Latency Constraint Guided Buffer Sizing and Layer Assignment for Clock Trees with Useful Skew

Necati Uysal*, Wen-Hao Liu†, Rickard Ewetz*
*University of Central Florida, Orlando, FL, 32816, USA
†Cadence Design Systems Inc, Austin, TX, 78759, USA
necati@knights.ucf.edu,rickard.ewetz@ucf.edu

ABSTRACT

Closing timing using clock tree optimization (CTO) is a tremendously challenging problem that may require designer intervention. CTO is performed by specifying and realizing delay adjustments in an initially constructed clock tree. Delay adjustments are typically realized by inserting delay buffers or detour wires. In this paper, we propose a latency constraint guided buffer sizing and layer assignment framework for clock trees with useful skew, called the (BLU) framework. The BLU framework realizes delay adjustments during CTO by performing buffer sizing and layer assignment. Given an initial clock tree, the BLU framework first predicts the final timing quality and specifies a set of delay adjustments, which are translated into latency constraints. Next, buffer sizing and layer assignment is performed with respect to the latency constraints using an extension of van Ginneken's algorithm. Moreover, the framework includes a feature of reducing the power consumption by relaxing the latency constraints and a method of improving the timing performance by tightening the latency constraints. The experimental results demonstrate that the proposed framework is capable of reducing the capacitive cost with 13% on the average. The total negative slack (TNS) and worst negative slack (WNS) are reduced with up to 58% and 20%, respectively.

1 INTRODUCTION

Power consumption has become a primary design constraint in advanced technology nodes. A large portion of the power consumption of each VLSI circuit is consumed by the clock tree used to synchronize the sequential elements. Clock trees must satisfy strict timing, or skew, constraints to guarantee the functional correctness of each circuit. Clock skew is the difference in the arrival time of the clock signal between a pair of sequential elements, or clock sinks. There is a skew constraint between each pair of clock sinks that are only separated by combinational logic in the data and control paths. The timing constraint must be satisfied even while the circuit is subject to on-chip variations (OCV). To meet both skew and power constraints under variations, it has become necessary to utilize every available timing margin by exploiting useful skew.

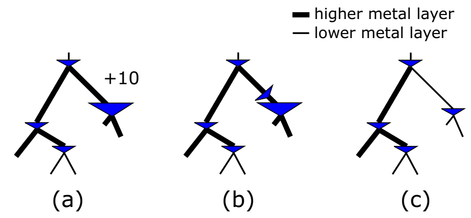


Figure 1: (a) A specified delay adjustment. (b) Delay adjustment realized by buffer insertion. (c) Proposed realization of delay adjustments using buffer sizing and layer assignment.

The construction of useful skew trees (USTs) has been explored in [4, 7]. USTs can be constructed by first specifying a set of latency constraints. Next, a clock tree satisfying the latency constraints is constructed. Latency constraints consist of a lower and upper bound on the arrival time of the clock signal to each clock sink with respect to the clock source. Latency constraints in the form of points were specified in [5]. The point constraints were extended into latency ranges in [1].

After an initial clock tree has been constructed, clock tree optimization (CTO) is applied to remove timing violations by specifying and realizing delay adjustments [3, 11, 12]. Note that the use of delay adjustments is equivalent to specifying latency constraints in the form of points [11, 12] or ranges [3]. A delay adjustment is a change of the propagation delay through a branch in the clock tree, which is illustrated in Figure 1(a). Delay adjustments are traditionally realized by inserting delay buffers, which is shown in Figure 1(b). In contrast, this paper proposes to realize delay adjustments using buffer sizing and layer assignment, which is illustrated in Figure 1(c). Compared with buffer insertion, layer assignment is a more gentle method of realizing delay adjustments that may result in lower power consumption.

Buffer sizing (or gate sizing) and layer assignment can be performed to save power while meeting constraints on the maximum latency (or propagation delay). Van Ginneken's algorithm is a well known technique based on dynamic programming [9, 17]. Buffer sizing and layer assignment for zero skew and bounded skew clock trees has been studied in [2, 10, 15, 18]. Buffer sizing for USTs was performed using a Taylor expansion and sequential linear programming in [6, 18]. Nevertheless, it is difficult to handle discrete buffer sizes and layer assignments using linearization.

In this paper, we present a latency constraint guided buffer sizing and layer assignment framework for clock trees with useful skew, called the (BLU) framework. The framework is applied after an initial clock tree has been constructed and before traditional

*This research was in part supported by NSF award CFF-1755825.

ASPDAC '19, January 21–24, 2019, Tokyo, Japan
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6007-4/19/01…$15.00
https://doi.org/10.1145/3287624.3287681

CTO is applied. The key idea is to perform CTO by realizing delay adjustments using buffer sizing and layer assignment.

