Toward Controllable Hierarchical Clock Tree Synthesis with Skew-Latency-Load Tree

Weiguo Li  
Minnan Normal University  
Beijing Institute of Open Source Chip

Zhipeng Huang  
Peng Cheng Laboratory

Bei Yu  
The Chinese University of Hong Kong

Wenxing Zhu  
Fuzhou University

Xingquan Li∗  
Minnan Normal University  
Peng Cheng Laboratory

ABSTRACT
Clock tree synthesis (CTS) constructs an efficient clock tree, meeting design constraints and minimizing resource usage. It serves as a bridge between placement and routing, facilitating concurrent optimization of multiple design objectives. To construct a clock tree with lower latency and load capacitance while maintaining a specified skew constraint, we introduce skew-latency-load tree (SLLT) which combines the merits of bound skew tree and Steiner shallow-light tree, along with an analysis and demonstration of the boundaries of these two tree types. We propose a method for constructing SLLT, which significantly reduces both the maximum latency and load capacitance compared to previous methods while ensuring skew control. Combining this routing topology generation method, we introduce a hierarchical CTS framework, and it is constructed by integrating partition schemes and buffering optimization techniques. We validate our solution at 28nm process technology, demonstrating superior performance compared to the solutions of OpenROAD and advanced commercial tool. Our approach outperforms in all metrics (max latency, skew, buffer number, clock capacitance), achieving a significant reduction in latency of 29.45% compared to OpenROAD and 6.75% compared to the commercial tool.

KEYWORDS
chip design automation, clock tree synthesis, skew-latency-load tree, buffer optimization

1 INTRODUCTION
Clock tree synthesis is a vital component of physical design as power consumption primarily stems from the interconnects and buffers involved in the construction process. The power consumption resulting from the CTS phase is commonly estimated to account for one-third of the total power consumption of an integrated circuit (IC) [17]. In certain designs, it can even constitute half of the total power consumption. Therefore, in the design of high-performance and low-power IC products, the requirements for the clock tree are becoming increasingly stringent. Due to the adverse effects of on-chip variation (OCV), conventional CTS method that focuses solely on skew optimization is inadequate for meeting the demands of advanced technology [10].

In the advanced process, wire delay has become increasingly important [21], especially for CTS. Meeting design requirements often entails special demands on interconnect wire, posing significant challenges to routability. In contrast, the proximity of the clock tree’s routing topology to the outcome of the routing stage improves its reliability and robustness. All these examples indicate that CTS imposes deeper requirements on interconnect structures beyond skew. To deal with such changes, besides addressing skew, it is crucial for CTS to also consider and analyze other interconnect characteristics present on the tree. Earlier works for CTS can be categorized into non-tree approaches and tree-based approaches [16]. Design methods like mesh topologies [7] and "Fishbone" topologies [8] mitigate clock skew and bolster the resilience of clock networks. Nevertheless, these methods exhibit limited structural variability, leading to reduced design flexibility. The symmetric topology of the H-tree allows for easy compliance with skew constraints, albeit with a notable increase in required design resources. The optimal generalized H-tree (GH-tree) [16] extends and optimizes the H-tree through the introduction of a branching factor. The deferred-merge embedding (DME) method performs bottom-up calculations of merging regions that satisfy the requirements, continuously complementing the upstream topological structure, and finally, performs top-down positional embedding. It can achieve various variants, including zero skew tree (ZST) [11], bounded skew tree (BST) [14], and useful skew tree (UST) [20]. All of the aforementioned skew-tree methods are capable of achieving clock skew control. Unfortunately, these traditional methods incur significant costs due to increased path length and total wirelength.

In addition, the rectilinear Steiner minimum tree (RSMT) method, exemplified by FLUTE [13], and the rectilinear Steiner SLT method, such as Steiner shallow-light tree (SALT) [15], aim to minimize the total wirelength and path length, respectively. A smaller total wirelength can directly help CTS reduce power consumption, while a shorter path length can reduce latency from the clock source to the sink and create greater freedom for timing optimization. Regrettably, these methods have been overlooked by CTS because they are unable to meet the most important skew objectives.

