An OCV-Aware Clock Tree Synthesis Methodology

Necati Uysal and Rickard Ewetz
Department of Electrical and Computer Engineering
University of Central Florida, Orlando, FL, USA
necati@knights.ucf.edu, rickard.ewetz@ucf.edu

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643585

***Abstract*—Closing timing after clock tree synthesis (CTS) is very challenging in the presence of on-chip variations (OCVs). State-of-the-art design flows first synthesize an initial clock tree that contains timing violations introduced by OCVs. Next, aggressive clock tree optimization (CTO) is applied to eliminate the timing violations. Unfortunately, it may be impossible to eliminate all violations given the structure of the initial clock tree. In this paper, we propose an OCV-aware clock tree synthesis methodology that aims to rethink how to account for OCVs. The key idea is to predict the impact of OCVs early in the synthesis process, which allows the variations to be compensated for using non-uniform safety margins. This results in a synthesis flow that is almost correct-by-design. In contrast, state-of-the-art design flows often have an unpredictable success rate because the OCVs are considered too late in the synthesis process. Concretely, this is achieved by top-down constructing a virtual clock tree that is refined bottom-up into a real clock tree implementation. To balance the quality of results (QoR) and runtime, multiple top-level tree topologies are enumerated and pruned in the synthesis process. Compared with the CTO based approach, the experimental results demonstrate that the proposed methodology reduces the total negative slack (TNS) and worst negative slack (WNS) with 90% and 75%, respectively.**

***Index Terms*—On-chip variations, clock network synthesis**

I. Introduction

Closing timing after clock tree synthesis (CTS) is one of the most challenging design steps in EDA flows for synchronous VLSI circuits. It often requires costly manual intervention in the form of engineering change orders (ECOs) [2], [6], which elongates turnaround time (TAT) and time-to-market (TTM). The problem is exacerbated in aggressively scaled technology nodes that are severely impacted by on-chip variations (OCVs). The variations are introduced by the intrinsic variability in the semiconductor fabrication processes and fluctuations in the environmental operating conditions. Clock networks are naturally vulnerable to variations because they span across the entire chip. The synthesis of clock networks satisfying timing constraints under OCVs has been the focus of many studies [4], [5], [7], [8], [11], [17].

Early studies on CTS focused on the construction of zero skew trees (ZSTs) and bounded skew trees (BSTs) [4], [5]. More recently, the construction of useful skew trees (USTs) was investigated to meet non-uniform skew constraints [8], [11], [17]. The synthesis of USTs is motivated by that the skew constraints imposed between pairs of sequential elements (or clock sinks) are non-uniform. While constructing clock trees meeting non-uniform skew constraints under nominal conditions is relatively easy [4], [5], [8], [11], [17], it is tremendously difficult to synthesize clock trees that satisfy timing constraints under the influence of OCVs. The challenge stems from that OCVs introduce delay variations, which cause unwanted clock skew between all pairs of clock sinks. Moreover, the magnitude of the introduced skew is dependent on the distance between the clock sinks in the tree topology [13]. Pairs of clock sink that are distant (close) in the topology is more (less) susceptible to OCVs.

Techniques of handling OCVs by inserting safety margins before (or during) the clock tree synthesis process have been explored in [10], [12], [17]. The insertion of uniform safety margins was investigated in [10], [17]. The limitation of that approach is that the required safety margins depend on the clock tree topology and are therefore non-uniform. Consequently, the use of non-uniform safety margins results in that many timing constraints will have excessive (or inadequate) timing margins inserted, which translates into substantial hardware overheads (or timing violations). To reduce the overheads, the magnitude of the safety margins can be tailored to the clock tree topology during the synthesis process [7], [12]. Unfortunately, these techniques still result in clock trees with timing violations [12] or unacceptable overheads [7].

The state-of-the-art methodology for synthesizing clock networks consists of a clock tree synthesis (CTS) phase and a clock tree optimization (CTO) phase. In the CTS phase, an initial clock tree is first constructed. Next, the impact of the OCVs and the associated timing violations are determined. Aggressive CTO is subsequently applied to eliminate all timing violations [6], [14]. The CTO process is based on first specifying delay insertions on the branches of the clock tree. Next, the delay insertions are realized physically by inserting delay buffers or detour wires. The delay insertions improve timing by redistributing timing margins from satisfied to unsatisfied timing constraints. While CTO is capable of significantly improving the timing quality of most clock trees, there is no guarantee that the optimization process will converge to a solution without timing violations. In particular, it may be impossible to close timing if the quality of the initial clock tree is poor. Advanced CTO techniques have recently been investigated to solve this challenge. The insertion of negative delay adjustments based on buffer sizing was proposed in [19]. The reconstruction of the clock tree topology with the objective of minimizing latency or placing

This research was in part supported by NSF awards CCF-1755825, CNS-1908471, and CNS-2008339.

certain clock sinks close in the topology was explored in [2], [15], [18]. Nevertheless, CTO flows are still time consuming and often require costly manual intervention. This stems from that the state-of-the-art design flows consider the impact of OCVs too late in the synthesis process.

In this paper, we propose an OCV-aware clock tree synthesis methodology that aims to rethink how to account for OCVs. The key idea of the methodology is to predict the impact of the OCVs early in the synthesis process. This will allow the OCVs to be compensated for using non-uniform safety margins during the initial tree construction. The goal is to only leverage CTO to eliminate minor timing violations arising from modeling errors, which results in that the synthesis flow is almost correct-by-design. In contrast, the state-of-the-art methodologies account for OCVs using aggressive CTO, which results in unpredictable optimization and timing results.