The BLU framework specifies a set of delay adjustments and predicts the final timing quality (P_{tns}, P_{wns}) that would be achieved using traditional CTO. P_{tns} and P_{wns} are respectively the predicted total negative slack (TNS) and worst negative slack (WNS). Next, the delay adjustments are translated into latency constraints without degrading P_{tns} and P_{wns}. Using the specified latency constraints, buffer sizing and layer assignment is conducted using an extension of van Ginneken's algorithm, i.e., delay adjustments are realized while reducing the total capacitive cost. To further reduce power consumption, the BLU framework attempts to relax each point constraints into a latency range without degrading P_{tns} and P_{wns}. Moreover, a method of improving P_{tns} and P_{wns} by specifying tight latency constraints using negative delay adjustments is proposed. Lastly, traditional CTO is applied to realize remaining delay adjustments such that TNS and WNS are reduced to P_{tns} and P_{wns}.

Compared with in [6, 18], the BLU framework allows buffer sizing and layer assignment to be performed while utilizing discrete buffer and interconnect libraries. We consider the BLU framework to be orthogonal to the techniques of realizing negative delay adjustments by reconstructing the topology of a clock tree in [12, 16].

The experimental results demonstrate that the BLU framework is capable of reducing total capacitance, total negative slack (TNS) and worst negative slack (WNS) with 13%, 58%, and 20%, respectively.

2 PRELIMINARIES

Every sequential circuit requires the clock signal to be delivered to the sequential elements, or flip flops (FFs), meeting setup and hold time constraints. There is a setup and hold time constraint between each pair of FFs that are only separated by combinational logic in the data and control paths. After a clock tree has been constructed, the slack in the constraints can be computed, as follows:

$$setup\_slack_{ij} = t_j - t_i + T - t_j^S - t_{ij}^{CQ} - t_{ij}^{max} - \delta_i - \delta_j, \tag{1}$$

$$hold\_slack_{ij} = t_i - t_j + t_{ij}^{CQ} + t_{ij}^{min} - t_j^H - \delta_j - \delta_i, \tag{2}$$

where $setup\_slack_{ij}$ and $hold\_slack_{ij}$ are the slacks in the respective constraints. A negative slack implies a violation of a timing constraint. t_i and t_j are the arrival times of the clock signal to FF$_i$ and FF$_j$, respectively. t_{ij}^{min} (t_{ij}^{max}) is the minimum (maximum) propagation delay through the combinational logic between FF$_i$ and FF_j. t_i^{CQ} is the clock to output of FF$_i$; T is the clock period; t_j^S and t_j^H are respectively the setup and hold time of FF_j. δ_i and δ_j are timing deteriorates introduced by OCV.

The timing deteriorates δ_i and δ_j are dependent on the distance between FF$_i$ and FF$_j$ in the clock tree topology. Let the closest common ancestor (CCA) between FF$_i$ and FF$_j$ in the clock tree be denoted CCA(i,j) [8]. Based on the model in [12], 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. The c_{ocv} parameter is set to 0.085.

The slack in the timing constraints can be captured in a slack graph (SG) [3]. In an SG $G = (V, E)$, the vertices V represent clock sinks and the edge weights E represent the slack in the timing constraints. An edge e_{ij} with weight $w_{ij} = setup\_slack_{ij}$ is added for each setup time constraint. An edge e_{ji} with weight $w_{ji} = hold\_slack_{ij}$ is added for each hold time constraint. The timing quality of a clock tree is measured in TNS and WNS, i.e., the sum and the maximum of the negative timing slacks in Eq (1) and Eq (2). The objective of this paper is to minimize TNS and WNS.

3 PREVIOUS WORK

In this section, we first review techniques for CTO and an extension of van Ginneken's algorithm. Next, we outline how the two algorithms are combined in the proposed framework.

3.1 Predicted timing quality and CTO

Timing violations in constructed clock trees are typically eliminated by realizing non-negative delay adjustments [3, 11, 12]. A delay adjustment is a change of the propagation delay through a branch in the clock tree. Let $\triangle_k \geq 0$ be a delay adjustment at a location k in a clock tree. Delay adjustments are typically restricted to locations where buffers are placed in the topology to avoid disrupting the overall timing [12]. Next, the final timing quality is predicted by specifying a set of delay adjustments using an LP formulation, as follows [3, 12]:

$$\begin{aligned} \min \quad & c_t \sum_{k \in B} \triangle_k + c_{wns} P_{wns} + c_{tns} P_{tns} \\ s.t. \quad & \sum_{k \in path(CCA(i,j),i)} (1 + c_{ocv}) \triangle_k - \sum_{h \in path(CCA(i,j),j)} (1 - c_{ocv}) \triangle_h - s_{ij} \leq w_{ij}, \\ & \qquad (i,j) \in E, \\ & s_{ij} \leq P_{wns}, \quad (i,j) \in E, \\ & \sum_{(i,j) \in E} s_{ij} = P_{tns}, \end{aligned} \tag{3}$$

where $path(i, CCA(i,j))$ and $path(j, CCA(i,j))$ respectively denote the buffers on the paths from $CCA(i,j)$ to FF$_i$ and FF$_j$. w_{ij} is the weight of an edge in the SG. $s_{ij} \geq 0$ is a timing violation that is not eliminated by realizing the specified delay adjustments. P_{tns} and P_{wns} are respectively the predicted TNS and WNS that is achieved by realizing the specified delay adjustments. The c_t, c_{wns} and c_{tns} parameters are used to balance the different terms in the objective function. The $(1+c_{ocv})$ and $(1-c_{ocv})$ factors account for the timing deteriorates introduced by the specified delay adjustments.