It is noteworthy that the Steiner-tree method can easily replace the skew-tree method when the skew constraint is relaxed. This scenario also exhibits the highest resemblance between the CTS and routing stages on the bottom nets of clock tree. However, this phenomenon precisely indicates that there is a missing bridge between the existing skew-tree method and the Steiner-tree method. With the help of this bridge, CTS can explore optimal solutions that were previously difficult to find by utilizing the structure of the Steiner-tree method. By achieving short path lengths and total wirelengths under acceptable skew conditions, CTS can achieve

∗Corresponding author: Xingquan Li (fxksz@gmail.com)

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 the author(s) 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.

DCC ‘24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-8601-1/24/06...
https://doi.org/10.1145/3649329.3658243
improved global performance. In this paper, we investigate the relationship between conventional skew-tree approaches and Steiner-tree approaches, and propose the skew-latency-load tree (SLLT) concept. Then we present a construction method of SLLT, named Concurrent BST and SALT (CBS), which can control skew and effectively reduce latency and load capacitance. We propose a hierarchical design framework for CTS, which iteratively performs partitioning of instances, routing topology generation, and buffer optimization at each level. The specific contributions can be summarized as follows:

- We define skewness to evaluate the difference of a Steiner tree from the root to all leaves. With skewness, shallowness, and lightness, SLLT enables comprehensive analysis of Steiner trees. We also provide a theoretical proof of the binary impossibility properties of shallowness and skewness.
- Our SLLT constructing method integrates the benefits of SALT [12] and BST-DME [14], effectively reducing both maximum latency and load capacitance compared to the skew-tree method.
- With SLT and CBS, We propose a hierarchical design framework for CTS. Honoring this framework, we can apply heuristic methods for optimizing solutions and accommodate diverse design requirements using various routing topology generation approaches.
- Our approach surpassed both OpenROAD and a commercial tool in all performance indicators, reducing latency by 29.45% and clock capacitance by 20.58% compared to OpenROAD, and achieving a 6.75% reduction in latency and a 16.36% reduction in clock capacitance compared to the commercial tool.
- By embracing the concept of SLT, we can achieve a higher-level description of the clock tree distribution and conduct a more in-depth analysis of its quality. Significantly, SLLT can cater to all network structures and introduce a novel interconnect quality evaluation system, offering effective and comprehensive assessments for various backend design processes.

2 SKEW-LATENCY-LOAD TREE CONSTRUCTION

In this paper, the primary design objective is to synchronize the clock signal by controlling skew. Given a clock tree represented by \( T \), we define the delay from clock source to sink sink \( s_j \in S \) as \( delay(s_j) \), where \( S \) is the set of sinks, then \( skew = \max_{s_j \in S} \{ delay(s_j) \} - \min_{s_j \in S} \{ delay(s_j) \} \). Moreover, CTS aims to design a chip with prioritize high performance and low power consumption. Therefore, effective control of maximum latency and load capacitance is also imperative. The maximum latency of a clock tree can be denoted by \( latency_{max} = \max_{s_j \in S} \{ delay(s_j) \} \). Let \( WL(T) \) denote the total wirelength of clock tree \( T \), and the load capacitance can be simplified calculated: \( load = \sum_{s_i \in S} Cap_{pin}(s_i) + c \cdot WL(T) \), where \( c \) represents the capacitance per unit length of the wire, and \( Cap_{pin}(s_i) \) denotes the capacitance on the pin of \( s_i \). To gain an intuitive understanding on skew, latency and load, we try to make a map between skew, latency and load with the physical length of clock tree. Firstly, since the path length from the source to sinks significantly influences delay calculation, the maximum latency is positively correlated with the longest path of the clock tree. Secondly, skew can be represented as the maximum deviation in path lengths of clock tree. Thirdly, the load capacitance is directly affected by the total wirelength of the clock tree. These relationships are given as below:

\[
\text{skew} = \max_{s_j \in S} \{ PL(s_j) \} - \min_{s_j \in S} \{ PL(s_j) \},
\]

\[
\text{latency}_{max} = \max_{s_j \in S} \{ PL(s_j) \},
\]

\[
\text{load} = WL(T),
\]

where \( PL(s_j) \) denotes the path length from the source to \( s_j \) of \( T \).

