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 (fzulxq@gmail.com)

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
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 leafs. 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 SLLT 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 SLLT, 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 the CTS phase, 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 $s_i \in \mathcal{S}$ as $delay(s_i)$, where $\mathcal{S}$ is the set of sinks, then $skew = \max_{s_i \in \mathcal{S}}\{delay(s_i)\} - \min_{s_i \in \mathcal{S}}\{delay(s_i)\}$. Moreover, CTS aims to design 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: $latency_{\max} = \max_{s_i \in \mathcal{S}}\{delay(s_i)\}$. Let $WL(T)$ denote the total wirelength of clock tree T, and the load capacitance can be simplify calculated: $load = \sum_{s_i \in \mathcal{S}} Cap_{pin}(s_i) + c \cdot WL(T)$, 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 T. Secondly, skew can be represented as the maximum deviation in path lengths of clock tree T. Thirdly, the load capacitance is directly affected by the total wirelength of the clock tree T. These relationships are given as below:

$$skew \propto \max_{s_i \in \mathcal{S}}\{PL(s_i)\} - \min_{s_i \in \mathcal{S}}\{PL(s_i)\}, \tag{1}$$

$$latency_{\max} \propto \max_{s_i \in \mathcal{S}}\{PL(s_i)\}, \tag{2}$$

$$load \propto WL(T), \tag{3}$$

where $PL(s_i)$ denotes the path length from the source to s_i 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, α represents shallowness with $\alpha = \max_{s_i \in \mathcal{S}}\{\frac{PL(s_i)}{MD(s_i)}\}$, where $MD(s_i)$ denotes the Manhattan distance from the source. β 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], β is also approximated as $\beta \approx \frac{WL(T)}{WL(T_{FLUTE})}$ in this paper, where $WL(T_{FLUTE})$ is the wirelength of Steiner tree generated by FLUTE [13]. According to these measurements, **shallowness** reflects the **latency** of the clock tree (referring to the delay from the source to the pin). Likewise, **lightness** reflects the **load capacitance** of the clock tree. In an $(\overline{\alpha}, \overline{\beta})$-SLT, where $\overline{\alpha} \geq \alpha \geq 1$ and $\overline{\beta} \geq \beta \geq 1$. The SLT has the property of concurrently optimizing latency and load capacitance, α and β in an $(\overline{\alpha}, \overline{\beta})$-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: $\gamma = \frac{\max_{s_i \in \mathcal{S}}\{PL(s_i)\}}{\overline{PL}}$.

In the above definition, $\overline{PL}$ represent the average path length. Similar to shallowness and lightness, it can be easily seen that $\gamma \geq 1$, and if $\gamma = 1$, i.e., $\max_{s_i \in \mathcal{S}}\{PL(s_i)\} \equiv \overline{PL} \equiv \min_{s_i \in \mathcal{S}}\{PL(s_i)\}$, 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 $\overline{\gamma}$ with $\overline{\gamma} \geq \gamma$. For a rectilinear Steiner tree, to evaluate its capacity of optimizing skew, maximum latency and load, we propose a Skew-Lantency-Load tree (SLLT) as follows:

*Definition 2.2 (**$(\overline{\alpha}, \overline{\beta}, \overline{\gamma})$-SLLT**).* A rectilinear Steiner tree with shallowness $\alpha \leq \overline{\alpha}$, lightness $\beta \leq \overline{\beta}$, and skewness $\gamma \leq \overline{\gamma}$ is denoted as **$(\overline{\alpha}, \overline{\beta}, \overline{\gamma})$-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 famous traditional 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 α, lightness β and skewness γ of these

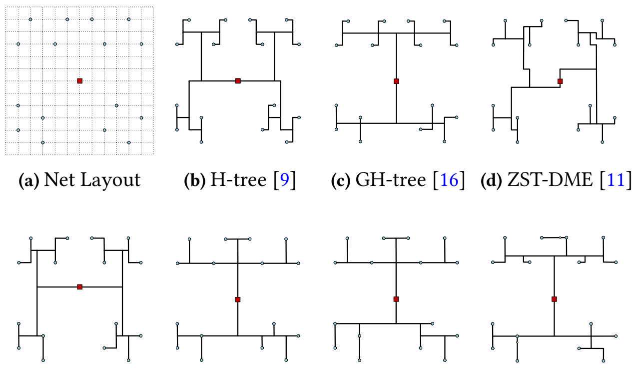
(a) Net Layout (b) H-tree [9] (c) GH-tree [16] (d) ZST-DME [11] (e) BST-DME [14] (f) FLUTE [13] (g) R-SALT [12] (h) Our CBS

Figure 1: Different SLLTs on net.