3.2 Van Ginneken's algorithm [17]

Van Ginneken's algorithm is well known dynamic programming algorithm that minimizes the latency of an RC tree using buffer sizing and layer assignment under the Elmore delay model [17]. In [9], the algorithm was extended to find all Pareto optimal solutions in terms of power consumption and latency while considering slew propagation. Moreover, it is straightforward to set different latency constraints for different clock sinks.

The algorithm solves the problem of selecting a buffer size for each buffer and a layer assignment for each wire in a clock tree by propagating candidate solutions from the leaf nodes to the source node, which is illustrated in Figure 2. Each candidate c_k stores the maximal downstream delay d_k, non-shielded downstream capacitance cap_k, and cost in terms of total capacitance $cost_k$, i.e.,

$c_k = (d_k, cap_k, cost_k)$ [9]. First, a candidate solution with zero maximum downstream delay is created at each clock sink. Both the non-shielded capacitance and the cost are set equal to the sink capacitance, which is illustrated in the bottom-left of Figure 2.

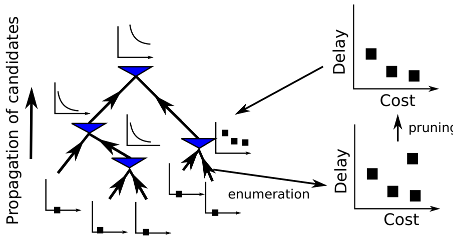


Figure 2: Extension of van Ginneken's algorithm [9].

The candidate solutions at the sinks are then propagated up towards the root of the tree. When a candidate solution is propagated through an edge (buffer or wire), all possible realizations (buffer sizes or layer assignments) are enumerated as candidates to realize the edge, shown in bottom-right of Figure 2. New candidates are also formed when two branches in the clock tree are joined. Pruning is applied to eliminate non-Pareto optimal solutions, which is shown in the top-right of Figure 2. Next, the minimum cost candidate solution that meets a defined latency requirement is selected at the root. Lastly, the size for each buffer and the layer assignment for each wire is determined based on the selected candidate.

3.3 Proposed framework

Non-negative delay adjustments are realized by inserting buffers and detour wires during CTO, which translates into overhead in terms of total capacitance. In contrast, van Ginneken's algorithm (with the extension in [9]) is capable of trading-off maximum delay for total capacitive cost. The key idea of the BLU framework is to use buffer sizing and layer assignment to realize delay adjustments. Consequently, the proposed framework has the potential to improve power consumption while reducing TNS and WNS to P_{tns} and P_{wns}, respectively. Extensions to further improve performance are presented in Section 4.

Van Ginneken's algorithm requires maximum delay (or latency) constraints. The latency constraints are obtained from the delay adjustments specified using Eq (3). The obtained constraints are in the form of points. Therefore, each delay adjustment is required to be realized exactly. Let l_i denote the upper bound of the latency constraint to sink i. The constraints requires each arrival time t_i to be equal to the latency constraint l_i. However, van Ginneken's algorithm only ensures that $t_i \leq l_i$. Nevertheless, it is expected that the arrival times t_i will be close to l_i, as increasing the arrival time (or delay) typically results in a reduction of total capacitive cost. Moreover, the difference between t_i and l_i can be realized through traditional CTO after the proposed framework has been applied.

4 THE BLU FRAMEWORK

The baseline of the BLU framework is presented in Section 4.1. In Section 4.2, point constraints are relaxed into range constraints to save capacitive cost. In Section 4.3, the latency constraints are tightened to improve P_{tns} and P_{wns}.

4.1 Baseline of the BLU framework

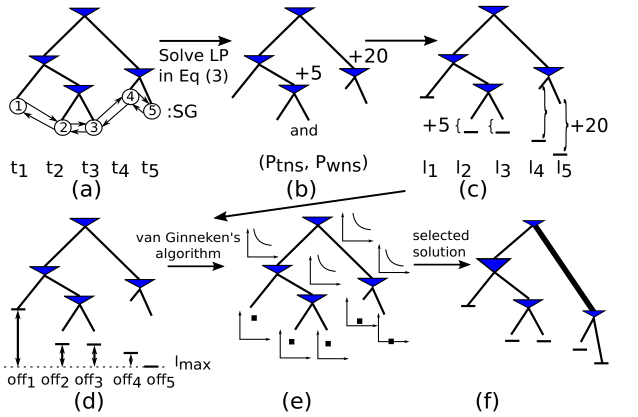


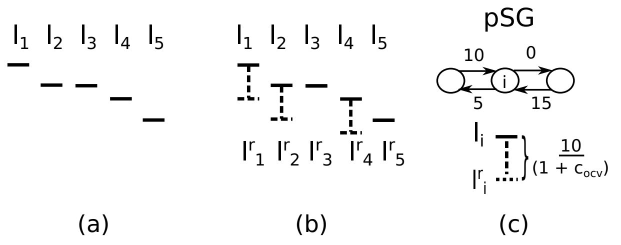
Figure 3: (a) SG. (b) P_{tns} and P_{wns} and delay adjustments. (c) latency constraints. (d) offsets. (e) van Ginneken's algorithm. (f) selected candidate solution.