2.1 Skew-Latency-Load Tree (SLLT)

Currently, there are some works dedicated to optimizing the aforementioned objectives. In Ref. [12], the authors proposed a shallow-light tree (SLT) construction algorithm to simultaneously approximates: 1) shortest distances from a root to the other vertices, and 2) the minimum tree weight. In SLT, \( \alpha \) represents shallowness with \( \alpha = \max_{s_j \in S} \{ PL(s_j) \} \), where \( MD(s_j) \) denotes the Manhattan distance from the source. \( \beta \) represents lightness with \( \beta = \frac{WL(T)}{WL(MST(G))} \), where \( MST(G) \) represents the minimum spanning tree of \( G \). Considering the layout characteristics of physical design, we generally assume the clock tree \( T \) to be a rectilinear Steiner tree in this paper. As in [12], \( \beta \) is also approximated as \( \beta \approx \frac{WL(T)}{WL(FLUITE)} \) in this paper, where \( WL(T_{FLUITE}) \) is the wirelength of Steiner tree generated by FLUTE [13]. According to these measurements, shallowness \( \alpha \) reflects the latency of the clock tree (referring to the delay from the source to the pin). Likewise, lightness \( \beta \) reflects the load capacitance of the clock tree. In an \( (\alpha, \beta) \)-SLT, where \( \alpha \geq \beta \geq 1 \) and \( \beta \geq 1 \). The SLT has the property of concurrently optimizing latency and load capacitance, and \( \alpha \) and \( \beta \) in an \( (\alpha, \beta) \)-SLT reflect the value of maximum latency and load capacitance of this tree.

However, the primary objective of traditional CTS algorithms is to effectively control skew. This paper aims to enhance the capabilities of SLT to enable it to be able to effectively control skew. Similar to shallowness and lightness, we define a metric of skew:

**Definition 2.1 (skewness).** According to Equation (1), we define the skewness of a Steiner tree: \( y = \frac{\max_{s_j \in S} \{ PL(s_j) \}}{PL} \).

In the above definition, \( PL \) represent the average path length. Similar to shallowness and lightness, it can be easily seen that \( y \geq 1 \), and if \( y = 1 \), i.e., \( max_{s_j \in S} \{ PL(s_j) \} = PL = \min_{s_j \in S} \{ PL(s_j) \} \), then we obtain a zero skew tree on the wirelength delay model. In general, the convergence objective of a modern CTS method or tool is to construct a clock tree with bounded skew or useful skew. Both of bounded skew and useful skew can be limited by an upper bound \( \gamma \) with \( \gamma \geq y \). For a rectilinear Steiner tree, to evaluate its capacity of optimizing skew, maximum latency and load, we propose a Skew-Latency-Load tree (SLLT) as follows:

**Definition 2.2 ((\( \alpha, \beta, \gamma \))-SLLT).** A rectilinear Steiner tree with shallowness \( \alpha \leq \gamma \), lightness \( \beta \leq \gamma \), and skewness \( \gamma \leq \gamma \) is denoted as \( (\alpha, \beta, \gamma) \)-SLLT. 

To compare the variations of shallowness, lightness and skewness among various SLLTs, we implemented several existing algorithms. As given in Fig. 1, the SLLTs generated by some fundamental conventional CTS algorithms (H-tree, GH-tree, ZST-DME, BST-DME) are shown from Fig. 1(b) to Fig. 1(e), Fig. 1(f) and Fig. 1(g) are built by FLUTE and rectilinear-SALT (R-SALT) [12] corresponding to the RSMT and rectilinear Steiner SLT algorithms, respectively. In addition, the shallowness \( \alpha \), lightness \( \beta \) and skewness \( \gamma \) of these
algorithms are calculated and listed in Table 1. Column “Skew Control” lists the controllability of all algorithms, and column “Mean” is the average value of the three metrics.