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.

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

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 $\alpha = 1$. And, FLUTE aims to implement a best RSMT with minimum wirelength, it achieves an SLLT with lightness $\beta = 1$. 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 $(1 + \epsilon, 2 + \lceil \log \frac{2}{\epsilon} \rceil)$-SLT. 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 $\gamma \leq 1+\epsilon$, where (ϵ is a small positive number). We propose a theorem that establishes the boundary where skewness and shallowness are mutually exclusive.

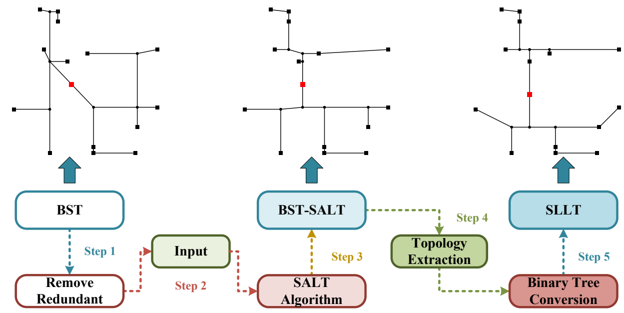

Figure 2: SLLT Construction Flow.

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

$$\frac{\max_{s_i \in \mathcal{S}}\{MD(s_i)\}}{\overline{MD}} > (1+\epsilon)^2, \tag{4}$$

for an SLLT, it is not possible to simultaneously satisfy $\alpha \leq 1+\epsilon$ and $\gamma \leq 1+\epsilon$.

Proof: Assume that there exists an SLLT with $\alpha \leq 1+\epsilon$ and $\gamma \leq 1+\epsilon$. Additionally, if shallowness is satisfied, then $\max_{s_i \in \mathcal{S}}\{\frac{PL(s_i)}{MD(s_i)}\} \leq 1+\epsilon$. Therefore, $\forall s_i \in \mathcal{S}, PL(s_i) \leq (1+\epsilon) \cdot MD(s_i)$. Further, we can deduce:

$$\begin{aligned}
\sum_{s_i \in \mathcal{S}} PL(s_i) &\leq (1+\epsilon) \cdot \sum_{s_i \in \mathcal{S}} MD(s_i), \\
\frac{\sum_{s_i \in \mathcal{S}} PL(s_i)}{|\mathcal{S}|} &\leq (1+\epsilon) \cdot \frac{\sum_{s_i \in \mathcal{S}} MD(s_i)}{|\mathcal{S}|}, \\
\overline{PL} &\leq (1+\epsilon) \cdot \overline{MD}.
\end{aligned} \tag{5}$$

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

$$\gamma = \frac{\max_{s_i \in \mathcal{S}}\{PL(s_i)\}}{\overline{PL}} \geq \frac{\max_{s_i \in \mathcal{S}}\{MD(s_i)\}}{\overline{PL}} \geq \frac{\max_{s_i \in \mathcal{S}}\{MD(s_i)\}}{(1+\epsilon) \cdot \overline{MD}} > 1+\epsilon.$$

We conclude that $\gamma > 1+\epsilon$, this result contradicts $\gamma \leq 1+\epsilon$. ■

We consider the case that Equation (4) does not hold, which can be derived as $\max_{s_i \in \mathcal{S}}\{MD(s_i)\} \approx \overline{MD}$, then $\max_{s_i \in \mathcal{S}}\{MD(s_i)\} \approx \min_{s_i \in \mathcal{S}}\{MD(s_i)\}$. This indicates that all pins have similar Manhattan distances to the root, meaning that all pins are distributed around a Manhattan circle near the root. In reality, pin distributions of this nature are uncommon. Condition Equation (4) describes the influence of the dispersion of the clock distribution on the simultaneous convergence of shallowness and skewness.