The BLU framework is illustrated with an example in Figure 3. First, an SG is formed based on the timing and the topology of the initial clock tree, as illustrated in Figure 3(a). Next, the LP formulation in Eq (3) is solved to specify delay adjustments and to predict P_{tns} and P_{wns}, which is illustrated in Figure 3(b). The latency constraint l_i for each clock sink i is obtained by floating down each delay adjustments to the clock sinks and combining the adjustments with the current arrival time of the clock signal, which is shown in Figure 3(c). The maximum latency constraint is denoted l_{max}. Next, an offset off_i equal to, $l_{max} - l_i$, is introduced for each clock sink, as illustrated in Figure 3(d). A candidate solution c_i (in van Ginneken's algorithm) is created at each clock sink i with a maximum downstream delay equal to off_i, which is illustrated in Figure 3(e). Subsequently, van Ginneken's algorithm is applied and the minimum cost candidate solution that satisfies l_{max} is selected at the root. The latency constraint l_{max} at the root ensures that each arrival time t_i is smaller than l_i. Moreover, it is guaranteed that at least one candidate solution will satisfy l_{max} at the root of the clock tree, i.e., the initial clock tree. Figure 3(f) shows the clock tree after buffer sizing and layer assignment has been performed with respect to the selected candidate solution. Compared with the clock tree in 3(a), the clock tree in Figure 3(f) will have the same predicted timing quality but smaller capacitive cost.

4.2 Relaxing the latency constraints

In this section, the BLU framework is extended to enable further savings in capacitive cost by relaxing the point constraints into latency range constraints. The latency constraints specified by Eq (3) are shown in Figure 4(a). The relaxed latency range constraints are shown in Figure 4(b). The BLU framework specifies the range constraints while guaranteeing that P_{tns} and P_{wns} are not degraded if every arrival time t_i is within the respective latency range. l_i^r is the relaxed latency constraint for clock sink i and $l_i \leq l_i^r$.

The method used to find the latency range constraint is illustrated in Figure 4(c). First, a predicted slack graph (pSG) is formed [16]. A pSG, captures the predicted slacks in the timing constraints after the delay adjustments specified by Eq (3) are realized. Moreover,



l_1 l_2 l_3 l_4 l_5
l_1 l_2 l_3 l_4 l_5
l^r_1 l^r_2 l^r_3 l^r_4 l^r_5
pSG
10
0
5
15
i
l_i
l^r_i
$\frac{10}{(1 + c_{ocv})}$
(a)
(b)
(c)

Figure 4: (a) Latency constraints. (b) Relaxed latency constraints. (c) Method to find l^r_i.

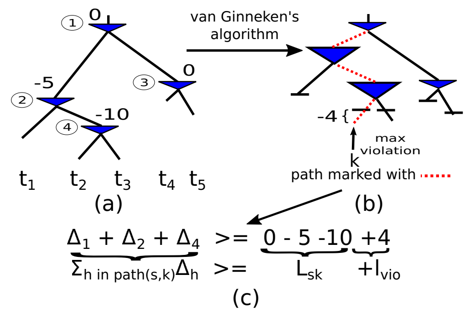
the predicted slack violations s_{ij} are added to the respective edges in the pSG such that all edges in the pSG are non-negative. Next, a relaxed latency constraint l^r_i is found, as follows:

$$l^r_i = l_i + \frac{w^{min}_{ki}}{(1 + c_{ocv})}, \tag{4}$$

where w^{min}_{ki} is the edge with the minimum weight of all the fan-in edges of node i in the pSG, i.e., $e_{ki} \in E$. The $(1 + c_{ocv})$ factor used to compensate for the increased timing deteriorates δ_i and δ_j.

4.3 Tightening the latency constraints

In this section, the BLU framework is extended by allowing negative delay adjustments to be specified in the clock tree, which may improve P_{wns} and P_{tns}, as illustrated in Figure 5.



van Ginneken's algorithm
0
-5
0
-10
1
2
3
4
t_1 t_2 t_3 t_4 t_5
-4
max violation
k
path marked with
(a)
(b)
$\underbrace{\Delta_1 + \Delta_2 + \Delta_4}_{\Sigma_{h \text{ in } path(s,k)} \Delta_h} >= \underbrace{0 - 5 - 10}_{L_{sk}} \underbrace{+4}_{+l_{vio}}$
(c)

Figure 5: (a) Delay adjustments specified by Eq (3). (b) Clock tree after van Ginneken's algorithm. l_{max} is violated by the path marked with a red dashed line. (c) Generation of a delay adjustment constraint.