Table 1: Different routing topologies on net. α, β, and γ represent shallowness, lightness, and skewness respectively.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Max PL</th>
<th>Min PL</th>
<th>Total WL</th>
<th>Mean PL</th>
<th>α</th>
<th>β</th>
<th>γ</th>
<th>Mean α</th>
<th>Mean β</th>
<th>Mean γ</th>
<th>Skew Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>H-tree</td>
<td>10</td>
<td>9</td>
<td>55.5</td>
<td>9.75</td>
<td>2.00</td>
<td>1.32</td>
<td>1.03</td>
<td>1.45</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GH-tree</td>
<td>10</td>
<td>7</td>
<td>47.5</td>
<td>8.50</td>
<td>1.60</td>
<td>1.13</td>
<td>1.18</td>
<td>1.30</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZST</td>
<td>10.5</td>
<td>10.5</td>
<td>55.5</td>
<td>10.50</td>
<td>2.33</td>
<td>1.32</td>
<td>1.00</td>
<td>1.55</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BST</td>
<td>10</td>
<td>8</td>
<td>50</td>
<td>9.25</td>
<td>2.25</td>
<td>1.19</td>
<td>1.08</td>
<td>1.51</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLUTE</td>
<td>9</td>
<td>5</td>
<td>42</td>
<td>7.44</td>
<td>1.40</td>
<td>1.00</td>
<td>1.21</td>
<td>1.20</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R-SALT</td>
<td>9</td>
<td>5</td>
<td>43</td>
<td>7.06</td>
<td>1.00</td>
<td>1.02</td>
<td>1.27</td>
<td>1.10</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBS</td>
<td>9</td>
<td>7</td>
<td>45</td>
<td>8.13</td>
<td>1.40</td>
<td>1.07</td>
<td>1.11</td>
<td>1.19</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As we all know, ZST-DME can achieve zero skew. From Table 1, we can see that H-tree and ZST-DME achieve smaller skewness, but have a larger shallowness and lightness. GH-tree is a variant of H-tree by expanding the number of branches. And the BST-DME is a variant of ZST-DME by relaxing skew bound from zero to a small positive number. Compared with H-tree and ZST-DME, GH-tree and BST-DME achieve better trade-off among shallowness, lightness and skewness. Since R-SALT can obtain the minimum path length of the source to each pin, R-SALT achieves an SLLT with shallowness α = 1. And, FLUTE aims to implement a best RSMT with minimum wirelength, it achieves an SLLT with lightness β = 1. Regrettably, R-SALT and FLUTE cannot effectively control skewness. In conclusion, traditional CTS algorithms effectively handle skewness, while R-SALT and FLUTE achieve superior shallowness and lightness. However, the primary objective of CTS is to achieve smaller latency and load capacitance within an acceptable skew range. In other words, it aims to minimize shallowness and lightness while ensuring controllable skewness. In this paper, we try to construct a class of SLLTs that meet this requirement.

2.2 Bound Analysis

In Ref. [12], the SALT algorithm can achieve the Steiner (1 + ϵ, 2 + \log \frac{2}{\epsilon})-SLLT. However, SALT cannot effectively control skewness. To verify the claim more theoretically, we try to give a proof by contradiction about that skewness and shallowness cannot be simultaneously less than a small positive number. Before that, we assume that skewness and shallowness have the same dimension, which is defined by the condition γ ≤ 1 + ϵ, where ϵ is a small positive number. We propose a theorem that establishes the boundary where skewness and shallowness are mutually exclusive.

Theorem 2.3. Given a set of pins and a small positive number ϵ, when the pin distribution has

\[
\max_{s \in S} \frac{MD(s_i)}{MD} > (1 + \epsilon)^2,
\]

for an SLLT, it is not possible to simultaneously satisfy α ≤ 1 + ϵ and γ ≤ 1 + ϵ.

Proof: Assume that there exists an SLLT with α ≤ 1 + ϵ and γ ≤ 1 + ϵ. Additionally, if shallowness is satisfied, then \[\max_{s \in S} \frac{PL(s_i)}{MD(s_i)} \leq 1 + \epsilon.\] Therefore, for all pins \( s_i \in S \), \( PL(s_i) \leq (1 + \epsilon) \cdot MD(s_i) \). Further, we can deduce:

\[
\sum_{s \in S} \frac{PL(s_i)}{|S|} \leq (1 + \epsilon) \cdot \frac{\max_{s \in S} MD(s)}{|S|},
\]

\[
\frac{PL}{(1 + \epsilon) \cdot MD} \leq 1 + \epsilon.
\]

Obviously \( \max_{s \in S} \frac{PL(s_i)}{MD(s_i)} \geq \frac{\max_{s \in S} MD(s_i)}{MD} \), then based on Equation (4) and Equation (5), we can deduce:

\[
\gamma = \frac{\max_{s \in S} \frac{PL(s_i)}{MD(s_i)}}{\frac{PL}{MD}} \geq \frac{\max_{s \in S} MD(s_i)}{(1 + \epsilon) \cdot MD} > 1 + \epsilon.
\]

We conclude that γ > 1 + ϵ, this result contradicts γ ≤ 1 + ϵ.