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 (iSLLT). In BST, the optional candidate merging topology includes *Greedy-Dist*, *Greedy-Merge*, *Bi-Partition*, and *Bi-Cluster*. [1]

[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.

	GreedyDist			GreedyMerge			BiPartition		
Skew (*ps*)	80	10	5	80	10	5	80	10	5
R-SALT	314.4	314.3	315.1	312.6	313.0	315.6	312.2	312.4	312.7
CBS	306.0	307.1	316.1	305.2	306.3	314.3	305.3	305.6	312.7
Reduce	**2.69%**	**2.29%**	-0.32%	**2.39%**	**2.14%**	**0.40%**	**2.20%**	**2.16%**	**0.00%**

As shown in Table 1, our results achieve better shallowness and lightness when skew is controlled. Compared with traditional CTS 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 construction 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 75*um* 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 80*ps*; 2) **(moderate)** skew bound is 10*ps*; 3) **(stringent)** skew bound is 5*ps*. 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 80*ps* and 10*ps*. 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.

	Wirelength (*um*)			Cap (*fF*)			Wire Delay (*ps*)		
Skew (*ps*)	80	10	5	80	10	5	80	10	5
BST-DME	363.3	367.6	373.2	77.4	78.1	79.1	15.3	11.5	10.2
CBS	305.6	306.1	314.0	67.5	67.6	68.9	11.2	9.2	7.4
Reduce	**15.9%**	**16.7%**	**15.9%**	**12.8%**	**13.5%**	**12.8%**	**26.6%**	**20.5%**	**26.8%**

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 $s_i^k \in \mathcal{S}^k$ be the clock nodes at level k, and divide these nodes as several clusters with $u_j^k \in \mathcal{U}^k$. Let $n_j^k \in \mathcal{N}^k$ be the net of u_j^k, and s_j^{k+1} be the driver node of net n_j^k. Let $L(s_j^{k+1})$ be the set of leaf nodes of s_j^{k+1} and $delay(s_j^{k+1}, s_i)$ be the delay from s_j^{k+1} to its leaf node $s_i \in L(s_j^{k+1})$. For level k, there are its corresponding constraints on skew, capacitance, fanout and wirelength. That is, for each clock net $n_j^k \in \mathcal{N}^k$, the following constraints should be satisfied $\max_{s_i \in L(s_j^{k+1})} \{delay(s_j^{k+1}, s_i)\} - \min_{s_i \in L(s_j^{k+1})} \{delay(s_j^{k+1}, s_i)\} \leq skew\_bound^k$, $|u_j^k| \leq max\_fanout^k$, $WL(n_j^k) \leq max\_length^k$, and $\sum_{s_i^k \in u_j^k} Cap_{pin}(s_i^k) + c \cdot WL(n_j^k) \leq max\_cap^k$.

Our hierarchical clock tree synthesis framework is given in Fig. 3. To manage fanout and minimize capacitance, we employ a balanced K-means and min-cost flow clustering for efficient clock node allocation. For optimizing capacitance, wirelength, and fanout together, a simulated annealing (SA) method is proposed for superior partitioning. Post-clustering, our CBS routing topology method achieves a desirable SLLT with low latency, capacitance, and skew compliance. Buffer insertion during clock net generation is guided by driver capability and buffer delay estimation.

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: $Cost^k = p \cdot \sigma(Cap^k) + q \cdot \sigma(T^k)$, where $Cap^k = (Cap_1^k, Cap_2^k, \cdots, Cap_{|\mathcal{N}^k|}^k)$, and Cap_j^k is the total capacitance of clock net $n_j^k \in \mathcal{N}^k$. Similarly, $T^k = (T_1^k, T_2^k, \cdots, T_{|\mathcal{N}^k|}^k)$, where $T_j^k = \max_{s_i \in L(s_j^{k+1})} \{delay(s_j^{k+1}, s_i)\}$. 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 wirelenth 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.

3.3 Routing Topology Generation

The routing topology generated during the CTS phase provides essential guidance for the subsequent Routing phase. Therefore, we need to use different topology generation methods according to the specific design requirements. Within our architecture, it serves three primary functions:

- During the violation detection process, it is necessary to quickly generate routing solutions to assess the occurrence of violations.