Compared with non-negative delay adjustments that are relatively easy to realize, it may be impossible to physically realize negative adjustments [12]. Consequently, it may be impossible to satisfy the latency constraint l_{max} using Van Ginneken's algorithm, which results in that TNS and WNS cannot be reduced to the predicted P_{tns} and P_{wns} using traditional CTO. The BLU framework solves this challenge by generating *delay adjustment constraints* to the LP formulation in Eq (3), which ensures that only negative delay adjustments that can be realized are specified.

First, Eq (3) is solved while allowing both non-negative and negative delay adjustments, which is shown in Figure 5(a). Negative delay adjustments are facilitated by replacing the expression $c_t \sum_{k \in B} \triangle_k$ with $c_p \sum_{k \in B} \triangle^+ - c_n \sum_{k \in B} \triangle^-$ in Eq (3), where $\triangle^+ \geq 0$ and $\triangle^- \geq 0$ are non-negative and negative delay adjustments, respectively. c_p and c_n are user defined parameters. Next, the remainder of the BLU framework is applied and the resulting clock tree is shown in Figure 5(b). If any candidate solution satisfies the latency constraint l_{max} at the root, the same flow as for non-negative delay adjustments is applied (see Section 4.1).

If no candidate solution meets the latency constraint l_{max}, the BLU framework selects the candidate solution that is closest to satisfying the latency constraint. Let l_{vio} be the violation of the latency constraint and let $path(s, k)$ be the path from the source to clock sink k that created the largest latency violation, which is illustrated with a dashed red line in Figure 5(b). It is straightforward to find the path based on backtracking the candidate solutions generated by van Ginneken's algorithm. Next, a delay adjustment constraint is generated to force the delay adjustments on the $path(s, k)$ to be l_{vio} larger than currently specified, as follows:

$$\sum_{h \in path(s,k)} \triangle_h \geq L_{sk} + l_{vio}, \tag{5}$$

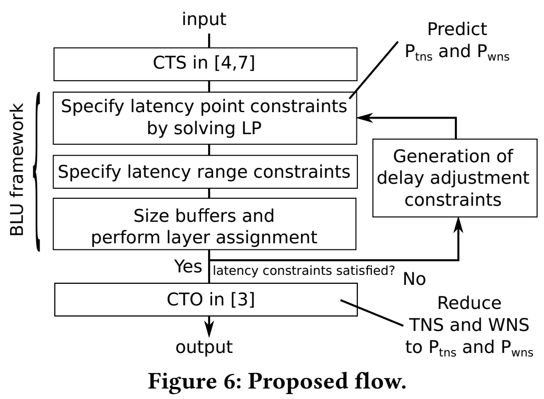
where L_{sk} is the sum of the delay adjustments on the path from the source to sink k in the current solution of Eq (3). $\triangle_h$ are variables in Eq (3).

A new set of latency constraints are specified by solving the LP formulation in Eq (3) in combination with the delay adjustment constraints in Eq (5), which is shown in Figure 5(c). The process is iteratively repeated until the latency constraint l_{max} is satisfied. P_{tns} and P_{wns} are increased in each iteration as additional constraints are introduced to the LP formulation. Latency constraints are also introduced at internal nodes to speed up the convergence process. Violations of latency constraints at internal nodes are accounted for by modifying the $cost_k$ of a candidate to be equal to $cost_i = cap^{tot} + c_{vio} \cdot l^{tot}_{vio}$, where l^{tot}_{vio} is the sum of the violations in the downstream subtree and c_{vio} is a user specified parameter.

5 METHODOLOGY

The flow of the framework is illustrated in Figure 6. First, an initial clock tree is constructed using CTS [3, 7], which is the input to the BLU framework. Next, the BLU framework is performed, as described in Section 4. Lastly, CTO is performed to reduce TNS and WNS to P_{tns} and P_{wns} using the techniques in [3].The high level flow for the BLU framework is outlined below.

A *specify latency point constraints* step is performed to predict P_{tns} and P_{wns} and to specify a set of latency constraints in the form of points. The point constraints are extended into latency ranges in a *specify latency range constraints* step. Next, *buffer sizing and layer assignment* step is performed using an extension of van Ginneken's algorithm that utilizes three-dimensional sampling and transition time constraints [14]. If the latency constraints are satisfied, buffer



input
CTS in [4,7]
Predict P_{tns} and P_{wns}
BLU framework
Specify latency point constraints by solving LP
Specify latency range constraints
Size buffers and perform layer assignment
Generation of delay adjustment constraints
Yes
latency constraints satisfied?
No
CTO in [3]
Reduce TNS and WNS to P_{tns} and P_{wns}
output

Figure 6: Proposed flow.

Table 1: Evaluation of various tree structures in terms of total capacitance and run-time.