2.3 Constructing SLLT by Concurrent BST and SALT (CBS)

To overcome the limitation of SALT and other SLLT construct algorithms, we present a concurrent BST and SALT construction method to generate SLLT, named CBS algorithm. The process of this method is shown in Fig. 2:

- **Step 1:** The BST is utilized to construct an initial SLLT (SLLT). In BST, the optimal candidate merging topology includes Greedy-Dist, Greedy-Merge, Bi-Partition, and Bi-Cluster.

[1] For the Greedy-Dist method, the two closest subtrees are merged greedily at each step. The Greedy-Merge method selects and merges the two subtrees with the minimum merging cost at each step. The Bi-Partition method performs binary partitioning.
• **Step 2**: The tree topology of the BST is extracted, and in which the redundant Steiner nodes will be eliminated, and the topology result is set as the input of SALT.

• **Step 3**: The SALT algorithm is used to relax and optimize above topology, then the path that is longer from the root path will be adjusted in this phase, but it breaks the skew legitimacy.

• **Step 4**: Then, we extract the tree topology of BST-SALT, and traverse all nodes to ensure the following rules should be satisfied: 1) the tree should be a binary tree, and 2) the load pin nodes must be leaf nodes.

• **Step 5**: After that, the BST is conducted on the tree topology of Step 4, and redundant nodes are further eliminated to ensure that skewness is satisfied and the obtained result closely approximate the result by SALT.

### Table 2: Wirelength (um) comparison between R-SALT and CBS.

<table>
<thead>
<tr>
<th>Skew (ps)</th>
<th>GreedyDist</th>
<th>GreedyMerge</th>
<th>BiPartition</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-SALT</td>
<td>314.4</td>
<td>314.3</td>
<td>315.1</td>
</tr>
<tr>
<td></td>
<td>312.6</td>
<td>313.0</td>
<td>315.6</td>
</tr>
<tr>
<td></td>
<td>312.2</td>
<td>312.4</td>
<td>312.7</td>
</tr>
<tr>
<td>CBS</td>
<td>306.0</td>
<td>307.1</td>
<td>316.1</td>
</tr>
<tr>
<td></td>
<td>305.2</td>
<td>306.3</td>
<td>314.3</td>
</tr>
<tr>
<td></td>
<td>305.3</td>
<td>305.6</td>
<td>312.7</td>
</tr>
<tr>
<td>Reduce</td>
<td>2.69%</td>
<td>2.29%</td>
<td>-0.32%</td>
</tr>
</tbody>
</table>

As shown in Table 1, our results achieve better shallowness and lightness when skew is controlled. Compared with traditionalCTS algorithm, our CBS achieves the best shallowness and lightness, and compared with FLUTE and R-SALT, we effectively control skewness. To further evaluate the effectiveness of this control method, we compare our CBS procedure with R-SALT and BST-DME by randomly generate a number of clock nets as shown in Table 2 and Table 3. All nets are generated within a box with boundary of 75um in both the x and y coordinates. And the numbers of load pins of all nets vary from 10 to 40. We set three different skew constraints: 1) (relaxed) skew bound is 80ps; 2) (moderate) skew bound is 10ps; 3) (stringent) skew bound is 5ps. For each skew level, we generate 10,000 nets to ensure the sufficiency of the analysis. Compared to R-SALT, as shown in Table 2, our CBS procedure achieves a smaller total wirelength under the skew bound of 80ps and 10ps. Compared to BST-DME, as shown in Table 3, our CBS procedure obtains significant reduction on total wirelength, load capacitance and wire delay under the same skew bound.