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.

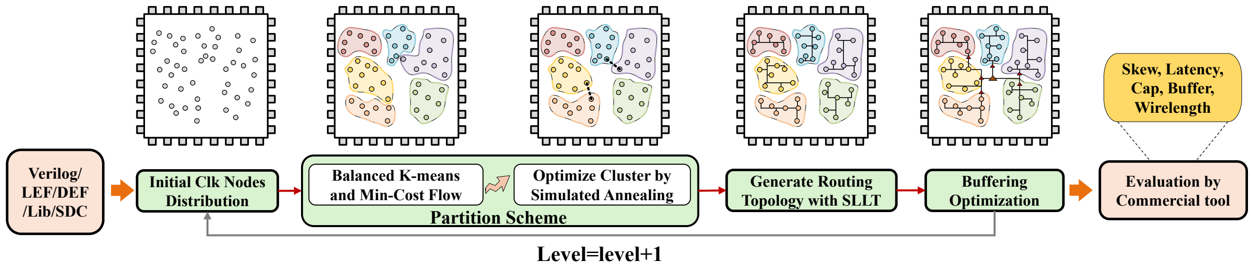

Figure 3: Hierarchical CTS Flow. (1) Achieve initial partition using balanced K-means and MCF, and optimize the partitioning result using simulated annealing. (2) Generate routing topology. (3) Find the buffering solution.

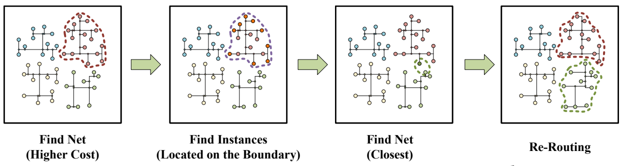

Figure 4: Simulated annealing operation. (1) Finding one n_j^k of nets with large cost. (2) Finding all instances ($\mathcal{BS}_j^k$) located at the boundary (convex hull) of n_j^k. (3) Finding the net n_l^k that is closest to these specific instances $s_l \in \mathcal{BS}_j^k$. (4) Adding s_l into n_l^k, and re-routing.

- 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 $D_{buf}(t)$ equation dependent on input slew $Slew_{in}(t)$ and load capacitance $Cap_{load}(t)$:

$$D_{buf}(t) = \omega_s \cdot Slew_{in}(t) + \omega_c \cdot Cap_{load}(t) + \omega_i, \tag{6}$$

where ω_s, ω_c, and ω_i represent slew, capacitance coefficients, and inherent delay, respectively. We consider the following scenario: buf_i and buf_j linked by a wire with buf_k inserted between them. Based on Equation (6), we can deduce the following result:

$$T(i,j) - T'(i,j) = \frac{r \cdot c \cdot (\ln 9 \cdot \omega_s + 1) \cdot L(i,j)^2}{4} - \omega_c \cdot Cap_{pin} - \omega_i,$$

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**: $L(i,j) = 2\sqrt{\frac{\omega_c \cdot Cap_{pin} + \omega_i}{r \cdot c \cdot (\ln 9 \cdot \omega_s + 1)}}$. By substituting Cap_{pin} with Cap_{load}, then $\hat{L}(i,j) = 2\sqrt{\frac{\omega_c \cdot Cap_{load} + \omega_i}{r \cdot c \cdot (\ln 9 \cdot \omega_s + 1)}}$, we obtain a refined estimation as Cap_{load} accounts for pin capacitance. Despite ignoring complex net slew effects, this approximation, favoring capacitance, is deemed acceptable for our analysis.

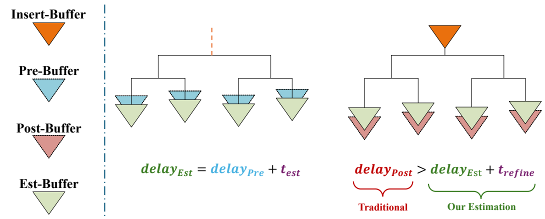

Figure 5: Buffering estimation.