Circuit	Cap (pF)						Run-Time (ps)					
(name)	UST [4]	UST-CTO [4]	UST-P	UST-P-CTO	UST-R	UST-R-CTO	UST [4]	UST-CTO [4]	UST-P	UST-P-CTO	UST-R	UST-R-CTO
s1423	3.43	3.43	3.02	3.02	2.96	2.96	0.0	0.1	0.0	0.1	0.1	0.1
s5378	5.65	5.65	5.04	5.04	4.87	4.87	0.0	0.1	0.1	0.1	0.1	0.1
s15850	18.09	18.85	15.84	16.86	15.77	16.62	0.2	2.3	0.3	14.7	0.2	2.0
msp	1.41	1.41	1.35	1.35	1.20	1.20	0.0	0.0	0.5	0.0	3.7	0.1
fpu	1.60	1.60	1.52	1.52	1.35	1.35	0.0	0.2	0.7	0.0	1.2	0.0
usbf	4.55	4.55	4.14	4.14	4.07	4.07	1.0	0.2	0.4	0.2	2.4	0.2
dma	5.06	5.17	4.49	4.65	4.44	4.56	1.0	2.1	1.1	2.5	10.3	2.1
pci	7.65	7.65	7.02	7.06	6.71	6.71	2.0	0.2	2.3	0.8	15.8	0.2
ecg	23.44	23.66	20.39	20.96	20.54	20.84	8.0	11.3	1.7	15.5	5.0	12.8
des	18.82	18.84	16.58	16.62	16.62	16.64	4.0	1.1	1.3	0.9	3.9	0.4
eht	20.14	20.14	17.85	17.85	17.62	17.62	8.0	0.6	28.6	0.4	107.2	0.4
aes	151.70	152.91	132.91	135.28	132.91	135.60	45.0	110.5	7.2	65.7	15.4	84.3
Norm.	0.99	**1.00**	0.89	0.90	0.87	**0.87**	0.30	1.00	0.60	1.10	1.20	1.70

sizing and layer assignment is performed. If the latency constraints are not satisfied, a *generation of delay adjustment constraints* step is performed to introduce delay adjustment constraints. Next, the framework returns to the specify point constraints step. The process is iteratively repeated until all the latency constraints are satisfied.

6 EXPERIMENTAL EVALUATION

The experimental evaluation is performed on a quad core 3.4 GHz Linux machine with 32GB of memory. The proposed algorithms are implemented in C++. IBM ILOG CPLEX is used to solve the LP formulations in the framework.

Table 2: Circuits in [4].

Circuit (name)	Sinks (num)	Skew constraints (num)
s1423	74	78
s5378	179	175
s15850	597	318
msp	683	44990
fpu	715	16263
usbf	1765	33438
dma	2092	132834
pci bridge32	3578	141074
ecg	7674	63440
des peft	8808	17152
eht	10544	450762
aes	13216	53382

The evaluation is performed using the framework proposed in [4], which is an extension of the problem formulation used in the ISPD 2010 contest [13]. The properties of the buffers and the wires are obtained from the 45 nm technology used in the ISPD 2010 contest. The non-uniform skew constraints and the sink locations are generated using Synopsys DC and ICC. Moreover, there is a transition time constraint at each buffer and clock sink. A summary of the circuits is shown in Table 2. We construct and compare eight different tree structures to evaluate the BLU framework.

(1) The UST structure is a clock tree constructed using the CTS engine provided by the authors in [4]. (2) The UST-CTO structure is the structure obtained by applying the CTO in [3] to the UST structure. (3) The UST-P structure is obtained by applying the BLU framework to the UST structure using point constraints, i.e., the framework presented in Section 4.1. (4) The UST-P-CTO is the structure obtained by applying CTO to the UST-P structure. (5) The UST-R structure is the UST-P structure obtained by relaxing the point constraints into latency ranges, i.e., the method described in Section 4.2. (6) The UST-R-CTO structure is the structure obtained by applying CTO to the UST-R structure. (7) The UST-RT structure is the UST-R structure combined with the technique of tightening the constraints proposed in Section 4.3. (8) The UST-RT-CTO structure is obtained by applying CTO to the UST-RT structure.

We evaluate the tree structures in terms of total capacitance, timing performance and run-time. It is well known that the power consumption of a clock tree is highly correlated with the total capacitance. The timing quality is evaluated using TNS and WNS, which are computed using Eq (1) and Eq (2); P_{tns} and P_{wns} are obtained from solving the Eq (3). The arrival times t_i and t_j are obtained using NGSPICE simulations. All tree structures in the experimental results satisfy the same transition time constraints as in [4]. In Section 6.1, we evaluate the BLU framework on the clock trees in [4], which only requires non-negative delay adjustments. In Section 6.2, we evaluate the BLU framework on the clock trees with strict timing constraints, which utilizes both non-negative and negative delay adjustments.

6.1 Evaluation of positive delay adjustments

In Table 2, the total capacitance and run-time are shown in the columns labeled as 'Cap' and 'Run-time', respectively. The normalized performances with respect to the UST-CTO structures are shown in the row labeled as 'Norm'. The run-times in the table are the run-times of individual synthesis steps. The normalized run-times are the cumulative run-times. The timing performance is not shown because there are no timing violations after CTO.

First, we apply traditional CTO to the UST structures. All timing violations are eliminated at the expense of an average 1% increase in capacitive cost. Compared with the UST structures, the UST-P structures have 10% lower capacitance. The capacitance reduction stems from that van Ginneken's algorithm assigns interconnects to lower metal layers and that buffers are downsized while still meeting the transition time constraints. Next, CTO is applied to the UST-P structures. The UST-P-CTO structures have 10% lower capacitance than the UST-CTO structures, as the CTO phase only resulted in a small increase in total capacitance. The capacitive improvements come at an expense of a 10% increase in run-time.