The proposed OCV-aware clock tree synthesis methodology consists of a top-down phase and bottom-up phase. In the top-down phase, numerous top-level tree topologies are enumerated and pruned. Next, a virtual clock tree is formed to estimate the timing and variations within every timing constraint. Subsequently, non-uniform safety margins are inserted to account for the variations. In the bottom-up phase, each virtual clock tree is refined into a real clock tree by constructing subtrees that connect the clock sinks to the top-level tree. If the constructed subtrees meet a set of latency constraints imposed by the virtual clock tree, the timing constraints are guaranteed to be satisfied by design. Otherwise, the virtual clock tree is updated with improved timing predictions and the process is iteratively repeated. The experimental results demonstrate that the proposed OCV-aware synthesis flow reduces the average total negative slack (TNS) and worst negative slack (WNS) by 90% and 75%, respectively. Moreover, the run-time of the proposed flow is 47% shorter when compared to the CTO based flow.

The remainder of the paper is organized as follows: Preliminaries are given in Section II. Motivation is highlighted in Section III. Methodology is explained in Section IV. Experimental results are presented in Section V. The paper is concluded with the summary and future works in Section VI.

II. Preliminaries

A synchronizing clock signal is delivered to each sequential element in a sequential circuit. There is a setup and hold time constraint between each pair of flip-flops (FF) that are only separated by combinational logic in the data and control paths. The setup and hold time constraints between the launching flip-flop FF_i and the capturing flip-flop FF_j are formulated as follows:

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

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

where t_i and t_j are the arrival times of the clock signal to the FF_i and FF_j, respectively. t_i^{CQ} is the clock to output delay of the capturing flip-flop; T is the clock period; t_j^S and t_j^H are the setup time and hold time of FF_j, respectively. t_{ij}^{max} (t_{ij}^{min}) is the maximum (minimum) delay through the combinational logic between FF_i and FF_j. δ_i and δ_j are the timing deteriorates introduced by OCVs.

By replacing the $T - t_i^{CQ} - t_{ij}^{max} - t_j^S$ and $t_j^H - t_i^{CQ} - t_{ij}^{min}$ respectively with the u_{ij} and l_{ij}, Eq (**??**) and Eq (2) can be reformulated into explicit skew constraints, as follows:

$$l_{ij} + (\delta_i + \delta_j) \leq t_i - t_j \leq u_{ij} - (\delta_i + \delta_j) \quad (3)$$

The timing deteriorates δ_i and δ_j are dependent on the distance between FF_i and FF_j in the clock tree topology. A clock tree containing the flip-flops FF_i and FF_j is illustrated in Figure 1. Let the closest common ancestor (CCA) between FF_i and FF_j in the clock tree be denoted CCA(i,j) [13]. Based on the model in [15], δ_i and δ_j are equal to $c_{ocv} \cdot t_{CCA(i,j),i}$ and $c_{ocv} \cdot t_{CCA(i,j),j}$, respectively. $t_{CCA(i,j),i}$ and $t_{CCA(i,j),j}$ are the propagation delays from the CCA(i,j) to FF_i and FF_j, respectively. c_{ocv} is a parameter determined by circuit simulations.

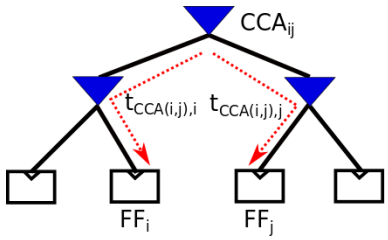

Fig. 1. The illustration of the path that introduces OCV into the timing constraints between FF_i and FF_j.

III. Motivation

In this section, we highlight the limitations of the previous works and provide an overview of the proposed methodology.

A. Limitations of previous works

The state-of-the-art synthesis methodology for clock networks is shown in Figure 2(a). As the OCVs depend on the timing and topology of the clock tree, uniform safety margins are first inserted into the timing constraints. Next, an initial clock tree is constructed. Given the topology and timing of the initial clock tree, the negative impact of OCVs and the associated timing violations are calculated. Finally, aggressive CTO is iteratively applied to eliminate timing violations. If the timing cannot be closed, expensive manual feedback is performed using ECOs. It is not surprising that ECOs are commonly required because the initial clock tree was constructed without accounting for OCVs.

B. Proposed methodology

The flow of the proposed OCV-aware clock tree synthesis methodology is shown in Figure 2(b). The methodology consists of a top-down phase and a bottom-up phase. In the top-down phase, a top-level tree is constructed. (In the proposed framework, numerous top-level tree topologies are enumerated to explore a larger solution space.) Next, a virtual clock tree

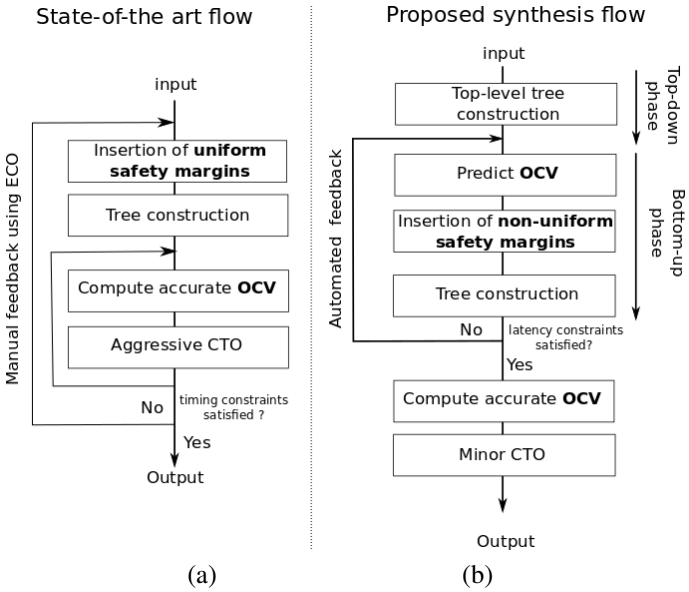

Fig. 2. (a) State-of-the-art CTS+CTO based synthesis flow. (b) Proposed OCV-aware clock tree synthesis flow.