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:

$$\hat{D}_{buf}(t) = \min_{\forall lib \in Lib}\{\omega_c(lib)\} \cdot Cap_{load}(t) + \min_{\forall lib \in Lib}\{\omega_i(lib)\}. \tag{7}$$

In Fig. 5, we estimate delay t_{est} prior to buffer insertion, initially increasing delay. Post-buffer insertion, accounting for downstream $Cap_{load}(t)$ and ω_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.

Case	s38584	s38417	s35932	salsa20	ethernet	vga_lcd	ysyx_0	ysyx_1	ysyx_2	ysyx_3
#Insts.	7510	6428	6113	13706	39945	60541	86933	93907	139178	139956
#FFs	1248	1564	1728	2375	10015	16902	18487	19090	27078	22810
Util	0.60	0.61	0.58	0.68	0.61	0.55	0.93	0.868	0.814	0.722

Table 5: Constraint list.

Constraint	Skew	Fanout	Cap	Wirelength
Val.	$80ps$	32	$150fF$	$300\mu m$

We performed experiments utilizing the $28nm$ 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*

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

Case	Latency (*ps*)			Skew (*ps*)			#Buffers			Buf Area (μm^2)			Clk Cap (*fF*)			Clk WL (μm)			Runtime (*s*)		
	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.
s38584	**71**	76	93	13	**8**	17	**43**	**43**	45	**26.6**	26.8	39.7	**904**	1019	1087	3382.0	**3366.5**	3478.9	**13**	326	52
s38417	**75**	84	94	**10**	19	16	**53**	54	55	**32.4**	33.6	48.5	**1083**	1235	1284	**3755.6**	3867.4	3870.5	**14**	357	47
s35932	**80**	81	100	13	**10**	17	**58**	59	64	**35.5**	36.5	56.4	**1217**	1380	1433	**4380.0**	4420.9	4449.4	**15**	334	52
salsa20	**82**	87	112	**19**	21	29	**81**	83	109	**49.6**	51.8	96.1	**1715**	2050	2160	**6446.9**	6580.9	6863.5	**17**	390	54
ethernet	**97**	104	159	34	**31**	51	**337**	352	455	**315.5**	320.4	408.9	**7314**	8823	9210	26113.9	**26105.5**	27248.8	**30**	416	50
vga_lcd	**134**	146	206	**41**	49	92	**575**	597	775	**416.9**	451.8	812.1	**12380**	14920	15815	46763.1	**45969.8**	47484.1	**55**	1271	59
Avg.	**1.000**	1.072	1.417	**1.000**	1.062	1.708	**1.000**	1.036	1.310	**1.000**	1.051	1.668	**1.000**	1.196	1.259	1.000	**0.994**	1.028	**1.000**	21.486	2.181

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

Case	Latency (*ps*)			Skew (*ps*)			#Buffers			Buf Area (μm^2)			Clk Cap (*fF*)			Clk WL (μm)			Runtime (*s*)		
	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.	Ours	Com.	OR.
ysyx_0	**113**	120	156	31	**15**	63	**639**	654	799	**550.368**	561.456	1719.018	29032	31662	**17130**	**42698.49**	42935.09	45389.69	51	573	**43**
ysyx_1	**118**	122	206	29	**12**	**110**	**656**	674	863	568.386	**568.26**	1896.426	29455	32204	**17907**	**44538.49**	45142.38	48113.24	55	556	**43**
ysyx_2	**140**	143	191	38	**16**	68	**943**	958	1113	**801.234**	814.59	2401.686	35272	39256	**25491**	**64884.11**	64901.76	69753.65	92	750	**45**
ysyx_3	144	**139**	193	36	**16**	59	**798**	808	914	**671.832**	688.968	1970.388	32483	35830	**21484**	57229.57	**56918.98**	59314.99	**85**	1350	148
Avg.	**1.000**	1.017	1.449	1.000	**0.440**	2.239	**1.000**	1.019	1.215	**1.000**	1.016	3.082	1.000	1.101	**0.650**	**1.000**	1.003	1.063	1.000	11.410	**0.986**

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 achieves 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 demonstrates superior performance across all measured indicators, while maintaining a similar total wirelength compared to the commercial tool.