Next, we compare UST-R structures with UST-P structures. The table shows that the UST-R structures have 2% lower capacitance than the UST-P structures. The improvement in capacitance is a result of the relaxation of point constraints into range constraints. Ideally, the UST-R structures should have better capacitive performance than the UST-P structures on all circuits. However, the sampling in van Ginneken's algorithm may result in minor capacitance fluctuations (see circuits des and aes). After CTO is applied, it can be observed that the UST-R-CTO structures have 3% lower total capacitance than the UST-P-CTO structures. The average run-time of the UST-R-CTO structures is 1.5X higher than the UST-P-CTO structures. The UST-R structures demonstrate that the BLU framework is capable of performing buffer sizing and layer assignment to reduce capacitive cost without degrading timing performance. The improvements are achieved at the expense of overhead in run-time.

Table 3: Evaluation of negative delay adjustments.

Circuit (name)	M (ps)	Structure (name)	TNS (ps)	WNS (ps)	P_{tns} (ps)	P_{wns} (ps)	Cap (pF)	Run-time (min)
ecg	30	UST [4]	643	18	0	0	23.4	8.0
		UST-CTO [4]	0	0	0	0	23.7	11.3
		UST-R	352	18	0	0	20.5	5.0
		UST-R-CTO	0	0	0	0	20.8	12.8
		UST-RT	414	15	0	0	20.5	8.0
		UST-RT-CTO	0	0	0	0	20.8	7.1
	15	UST [4]	2218	19	50	2	19.7	9.2
		UST-CTO [4]	929	3	709	2	21.3	23.4
		UST-R	4286	24	23	1	17.7	6.1
		UST-R-CTO	150	4	86	3	19.5	17.6
		UST-RT	1544	18	0	0	18.7	29.9
		UST-RT-CTO	18	1	0	0	20.1	16.8
	0	UST [4]	15059	33	5299	22	17.6	4.5
		UST-CTO [4]	6259	25	5919	23	20.6	42.9
		UST-R	14865	41	6121	25	15.0	5.0
		UST-R-CTO	7136	28	6615	26	17.9	21.3
		UST-RT	16553	35	2882	19	17.4	21.3
		UST-RT-CTO	3332	20	3928	19	20.6	35.4
aes	50	UST [4]	1315	22	0	0	151.7	45.0
		UST-CTO [4]	0	0	0	0	152.9	172.5
		UST-R	3045	26	0	0	132.9	15.4
		UST-R-CTO	0	0	0	0	135.6	84.3
		UST-RT	3028	26	0	0	132.9	19.0
		UST-RT-CTO	0	0	0	0	135.5	86.4
	40	UST [4]	9897	33	2604	12	135.6	12.1
		UST-CTO [4]	3208	16	3064	14	145.8	110.6
		UST-R	25873	44	2200	11	119.1	21.6
		UST-R-CTO	2972	17	2500	14	128.9	143.4
		UST-RT	14118	36	2151	11	123.5	51.2
		UST-RT-CTO	2498	14	2413	13	132.8	19.2
	30	UST [4]	16041	32	7095	14	112.1	24.9
		UST-CTO [4]	8448	18	7950	15	121.6	139.0
		UST-R	36367	47	4478	13	97.8	9.2
		UST-R-CTO	5697	19	5186	15	111.8	73.5
		UST-RT	15685	36	2636	11	102.2	56.5
		UST-RT-CTO	3569	16	3330	14	113.4	189.2
Norm.	-	UST [4]	3.56	3.04	**1.00**	**1.00**	0.91	0.13
		UST-CTO [4]	**1.00**	**1.00**	-	-	**1.00**	1.00
		UST-R	6.33	3.84	0.77	0.88	0.79	0.26
		UST-R-CTO	0.71	1.13	-	-	**0.90**	0.95
		UST-RT	4.24	3.19	**0.44**	**0.63**	0.85	0.60
		UST-RT-CTO	**0.42**	**0.80**	-	-	**0.95**	1.21

6.2 Evaluation of negative delay adjustments

In this section, we focus on how the BLU framework performs on clock trees with non-zero P_{tns} and P_{wns}. The TNS, WNS, P_{tns}, P_{wns}, are respectively labeled 'TNS', 'WNS', 'P_{tns}' and 'P_{wns}' in Table 3. The evaluation is performed on the circuits *ecg* and *aes*. The synthesis tool in [4] is used to generate clock trees with different guard bands (labeled 'M'), which regulates a trade-off between total capacitance and timing performance.

First, traditional CTO is applied to the UST structures. The UST-CTO structures have 72%, 67% lower TNS and WNS than the UST structures, respectively. It can be observed that TNS and WNS after CTO is strongly correlated with P_{tns} and P_{wns} before CTO. On the other hand, the timing improvement result in a 10% increase in total capacitance, which stems from the insertion of delay buffers.