is constructed based on the top-level tree to estimate the non-uniform safety margin required in each timing constraint. In the bottom-up phase, subtrees are constructed to connect the clock sinks to the top-level tree. If the subtrees meet the constraints imposed by the virtual tree, the timing is satisfied by design. Otherwise, the timing and impact of OCVs within the virtual tree are updated, and the bottom-up phase is repeated. As the flow accounts for the impact of OCVs early in the synthesis process, there is no need for costly manual intervention using ECOs. The subsequent CTO is only used to eliminate minor violations introduced by modeling errors[1].

TABLE I
HOLISTIC GUIDELINES AND OBJECTIVES FOR DIFFERENT PARTS OF A CLOCK TREE.

Part of Topology	Tree properties		Objective	Construction method
	Timing & latency	Hardware cost		
Top-level	Majority	Minority	Minimize latency	Top-down
Bottom-level	Minority	Majority	Minimize cost	Bottom-up

The proposed flow is inspired by the holistic guidelines for clock tree synthesis shown in Table I. A clock tree can be decomposed into a top-part and a bottom-part. The table shows that the top-part of a clock tree stands for a very small portion of the total capacitance and hardware overheads. At the same time, it defines the overall timing of the clock tree. In contrast, the bottom part of the clock tree accounts for a majority of the hardware resources and capacitance. On the other hand, it has a limited impact on the timing. Therefore, we speculate that it would be advantageous to construct the top-part of the clock tree first to define the timing and the negative impact of the OCVs. While fixating the top-level tree (a restriction) may introduce some hardware overheads, the overheads are expected to be small as the cost of the top-part of the clock tree is minor compared with the bottom-part. Next, the bottom-part of the clock tree can be constructed in a cost efficient manner bottom-up. The main challenge to this approach is that it is difficult to select the ideal top-level tree. The larger the specified top-level tree is, the more predictable the OCVs become. On the other hand, the introduced overheads become larger when the top-level tree is larger. Consequently, designs with tight (loose) timing constraints require a larger (smaller) top-level tree. To solve this challenge, the methodology enumerates and prunes out several different top-level tree topologies.

[1]The construction of clock trees is traditionally guided by less accurate delay models that can be evaluated quickly. In contrast, CTO is commonly performed using computationally expensive delay models that are more accurate. We refer to the difference in accuracy between the two models as modeling errors.

IV. METHODOLOGY

In this section, we provide the details of the proposed OCV-aware clock synthesis methodology.

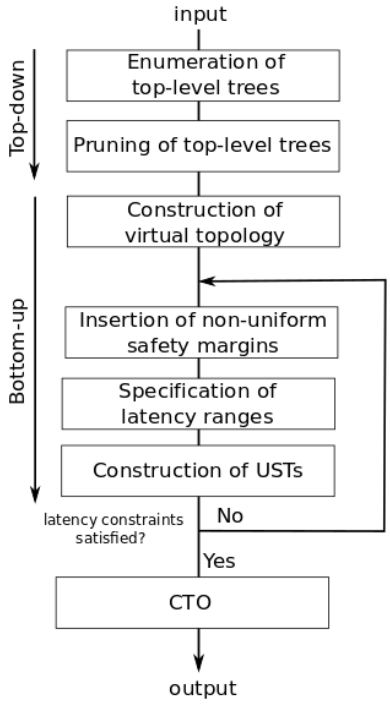

Fig. 3. Detailed flow of proposed OCV-aware synthesis methodology.

A. The overview of the framework

The flow of the proposed methodology is shown in Figure 3 and illustrated with an example in Figure 4. The flow consists of a top-down phase and a bottom-up phase. In the top-down phase, a top-level tree is constructed by enumerating and pruning top-level tree topologies, which is shown in (a) and (b) of Figure 4. The details are provided in Section IV-B