### Table 3: Comparison on wirelength, cap and delay between BST-DME and CBS.

<table>
<thead>
<tr>
<th>Skew (ps)</th>
<th>Wirelength (um)</th>
<th>Cap (ff)</th>
<th>Wire Delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>BST-DME</td>
<td>363.3</td>
<td>367.6</td>
<td>373.2</td>
</tr>
<tr>
<td></td>
<td>372.6</td>
<td>77.4</td>
<td>78.1</td>
</tr>
<tr>
<td></td>
<td>79.1</td>
<td>15.3</td>
<td>11.5</td>
</tr>
<tr>
<td></td>
<td>10.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBS</td>
<td>306.5</td>
<td>306.1</td>
<td>314.0</td>
</tr>
<tr>
<td></td>
<td>67.5</td>
<td>67.6</td>
<td>68.9</td>
</tr>
<tr>
<td></td>
<td>11.2</td>
<td>9.2</td>
<td>7.4</td>
</tr>
<tr>
<td>Reduce</td>
<td>15.9%</td>
<td>16.7%</td>
<td>15.9%</td>
</tr>
<tr>
<td></td>
<td>12.8%</td>
<td>13.5%</td>
<td>12.8%</td>
</tr>
<tr>
<td></td>
<td>26.6%</td>
<td>20.5%</td>
<td>26.8%</td>
</tr>
</tbody>
</table>

3 **HIERARCHICAL CLOCK TREE SYNTHESIS**

In this section, we introduce our hierarchical clock tree synthesis framework: partition, routing topology generation and buffer optimization.

### 3.1 Formulation and Framework

In our framework, we design clock tree by a hierarchical process with different sub-process at each level. For the sub-process at level $k$, we let $S^k_i \subseteq S^k$ be the clock nodes at level $k$, and divide these nodes as several clusters with $u^k_j \in L^k$. Let $n^k_j \in N^k$ be the net in each round based on the diameter cost of the partitioned subsets. Our proposed Bi-Cluster method achieves quick partitioning by recursively performing binary partitions in a clustering manner.

### 3.2 Partition Scheme

Latency and Capacitance Adaptive Clustering. At clustering stage, by combining K-means clustering with the min-cost flow (MCF), Ref. [16] controls the maximum number of nodes in cluster. By evaluating the clustering quality, we can calculate the silhouette score of K-means. To balance latency and capacitance, we define a new evaluation metric by integrating the variances of capacitance and delay as:

$$
\text{Cost}^k = p \cdot \sigma(C) + q \cdot \sigma(T),
$$

where $C^k = (C^k_1, C^k_2, \ldots, C^k_{|N^k|})$, and $C^k_1$ is the total capacitance of clock net $n^k_1 \in N^k$. Similarly, $T^k = (T^k_1, T^k_2, \ldots, T^k_{|N^k|})$, where $T^k_j = \max_{s_i \in L^k(s^k_j)} \{\text{delay}(s^k_{j+1}, s_i)\}$. And $\sigma(\cdot)$ denotes the variance function. In CTS, the delay will increase with the increase of levels, and the load capacitance is mostly accumulated in the bottom level. Hence, this cost scheme adapts the level characteristic of clock net.

Optimizing Partition by Simulated Annealing. On the basis of the above adaptive clustering result, we further optimize the capacitance and wirelength violation to obtain a better partition solution by introducing simulated annealing. To improve the search efficiency of SA, we use capacitance as the unified metric.

In such computations, all constraint costs have equivalent numerical ranges. After formulating evaluation metric, a more crucial thing is deciding the search neighborhood radius. During our experimental process, we have the following observations:

- To prevent the crossover of net interconnections, we should prioritize selecting instances along the boundary of the net.
- Since the net costs are independent, greedy strategy can effectively reduce the global cost by net cost in descending order.

According to the above observations, we propose a local search strategy for SA, as shown in Fig. 4.
The routing topology helps us determine the location of the driver buffer. In a global clock tree, we can find the common ancestor node of bottom clock nodes as the reference location of the driver buffer.

It is crucial to develop an accurate routing solution that updates the timing information of driver instance, facilitating the progress to the next level of the process.

Design requirements dictate the choice of generation methods for traditional CTS construction. Algorithms with skew-control are preferred, while routability concerns necessitate lighter SLLT, favoring FLUTE-like tree structures. For larger designs, minimizing latency and the clock source-to-sink distance is key, requiring less shallow SLLT for shorter paths.