Compared with the UST structure, the UST-R structures have 13%, 23%, 12%, lower total capacitance, P_{tns}, and P_{wns}, respectively. The P_{tns}, and P_{wns} improvements stem from that the layer assignment of interconnects under the bottom most buffers. The TNS and WNS may be improved or degraded. However, the timing performance is expected to be recovered after CTO, as P_{tns} and P_{wns} are improved.

Compared with the UST-CTO structures, the UST-R-CTO structures have 10% lower capacitance and 5% shorter run-time. The TNS is 29% lower and WNS is 13% higher. Compared with the UST-R structures, the UST-RT structures have 43% and 28% lower P_{tns} and P_{wns}, respectively. The timing improvements come from realizing negative delay adjustments. The total capacitance of the UST-RT structures are 8% higher than the UST-R structures. The increase in total capacitance stems from realizing negative delay adjustments using large buffers. It is expected that the UST-RT structures and UST-R structures have similar performances on the clock trees with zero P_{tns} and P_{wns}, as no negative delay adjustments are specified. Compared with the UST-R-CTO structures, the UST-RT-CTO structures have 41%, 29%, lower TNS and WNS, respectively. The timing improvements of the UST-RT-CTO structures come at an expense of a 6% increase in total capacitance and a 27% increase in run-time.

The UST-R-CTO structures and UST-RT-CTO structures demonstrate that the BLU framework is capable of exploring a trade-off between timing quality and capacitive cost using negative delay adjustments. Moreover, the results of the UST-R-CTO structures and UST-RT-CTO structures are notably better than the UST-CTO structures in [4].

7 SUMMARY AND FUTURE WORK

In this paper, a latency constraint guided buffer sizing and layer assignment framework is proposed. The framework is capable of handling discrete buffer sizes and layer assignments while utilizing useful skew. In the future, we plan to integrate topological modifications into the proposed framework.

REFERENCES

[1] Christoph Albrecht et al. 2002. Maximum Mean Weight Cycle in a Digraph and Minimizing Cycle Time of a Logic Chip. *Discrete Applied Mathemathics* 123, 1-3 (2002), 103–127.

[2] Krit Athikulwongse, Xin Zhao, and Sung Kyu Lim. 2010. Buffered Clock Tree Sizing for Skew Minimization Under Power and Thermal Budgets *(ASP-DAC'10)*. 474–479.

[3] Rickard Ewetz. 2017. A Clock Tree Optimization Framework with Predictable Timing Quality *(DAC'17)*. 13–18.

[4] Rickard Ewetz and Cheng-Kok Koh. 2018. Scalable Construction of Clock Trees with Useful Skew and High Timing Quality. *TCAD* (2018).

[5] John P. Fishburn. 1990. Clock Skew Optimization. *IEEE Trans. Comput.* 39, 7 (1990), 945–951.

[6] Matthew R. Guthaus, Dennis Sylvester, and Richard B. Brown. 2006. Clock Buffer and Wire Sizing Using Sequential Programming *(DAC'06)*. 1041–1046.

[7] S. Held et al. 2003. Clock scheduling and clocktree construction for high performance ASICs *(ICCAD'03)*. 232–239.

[8] Dong-Jin Lee and Igor L. Markov. 2011. Multilevel tree fusion for robust clock networks *(ICCAD'2011)*. 632–639.

[9] J. Lillis, Chung-Kuan Cheng, and T. T. Y. Lin. 1996. Optimal wire sizing and buffer insertion for low power and a generalized delay model. *IEEE Journal of Solid-State Circuits* 31, 3 (1996).

[10] Logan Rakai et al. 2013. Buffer Sizing for Clock Networks Using Robust Geometric Programming Considering Variations in Buffer Sizes *(ISPD'13)*. 154–161.

[11] Venky Ramachandran. 2012. Construction of Minimal Functional Skew Clock Trees *(ISPD'12)*. 119–120.

[12] Subhendu Roy et al. 2015. Clock Tree Resynthesis for Multi-Corner Multi-Mode Timing Closure. *TCAD* (2015), 589–602.

[13] Cliff N. Sze. 2010. ISPD 2010 High Performance Clock Network Synthesis Contest: Benchmark Suite and Results *(ISPD'10)*. 143–143.

[14] King Ho Tam et al. 2008. Dual-V_{dd} Buffer Insertion for Power Reduction. *TCAD* 27 (2008), 1498–1502.

[15] Jeng-Liang Tsai, Tsung-Hao Chen, and C. C. P. Chen. 2004. Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing. *TCAD* 23, 4 (2004), 565–572.

[16] Necati Uysal and Rickard Ewetz. 2018. OCV Guided Clock Tree Topology Reconstruction. In *ASP-DAC'18*. 1–6.

[17] L. P. P. P. van Ginneken. 1990. Buffer placement in distributed RC-tree network for minimal Elmore delay. In *Proc. IEEE Int. Symp. Circuits Syst.* 865–868.

[18] Kai Wang and Malgorzata Marek-Sadowska. 2004. Buffer Sizing for Clock Power Minimization Subject to General Skew Constraints *(DAC'04)*. 159–164.