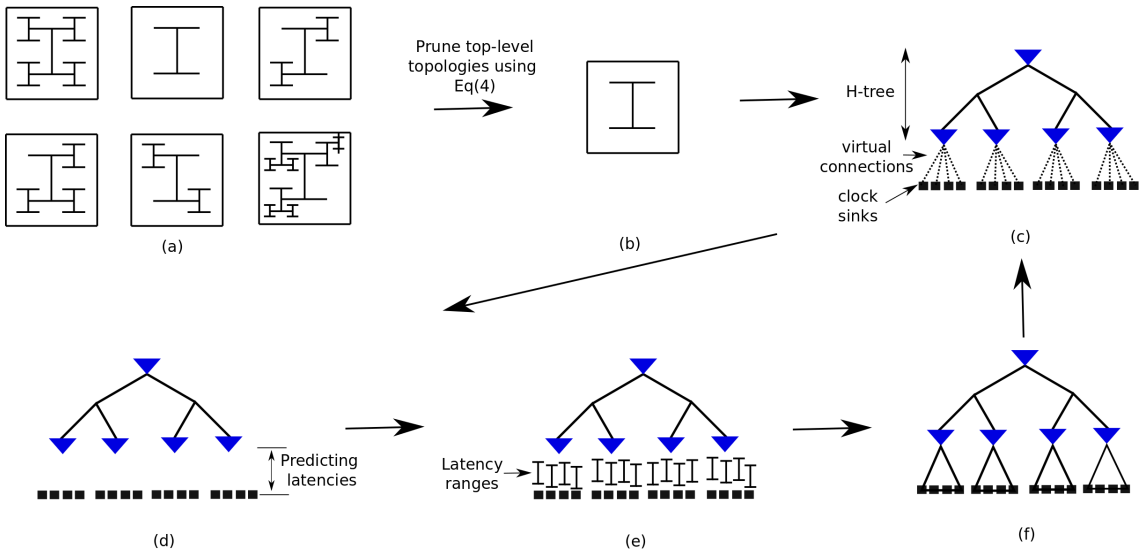

Fig. 4. (a) Multiple candidate top-level topologies are enumerated. (b) The top-level topology is determined after pruning the candidate topologies. (c) The top-level clock tree and the virtual topology are constructed. (d) The latencies to the clock sinks are estimated using the virtual topology. (e) Non-uniform safety margins are inserted in the skew constraints and a latency range is specified for each clock sinks. (f) Bottom-level subtrees are iteratively constructed by inserting safety margins that are tailored to the topology. If the latency constraints are not satisfied, the construction process returns to (c) with improved timing predictions.

and Section IV-C. In the bottom-up phase, a virtual clock tree is first formed based on the top-level tree, which is shown in Figure 4(c) and detailed in Section IV-D. Next, the timing and OCV impact is estimated based on the virtual tree. The estimates are used to insert non-uniform safety margins in the timing constraints, as shown in Figure 4(d) and explained in Section IV-E. Next, the timing constraints are converted into latency constraints, which is shown in Figure 4(e) and explained in Section IV-F. A latency constraint is a lower and upper bound on the arrival time of the clock signal to a clock sink. The conversion is motivated by that it is typically easier to construct clock trees based on latency constraints than skew constraints. Finally, subtrees are constructed bottom-up to connect the clock sinks to the leaf nodes of the top-level tree as shown in Figure 4(f) and Section IV-G.

After bottom-level subtrees have been constructed, we compute the maximum latency to each clock sink. If all the latency constraints are satisfied, the synthesis process has converged and timing is closed in the presence of OCVs. Otherwise, the synthesis process returns to the insertion of non-uniform safety margins step. However, the timing of the virtual tree is updated with timing from the constructed subtrees. Hence, the timing and the impact of the OCVs can be computed more accurately. In our future work, we plan to explore adjusting the topology of the top-level tree based on feedback from the construction process.

B. Enumeration of top-level trees

In this section, it is explained how different top-level trees are enumerated. The top-level trees in this paper are in the form of uniform and non-uniform H-trees [1]. While any type of top-level trees could be used, we select H-trees because they have short latency, which in turn results in smaller OCVs. The H-trees divide the die into multiple rectangular regions. Each leaf node of the top-level tree is expected to drive all the clock sinks in the corresponding region.

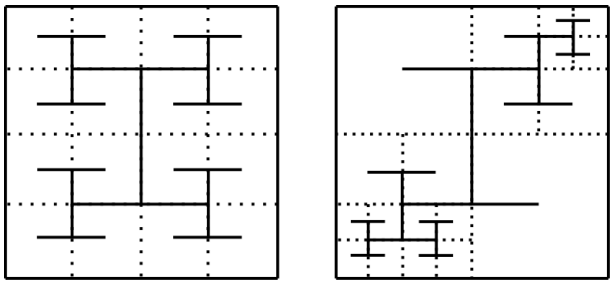

Fig. 5. (a) Uniform H-tree (b) Non-uniform H-tree

In the proposed framework, all possible uniform and non-uniform top-level H-trees with a maximum of three levels are enumerated. A uniform H-tree have the same amount of levels from the root node to each of the leaf node, which divides the die into a set of uniform regions as shown in Figure 5(a). On the other hand, a non-uniform H-tree divides the die into a set of non-uniform regions by constructing irregular level of H-trees, which is illustrated in Figure 5(b). In our framework, we primarily use non-uniform H-trees to better adapt the top-level tree to the properties of the input circuit. Note that we consider uniform H-trees are considered a subset of non-uniform H-trees. We also limit the maximum level of an H-tree to be three. The maximum level was selected to balance a trade-off between runtime and solution quality.

C. Pruning of top-level trees

In this section, we prune down the enumerated top-level tree topologies into a single top-level tree topology. An alternative

is to keep multiple promising topologies to improve the solution space exploration at the expense of longer synthesis time. The pruning is performed by defining a cost metric for each region. Next, the smallest top-level tree that satisfies a cost constraint on each region is selected.

The cost metric attempts to limit the size of the subtrees that will be connected to each leaf node of the top-level tree. If the subtrees become too large, the magnitude of the introduced OCVs between different sink pairs will be large, requiring excessive safety margins. Consequently, it is advantageous to limit the size of the subtrees. There are several features that determine the size of the subtree constructed from the clock trees within a region of the die. We observe that the most prevalent features are: i) the number of clock sinks, ii) the total sink capacitance, and iii) the region die area. Therefore, we define the cost metric, as follows:

$$cost = \alpha \cdot N_{sink} + \beta \cdot C_{total} + \gamma \cdot A_{die} \tag{4}$$

where N_{sink} is the total number of clock sinks in a region; C_{total} is the total downstream capacitance at the inputs of clock sinks in a region; A_{die} is the area of a region. α, β and γ are the parameters to balance different terms in the cost function.

D. Construction of virtual topology

In this section, it is explained how the virtual clock tree topology is formed. The root node of each region is located at the mid-point of a region and corresponds to the root node of a bottom-level subtree. To obtain the virtual clock tree topology, first the clock sinks that are located at the same region are clustered. Next, the virtual clock tree topology is formed by inserting a virtual connection from each clock sink to its corresponding root node. The virtual clock tree topology that is obtained after inserting the virtual connections is shown in Figure 4(c).