With CBS, we can implement trade-offs for these scenarios. We can fulfill the traditional CTS requirements for bounded skew, while keeping the topology and Routing stage as close as possible, and shortening the path length to reduce the maximum latency to meet the challenges of large-scale clock tree design.

### 3.4 Buffering Optimization

**Buffer Driver Capability Estimation.**  Slew and delay are calculated as in [19], with [18] showing a first-order model’s accuracy through a linear buffer delay \( D_{\text{buf}}(t) \) equation dependent on input slew \( \text{Slew}_\text{in}(t) \) and load capacitance \( \text{Cap}_{\text{load}}(t) \):

\[
D_{\text{buf}}(t) = \omega_s \cdot \text{Slew}_\text{in}(t) + \omega_c \cdot \text{Cap}_{\text{load}}(t) + \omega_i,
\]

where \( \omega_s, \omega_c, \) and \( \omega_i \) represent slew, capacitance coefficients, and inherent delay, respectively. We consider the following scenario: buffer \( b_{f1} \) and buffer \( b_{f2} \) linked by a wire with \( b_{u1}, b_{u2} \) inserted between them.

Based on Equation (6), the following result is clear:

\[
T(i,j) - T'(i,j) = \frac{r \cdot c \cdot (\ln \circ\omega_c + 1) \cdot L(i,j)^2}{4} - \omega_c \cdot \text{Cap}_{\text{pin}} - \omega_i,
\]

where \( r \) represents the resistance per unit length of the wire. By solving \( T(i,j) = T'(i,j) \), we can obtain the **critical wirelength**:

\[
L(i,j) = 2 \sqrt{\frac{\omega_c \cdot \text{Cap}_{\text{pin}} \cdot \tau_{\text{delay}}}{r \cdot c \cdot (\ln \circ\omega_c + 1)}}.
\]

By substituting \( \text{Cap}_{\text{pin}} \) with \( \text{Cap}_{\text{load}} \), then \( \hat{L}(i,j) = 2 \sqrt{\frac{\omega_c \cdot \text{Cap}_{\text{load}} \cdot \tau_{\text{delay}}}{r \cdot c \cdot (\ln \circ\omega_c + 1)}} \), we obtain a refined estimation as \( \text{Cap}_{\text{load}} \) accounts for pin capacitance. Despite ignoring complex net slew effects, this approximation, favoring capacitance, is deemed acceptable for our analysis.

**Insertion Delay Lower Bound Estimation.**  To correct skew, previous methods necessitated adjustments in downstream buffer sizes or iterative net construction. We discovered that estimating a provisional delay for each node during timing analysis minimizes downstream alterations upon upstream node merging. We use the most conservative lower bound estimate:

\[
\hat{D}_{\text{buf}}(t) = \min_{\text{lib} \in \text{Lib}} \{\omega_i(\text{lib}) \cdot \text{Cap}_{\text{load}}(t) + \min_{\text{lib} \in \text{Lib}} \{\omega_s(\text{lib})\}\}. \tag{7}
\]

In Fig. 5, we estimate delay \( t_{\text{buf}} \) prior to buffer insertion, initially increasing delay. Post-buffer insertion, accounting for downstream \( \text{Cap}_{\text{load}} \) and \( \omega_i \), downstream changes diminish. This method lowers skew repair costs and latency by reducing downstream node disparities.

In Equation (6), load capacitance impacts delay more than input slew, yielding a smaller delay gap for an effective lower bound. However, overly aggressive estimates may not always narrow the delay gap, risking increased variations in downstream nodes.

### 4 EXPERIMENTAL RESULTS

**Table 4: Design statistics.**

<table>
<thead>
<tr>
<th>Case</th>
<th>s38584</th>
<th>s38417</th>
<th>s35932</th>
<th>salsa20</th>
<th>ethernet</th>
<th>iyyx_1</th>
<th>iyyx_2</th>
<th>iyyx_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insts.</td>
<td>7510</td>
<td>6428</td>
<td>6113</td>
<td>39945</td>
<td>60541</td>
<td>86933</td>
<td>93907</td>
<td>139956</td>
</tr>
<tr>
<td>FFs</td>
<td>1248</td>
<td>1564</td>
<td>1728</td>
<td>2375</td>
<td>10015</td>
<td>16902</td>
<td>18487</td>
<td>27078</td>
</tr>
<tr>
<td>Util</td>
<td>0.60</td>
<td>0.61</td>
<td>0.58</td>
<td>0.68</td>
<td>0.61</td>
<td>0.55</td>
<td>0.93</td>
<td>0.868</td>
</tr>
</tbody>
</table>