We conduct further experiments on our actual design *ysyx* as shown in Table 4. The results of these experiments are presented in Table 7. Our solution exhibits superior performance across various metrics, including **maximum latency**, **buffer count**, **buffering area**, and **total wirelength**. Compared to the commercial tool, both solutions remain within the acceptable range dictated by the constraints. In addition, OpenROAD's solution violates the skew constraints in certain designs. Regarding clock capacitance, OpenROAD minimizes by employing a large number of larger buffers.

5 CONCLUSION

Under the concept of SLLT, 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 SLLT 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. PCL2023A03), the National Natural Science Foundation of China (No. 62174033, No. 61907024).

REFERENCES

[1] 1989. ISCAS'89 Benchmarks Circuits. (1989). http://www.cbl.ncsu.edu
[2] 2019. OpenROAD. (2019). https://github.com/The-OpenROAD-Project
[3] 2023. Cadence Inc. Innovus. (2023). https://www.cadence.com
[4] 2023. OpenCores: Open Source IP-Cores. (2023). http://www.opencores.org
[5] 2023. OpenLane CI Designs. (2023). https://github.com/efabless/openlane-ci-designs
[6] 2023. Synopsys Inc. Design Compiler. (2023). https://www.synopsys.com
[7] Ameer Abdelhadi, Ran Ginosar, Avinoam Kolodny, and Eby G Friedman. 2013. Timing–driven variation–aware synthesis of hybrid mesh/tree clock distribution networks. *Integration* 46, 4 (2013), 382–391.
[8] Alexander Andreev, Andrey Nikishin, Sergey Gribok, Phey-Chuin Tan, and Choon-Hun Choo. 2014. Clock network fishbone architecture for a structured ASIC manufactured on a 28 NM CMOS process lithographic node. (Jan. 14 2014). US Patent 8,629,548.
[9] Halil B Bakoglu. 1990. Circuits, interconnections, and packaging for VLSI. *(No Title)* (1990).
[10] Tuck-Boon Chan, Kwangsoo Han, Andrew B Kahng, et al. 2014. OCV-aware top-level clock tree optimization. In *Proc. of GLSVLSI*. 33–38.
[11] Ting-Hai Chao, Yu-Chin Hsu, Jan-Ming Ho, and AB Kahng. 1992. Zero skew clock routing with minimum wirelength. *IEEE Trans. on CAS II* 39, 11 (1992), 799–814.
[12] Gengjie Chen and Evangeline FY Young. 2019. Salt: provably good routing topology by a novel steiner shallow-light tree algorithm. *IEEE Trans. on CAD* 39, 6 (2019), 1217–1230.
[13] Chris Chu and Yiu-Chung Wong. 2007. FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE Trans. on CAD* 27, 1 (2007), 70–83.
[14] Jason Cong, Andrew B Kahng, Cheng-Kok Koh, and Chung-Wen Albert Tsao. 1998. Bounded-skew clock and Steiner routing. *ACM Trans. on DAES* 3, 3 (1998), 341–388.
[15] Michael Elkin and Shay Solomon. 2015. Steiner shallow-light trees are exponentially lighter than spanning ones. *SIAM J. Comput.* 44, 4 (2015), 996–1025.
[16] Kwangsoo Han, Andrew B Kahng, and Jiajia Li. 2018. Optimal generalized H-tree topology and buffering for high-performance and low-power clock distribution. *IEEE Trans. on CAD* 39, 2 (2018), 478–491.
[17] Dake Liu and Christer Svensson. 1994. Power consumption estimation in CMOS VLSI chips. *IEEE Journal of Solid-State Circuits* 29, 6 (1994), 663–670.
[18] Can Sitik, Scott Lerner, and Baris Taskin. 2014. Timing characterization of clock buffers for clock tree synthesis. In *Proc. of ICCD*. IEEE, 230–236.
[19] Can Sitik, Weicheng Liu, Baris Taskin, and Emre Salman. 2016. Design methodology for voltage-scaled clock distribution networks. *IEEE Trans. on VLSI* 24, 10 (2016), 3080–3093.
[20] Chung-Wen Albert Tsao and Cheng-Kok Koh. 2002. UST/DME: a clock tree router for general skew constraints. *ACM Trans. on DAES* 7, 3 (2002), 359–379.
[21] Kenneth Yip. 1997. Clock tree distribution. *IEEE Potentials* 16, 2 (1997), 11–14.