E. Insertion of non-uniform safety margins

In this section, we detail how the non-uniform safety margins are inserted in the timing constraints. The magnitude of the OCVs in each timing constraint depends on the timing and topology of the clock tree. In our framework, we utilize the virtual clock tree to predict the magnitude of the OCVs. Next, we insert the appropriate non-uniform safety margins.

The timing of the clock tree and the latencies to the clock sinks are first estimated using the virtual clock tree, which is shown in Figure 4(c). Let T_h and T_k be the leaf node of the H-tree that is located at the mid-point of region h and k, respectively. The latency on the path from $CCA(h,k)$ to the clock sinks in region h and k are respectively estimated as follows:

$$\begin{aligned} l_{CCA(h,k),h} &= l_{CCA(T_h,T_k),T_h} + l_h \\ l_{CCA(h,k),k} &= l_{CCA(T_h,T_k),T_k} + l_k \end{aligned} \tag{5}$$

where $l_{CCA(T_h,T_k),T_h}$ is the propagation delay on the path from $CCA(T_h,T_k)$ to T_h; $l_{CCA(T_h,T_k),T_k}$ is the propagation delay from $CCA(T_h,T_k)$ to T_k. l_h and l_k are the average latency of the bottom-level subtrees in the virtual clock tree that are constructed in region h and k, respectively. The average latency is used because the bottom-most subtrees are constructed without any internal restrictions on the topology. Next, the safety margin $M_{(h,k)}$ is inserted in the timing constraints between clock sinks in region h and k, as follows:

$$M_{(h,k)} = c_{ocv} \cdot (l_{CCA(h,k),h} + l_{CCA(h,k),k}) \cdot c_s \tag{6}$$

where c_s is a user specified parameter that can be used to scale the inserted safety margins. The default value for the parameter c_s is 1. Finally, the safety margins are inserted into the skew constraints by replacing the δ_i and δ_j terms in Eq (3) with $M_{(h,k)}$ in Eq (6) as follows:

$$\begin{aligned} l_{i,j} + M_{(h,k)} \leq t_i - t_j \leq u_{i,j} - M_{(h,k)} \\ i \in h, j \in k \end{aligned} \tag{7}$$

The latencies $l_{CCA(h,k),h}$ and $l_{CCA(h,k),k}$ are obtained using SPICE simulations of the top-level tree. The average subtree latencies l_h and l_k are estimated to be zero in the first iteration. In subsequent iterations, the average latency of the subtree constructed in the previous iteration is used. The latencies of the subtrees are also obtained using SPICE simulations.

F. Specification of latency ranges

In this section, we explain how to specify latency constraints that ensure that the skew constraints are satisfied. A set of latency constraints consist of a latency range for each clock sinks. A latency range specifies a lower and upper bound on the arrival time of the clock signal to a clock sink. Skew constraints specify a restriction on the relative arrival time of the clock signal in between a pair of clock sinks. The motivation for converting the skew constraints into latency constraints is that it is easier to construct clock trees with respect to latency constraints [8]. The specification of the latency constraints is performed by capturing the skew constraints using a skew constraint graph (SCG). Next, a latency range is specified for each clock sink by formulating and solving an LP problem.

An SCG captures the skew constraints with non-uniform safety margins in Eq (7). In an SCG $G = (V, E)$, V is the set of clock sinks and E is the set of skew constraints. Let the flip-flops FF_i and FF_j be represented by vertices i and j, respectively. For each constraint in Eq (7), edges e_{ij} and e_{ji} are inserted to the SCG. Let $w_{ij} = u_{i,j} - M_{(h,k)}$ be the weight of edge e_{ij} and $w_{ji} = -l_{i,j} - M_{(h,k)}$ be the weight of edge e_{ji}. Next, we formulate the LP formulation in [8] to specify latency ranges, as follows:

$$\begin{aligned} &\min c_{tns} \cdot P_{tns} + c_{wns} \cdot P_{wns} + \sum_{i \in V} c_r f(x_i^{ub}, x_i^{lb}) \\ &x_i^{lb} \leq x_i^{ub}, \forall i \in V \\ &x_i^{ub} - x_j^{lb} - P_{ij} \leq w_{ij}, \quad \forall (i,j) \in E \\ &P_{tns} = \sum_{(i,j) \in E} P_{ij} \\ &P_{wns} \geq P_{ij}, \end{aligned} \tag{8}$$

x_i^{lb} and x_i^{ub} are the lower bound and upper bound of latency range for sink i, respectively. $f(.)$ is a piece-wise linear function that aims to maximize the length of the latency ranges. More details about the piece-wise linear function is in [8]. P_{ij} is the term that captures the timing violation (negative slack) in the skew constraint between FF_i and FF_j. P_{tns} and P_{wns} denote the predicted total negative slack and predicted worst negative slack, respectively. c_r, c_{tns} and c_{wns} are respectively the parameters to balance the weights of latency range, total negative slack and worst negative slack in the objective. In our framework, c_{tns} and c_{wns} are specified to be significantly larger than c_r to minimize timing violations.

G. Construction of USTs

In this section, it is explained how subtrees are constructed bottom-up to connect the clock sinks to the top-level tree. A separate subtree construction is performed for the clock sinks assigned to each leaf node in the top-level tree. The subtrees are constructed using the useful skew tree construction algorithm in [8] that is based on the BST construction in [4]. Both algorithms are based on the deferred-merge embedding (DME) paradigm, where smaller subtrees are iteratively merged to create larger subtrees.