We performed experiments utilizing the 28\(nm\) manufacturing process and a standard cell library. The input placement schemes were executed utilizing the Cadence Innovus Implementation System v19.1 [3]. Our solution was developed employing C++ and Tcl scripts, with all experiments conducted on a 2.40GHz Intel Xeon Platinum-8260 server. To construct the reference clock tree solution, we leveraged prominent commercial P&R tool alongside the widely-used open-source tool OpenROAD [2].

We evaluate six open source designs and four internal testing designs to assess the effectiveness of all solutions. These designs include s38584, s38417, and s35932 from the ISCAS’89 [1], salsa20

\[\text{Cap}_{\text{load}}(t) + \min_{\text{lib} \in \text{Lib}} \{\omega_s(\text{lib})\}. \tag{7}\]
from OpenLane CI Designs [5], ethernet and vga_lcd from OpenCores [4], $ysyx_0$ to $ysyx_3$ from our internal testing. And we use Synopsys Design Compiler R-2020.09-SP3a [6] to synthesize these designs. The statistics of these designs are shown in Table 4.

The tools were configured with design constraints outlined in Table 5, based on engineers’ practical experiences in chip fabrication. Table 6 presents the experimental results obtained in this study. The columns “Ours”, “Com.”, “OR.” represent our solution and the commercial tool, OpenROAD’s solution, respectively. In terms of maximum latency, our solution achieves a reduction of 6.75% compared to commercial tool and a reduction of 29.45% compared to OpenROAD. With regards to clock skew, we achieve a reduction of 5.80% compared to commercial tool and a reduction of 41.44% compared to OpenROAD, and in the case of vga_lcd, the OpenROAD solution exceeds the constraint skew. In terms of buffer count, buffer area, and clock capacitance, we outperform both the commercial tool and OpenROAD. Additionally, in terms of total wirelength, the discrepancy between our solution and commercial tool is a mere 0.59%, while compared to OpenROAD there is a reduction of 2.73%. In conclusion, our solution and commercial tool is a mere 0.59%, while compared to OpenROAD there is a reduction of 2.73%. In conclusion, our solution

### Table 6: Comparison between clock tree solutions from ours, commercial tool, and OpenROAD.

<table>
<thead>
<tr>
<th>Case</th>
<th>Latency (ps)</th>
<th>Skew (ps)</th>
<th>#Buffers</th>
<th>Buf Area (μm²)</th>
<th>Clk Cap (FF)</th>
<th>Clk WL (μm)</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>0.992</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Com.</td>
<td>0.986</td>
<td>0.988</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>OR.</td>
<td>0.992</td>
<td>0.996</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

### Table 7: Comparison of $ysyx$ designs among clock tree solutions from ours, commercial tool, and OpenROAD.

<table>
<thead>
<tr>
<th>Case</th>
<th>Latency (ps)</th>
<th>Skew (ps)</th>
<th>#Buffers</th>
<th>Buf Area (μm²)</th>
<th>Clk Cap (FF)</th>
<th>Clk WL (μm)</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>0.992</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Com.</td>
<td>0.986</td>
<td>0.988</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>OR.</td>
<td>0.992</td>
<td>0.996</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

5 CONCLUSION

Under the concept of SLIT, we have bridged the CTS and routing phases, as well as several intermediate algorithms. From an algorithmic perspective, the method we proposed effectively combines traditional shallow-light tree and conventional clock tree methods. This approach optimizes resource utilization while maintaining skew control, thereby demonstrating superior low-power characteristics of our design methodology and architecture in experimental results. In future research, we plan to develop a comprehensive SLIT model, explore feasible metric transformations, and refine the architecture further.

ACKNOWLEDGMENT

This work is supported in part by the Major Key Project of PCL (No. PCL.2023A03), the National Natural Science Foundation of China (No. 62174033, No. 61907024).

REFERENCES

[14] 2010. SLLT model, explore feasible metric transformations, and refine the architecture further.