of low power clock trees for handling power-supply variations. ISPD ’11, pages 37–44, 2011.
- [3] R. Chaturvedi and J. Hu. Buffered clock tree for high quality ic design. In *Quality Electronic Design, 2004. Proceedings. 5th International Symposium on*, pages 381–386, 2004.
- [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] Y.-Y. Chen, C. Dong, and D. Chen. Clock tree synthesis under aggressive buffer insertion. DAC’10, pages 86–89, June 2010.
- [6] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Ed., 2001.
- [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. MCMM clock tree optimization based on slack redistribution using a reduced slack graph. ASP-DAC’16, pages 366–371, 2016.
- [10] D.-J. Lee, M.-C. Kim, and I. L. Markov. Low-power clock trees for cpus. ICCAD’10, 2010.
- [11] A. Rajaram and D. Z. Pan. Variation tolerant buffered clock network synthesis with cross links. ISPD’06, pages 157–164, 2006.
- [12] V. Ramachandran. Construction of minimal functional skew clock trees. ISPD’12, pages 119–120, 2012.
- [13] S. Roy, P. M. Mattheakis, L. Masse-Navette, and D. Z. Pan. Clock tree resynthesis for multi-corner multi-mode timing closure. *CAD of IC and Sys.*, pages 589–602, 2015.
- [14] C. N. Sze. ISPD 2010 high performance clock network synthesis contest: Benchmark suite and results. ISPD’10, pages 143–143, 2010.
- [15] C.-W. A. Tsao and C.-K. Koh. UST/DME: a clock tree router for general skew constraints. (TODAES), pages 359–379, 2002.