The BST construction in [4] is based on keeping track of the minimum and maximum delay to a clock sink within a subtree. This makes it easy to merge (or join) two subtrees while meeting a skew bound B. The generalization to UST construction in [8] is based on converting the latency ranges into a virtual minimum and maximum delay offset for each clock sink. Next, USTs can be constructed while only checking that each subtree satisfies a skew bound B. Please refer to [8] for the technical details of specifying the virtual delay offsets. The BST construction is based on the DME paradigm and is performed exactly as in [4].

The DME algorithm is based on iteratively merging the pair of subtree (or clock sinks) that requires minimum wirelength to be joined while meeting a skew bound B under the Elmore delay model. The transition time is evaluated after each pair of subtrees have been merged. If the transition time constraint is violated, subtrees are unmerged and locked for further merging operations. Merging operations are performed until all the subtrees are locked. When all the subtrees are locked, a buffer with the minimum size that can drive each subtree is inserted to the root of each respective subtree. Next, a stem wire is inserted between the buffer and the subtree [3]. The merging and buffer insertion operations are repeatedly performed until there is one single subtree left. Lastly, a top-down embedding phase is performed to determine the exact locations of internal nodes within each subtree.

After all the clock sinks have been connected to the top-level tree, each subtree is simulated using SPICE simulations. If the latency constraints are satisfied, the flow converges. Otherwise, the framework returns to the insertion of non-uniform safety margins in Section IV-E. The latency constraints are required to be checked because the UST construction only guarantees that the clock signal is delivered within latency ranges with an arbitrary offset [8].

V. Experimental Evaluations

The proposed framework is implemented in C++ and evaluations are performed using a quad-core 3.4 GHz Linux machine with 32GB of memory. The properties of buffers and wires are obtained from a 45nm technology library in [16]. The clock tree methodology is evaluated using the benchmarks in [9]. The details of each benchmark circuit is shown in Table II. The timing of the constructed clock trees are evaluated with circuit simulations using NGSPICE. The c_{ocv} parameter is set to be 0.085 in our framework.

TABLE II
THE PROPERTIES OF BENCHMARK CIRCUITS IN [9]

Circuit (name)	Sinks (num)	Skew constraints (num)
s15850	597	318
usbf	1765	33438
ecg	7674	63440
dma	2092	132834
pci bridge32	3578	141074
des peft	8808	17152

The structures are evaluated in terms of capacitance and timing quality. The timing quality is evaluated using TNS and WNS. The performance of the proposed framework is evaluated by constructing three different clock tree structures. The UST-U-CTS structure is a useful skew tree constructed with uniform safety margins inserted in the skew constraints [7], [17]. The structure is obtained by performing multiple tree constructions with uniform safety margins of different magnitude. The safety margin configuration that achieves the best timing quality is selected as the UST-U-CTS structure. The UST-U-CTO structure is obtained by applying CTO [6] to the UST-U-CTS structure. The UST-N structures are clock trees obtained using the proposed OCV-aware methodology.

In Section V-A, the configurations within the framework are evaluated. In Section V-B, the proposed methodology is compared with a state-of-art synthesis flow. The evaluation is performed using the aforementioned tree structures.

A. Evaluation of framework configurations

In this section, we evaluate the algorithm design and parameter configurations within the proposed methodology.

The iterative subtree construction is evaluated using the dma circuit in Figure 6. The evaluations are performed by comparing the performance of each constructed clock tree within the iterative construction process.

The predicted latency and the actual latency of a bottom-level subtree with respect to the iteration are shown in Figure 6(a). It can be observed that the gap between the predicted latency and the actual latency is gradually reduced until the predicted latency is smaller than the actual latency, which results in that the flow converges. In Figure 6(b), we evaluate the capacitance of the constructed clock tree. It can

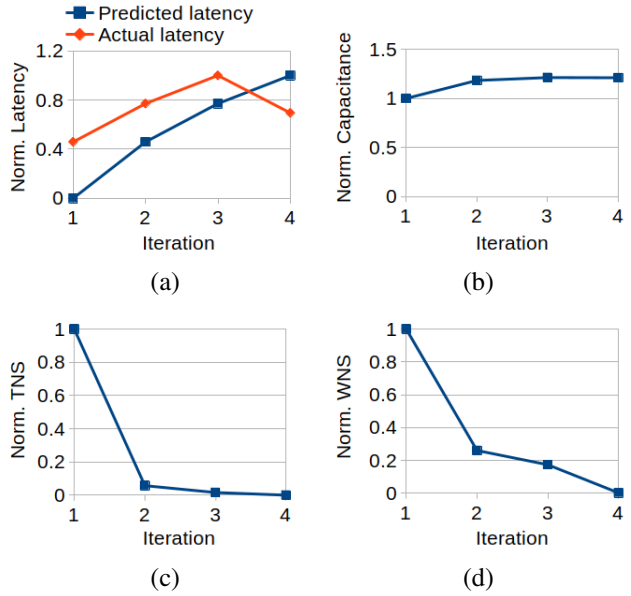

Fig. 6. Evaluation of the framework in terms of (a) Latency. (b) Capacitance. (c) TNS. (d) WNS.

be observed that the capacitance of the clock tree increases with each iteration. This is easy to understand because the framework predicts larger and larger latencies, which results in that gradually larger non-uniform safety margins are inserted in the skew constraints. Intuitively, larger safety margins constrain the tree construction more, which translates into the synthesis of clock trees with higher capacitive overheads. Next, we evaluate the timing quality of the constructed clock trees using TNS and WNS in (c) and (d) of Figure 6, respectively. The figures show that the timing quality of the constructed clock tree is improved with each iteration. This stems from that better latency predictions are performed, which results in that safety margins with adequate magnitude are specified. The figure shows that the framework converges to a clock tree structure without timing violations after four iterations, which validates the effectiveness of the proposed framework.

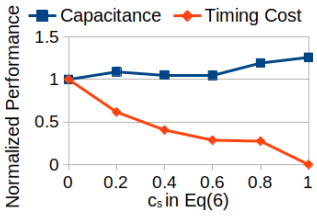

Fig. 7. Sensitivity to scaling parameter c_s on ecg.

We analyze the sensitivity of the framework to the c_s parameter from Eq (6) in Figure 7. The performance of the framework is evaluated in terms of total clock tree capacitance and the timing cost on the ecg benchmark circuit. The timing cost is the combined cost that consists of both TNS and WNS. The costs are weighted using c_{tns} and c_{wns} in Eq (6). The figure shows that the timing cost of the clock tree reduces with the increase of the c_s parameter. On the other hand, the capacitance increases when the c_s increases. This stems from that larger safety margins are specified when c_s is set to be larger, which constrains the clock tree construction.

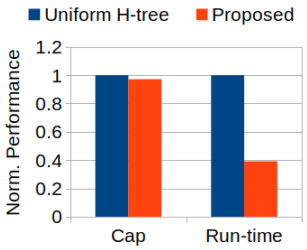

Fig. 8. Evaluation of top-level tree construction using uniform H-tree vs. non-uniform H-tree.

We evaluate the top-level tree construction based on uniform and non-uniform H-trees in Figure 8. The top-level tree construction technique using only uniform H-tree is based on constructing 1-level, 2-level, 3-level H-trees and selecting the structure that provides the best performance in terms of timing quality. The proposed top-level tree construction is based on enumerating and pruning candidate topologies as explained in Section IV-B and Section IV-C. The comparison is performed in terms of run-time and total capacitance on the $s15850$ benchmark circuit. As it can be observed in Figure 8, the proposed technique results in 61% shorter run-time without any capacitive overhead. The improvement in the run-time stems from that the clock tree can be constructed by performing a single synthesis process using the proposed technique, while multiple clock trees are required to be synthesized to select the uniform H-tree that performs best.

B. Comparisons with state-of-the-art

In Table III, we evaluate the performance of the three clock tree structures in terms of total capacitance, timing quality and run-time. We evaluate the total capacitance of the structures because it is well known to be highly correlated with the power consumption. The capacitance breakdown of the constructed clock trees are reported in the 'Capacitance' column. The timing quality is evaluated using latency, total negative slack (TNS) and worst negative slack (WNS). The latency, TNS and WNS of the structures are respectively reported in the columns of 'Latency', 'TNS' and 'WNS'. The minimum, average and maximum safety margins that are used to construct the UST-U-CTS and UST-N structures are reported in the 'Safety margin' column. The run-time of the synthesized clock trees are reported in the 'Runtime' columns. The run-times of the UST-U-CTS structure and the UST-N structure in the table are the run-times of individual synthesis steps. The run-time of the UST-U-CTO structure is the cumulative run-time, i.e., the sum of the run-time of both the UST-U-CTS structure and the CTO process.

The normalized performance in terms of total capacitance, latency and run-time are normalized with respect to the UST-U-CTO structure. The TNS and WNS performance of the structures are normalized with respect to the UST-U-CTS structure. Note that all the synthesized clock trees in Table III meet the transition time constraint.

TABLE III
EVALUATION OF THE CLOCK TREE STRUCTURES IN TERMS OF PERFORMANCE AND SYNTHESIS TIME.

Benchmark	Structure	Safety margin			Capacitance			Latency	TNS	WNS	Run-time
		min (ps)	avg (ps)	max (ps)	Buffer (pF)	Wire (pF)	Total (pF)	(ps)	(ps)	(ps)	(min)
s15850	UST-U-CTS [7], [17]	40	40	40	5.4	14.8	23.1	374.0	66.1	14.8	1.7
	UST-U-CTO [6]	'-'	'-'	'-'	6.6	15.0	24.6	378.5	31.8	6.1	36.3
	UST-N	11.9	17.2	29.0	4.7	9.5	17.3	311.8	0.0	0.0	6.9
usbf	UST-U-CTS [7], [17]	30	30	30	1.9	5.1	8.0	214.1	37.0	11.4	2.9
	UST-U-CTO [6]	'-'	'-'	'-'	2.4	5.1	8.5	230.3	0.0	0.0	21.9
	UST-N	9.7	17.7	24.4	3.2	4.4	8.7	228.8	0.0	0.0	11.8
ecg	UST-U-CTS [7], [17]	35	35	35	7.7	22.4	34.8	267.1	685.4	18.9	44.4
	UST-U-CTO [6]	'-'	'-'	'-'	11.2	23.1	39.0	283.7	31.7	3.6	416.7
	UST-N	20.5	26.6	35.5	13.3	24.1	42.1	333.4	25.4	3.0	165.1
dma	UST-U-CTS [7], [17]	20	20	20	1.8	4.5	7.6	180.2	144.9	8.1	5.3
	UST-U-CTO [6]	'-'	'-'	'-'	2.6	4.6	8.5	180.2	25.9	4.4	29.8
	UST-N	8.8	18.1	20.7	3.2	4.4	8.9	173.5	1.0	0.3	32.2
pci	UST-U-CTS [7], [17]	30	30	30	2.5	6.5	11.2	267.1	38.9	7.3	7.3
	UST-U-CTO [6]	'-'	'-'	'-'	3.0	6.5	11.8	276.1	2.8	0.7	67.9
	UST-N	9.4	18.0	24.1	3.8	6.2	12.2	218.6	1.1	0.6	40.2
des	UST-U-CTS [7], [17]	30	30	30	7.4	21.3	34.2	206.4	614.8	15.0	44.8
	UST-U-CTO [6]	'-'	'-'	'-'	10.2	21.8	37.5	241.0	6.9	0.7	259.8
	UST-N	15.8	17.6	28.7	10.4	19.0	34.9	236.3	1.7	0.5	91.1
Norm	UST-U-CTS [7], [17]						**0.92**	0.95	1.00	1.00	**0.12**
	UST-U-CTO [6]						1.00	1.00	0.13	0.21	1.00
	UST-N						0.97	**0.95**	**0.01**	**0.05**	**0.53**

Compared with the UST-U-CTS structure, the UST-U-CTO structure have 87% and 79% lower TNS and WNS, respectively. The improvements in the timing quality stem from that CTO is applied to remove the timing violations that are obtained from the UST-U-CTS structure. The improvements in timing quality come at the expense of increasing the capacitance and the latency respectively by 8% and 5%. This is expected because CTO inserts buffers and wires into the clock tree to realize the delay adjustments, which introduces overheads in terms of total capacitance. The run-time of the UST-U-CTO is 88% longer because CTO requires performing a vast amount of timing analysis using SPICE simulations.

Next, we compare the performance of the UST-U-CTO structure with the UST-N structure. The table shows that the UST-N structure has 90% and 75% lower TNS and WNS, respectively. The improvements in the timing quality stem from synthesizing the clock tree by inserting non-uniform safety margins that are tailored to the topology. It is promising that the capacitance and the latency of the UST-N structure are respectively 3% and 5% lower when compared to the UST-U-CTO structure. This stems from that the UST-U-CTO structure is constructed by inserting safety margins that are larger than required for certain skew constraints, which results in constructing larger clock trees. On the other hand, the UST-N structure inserts only the required amount of safety margins that are tailored to the topology, which are reported in Table III. The run-time of the UST-N structure is 48% shorter when compared to the UST-U-CTO structure. The run-time improvements stem from that the UST-N structure is obtained by synthesizing a single clock tree while the UST-U-CTO structure is obtained by constructing multiple clock trees using different safety margins and selecting the clock tree structure that performs best in terms of TNS and WNS.

Based on the evaluations above, the UST-N structure demonstrates that the proposed methodology in Figure 2(b) outperforms the traditional clock tree construction methodology in Figure 2(a) in terms of average timing quality, capacitance and run-time.

VI. SUMMARY AND FUTURE WORKS

In this work, we proposed an OCV-Aware clock tree synthesis methodology that is capable of accounting the impact of OCV during the synthesis process. Moreover, the convergence of the proposed synthesis flow is completely automated unlike the state-of-the-art synthesis flow that often requires costly manual intervention in the form of ECOs to close timing. Compared to the state-of-the-art flow, the proposed flow demonstrates higher timing quality with shorter run-time and lower capacitance. In the future, we plan to extend the proposed flow to incorporate techniques of reconstructing the top-level tree topology during the synthesis process.

REFERENCES

[1] H. Bakoglu. Circuits, interconnects, and packaging for VLSI. Reading, MA:Addison-Wesley., 1990.

[2] T.-B. Chan, K. Han, A. B. Kahng, J.-G. Lee, and S. Nath. OCV-aware top-level clock tree optimization. In *Proc. Great Lakes Symposium on VLSI*, pages 33–38, 2014.

[3] Y. P. Chen and D. F. Wong. An algorithm for zero-skew clock tree routing with buffer insertion. EDTC'96, pages 230–237, 1996.

[4] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao. Bounded-skew clock and steiner routing. *ACM TODAES*, 3(3):341–388, July 1998.

[5] M. Edahiro. Minimum skew and minimum path length routing in VLSI layout design. In *NEC Research and Development*, pages 569–575, 1991.

[6] R. Ewetz. A clock tree optimization framework with predictable timing quality. DAC'17, pages 13–18, 2017.
[7] R. Ewetz and C. Koh. Scalable construction of clock trees with useful skew and high timing quality. *TCAD*, 2018.
[8] R. Ewetz and C.-K. Koh. Clock tree construction based on arrival time constraints. ISPD'17, pages 67–74, 2017.
[9] R. Ewetz, T. C. Yean, S. Janarthanan, and C.-K. Koh. Benchmark circuits for clock scheduling and synthesis. [Available Online] https://purr.purdue.edu/publications/1759, 2015.
[10] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, 1990.
[11] 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.
[12] W.-C. D. Lam and C.-K. Koh. Process variation robust clock tree routing. ASP-DAC '05, pages 606–611, 2005.
[13] D.-J. Lee and I. L. Markov. Multilevel tree fusion for robust clock networks. ICCAD'2011, pages 632–639, 2011.
[14] J. Lu and B. Taskin. Post-CTS clock skew scheduling with limited delay buffering. In *Proc. Int. Midwest Symposium on Circuits and Systems*, pages 224–227, 2009.
[15] S. Roy, P. M. Mattheakis, L. Masse-Navette, and D. Z. Pan. Clock tree resynthesis for multi-corner multi-mode timing closure. *TCAD*, pages 589–602, 2015.
[16] C. N. Sze. ISPD 2010 high performance clock network 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. *ACM TODAES*, 7(3):359–379, 2002.
[18] N. Uysal and R. Ewetz. OCV guided clock tree topology reconstruction. In *ASP-DAC'18*, pages 1–6, 2018.
[19] N. Uysal, W.-H. Liu, and R. Ewetz. Latency constraint guided buffer sizing and layer assignment for clock trees with useful skew. In *ASPDAC '19*, pages 761–766, 2019.