OCV-Aware Top-Level Clock Tree Optimization

Tuck-Boon Chan[‡], Kwangsoo Han[‡], Andrew B. Kahng[†‡],
Jae-Gon Lee[§] and Siddhartha Nath[†]
[†]CSE and [‡]ECE Departments, UC San Diego, [§]Samsung Electronics Co., Ltd.
tbchan@ucsd.edu, kwhan@eng.ucsd.edu, abk@cs.ucsd.edu,
altair.lee@samsung.com, sinath@cs.ucsd.edu

ABSTRACT

The clock trees of high-performance synchronous circuits have many clock logic cells (e.g., clock gating cells, multiplexers and dividers) in order to achieve aggressive clock gating and required performance across a wide range of operating modes and conditions. As a result, clock tree structures have become very complex and difficult to optimize with automatic clock tree synthesis (CTS) tools. In advanced process nodes, CTS becomes even more challenging due to on-chip variation (OCV) effects. In this paper, we present a new CTS methodology that optimizes clock logic cell placements and buffer insertions in the top level of a clock tree. We formulate the top-level clock tree optimization problem as a linear program that minimizes a weighted sum of timing slacks, clock uncertainty and wirelength. Experimental results in a commercial 28nm FDSOI technology show that our method can improve post-CTS worst negative slack across all modes/corners by up to 320ps compared to a leading commercial provider's CTS flow.

1. INTRODUCTION

In a modern SOC, *clock logic cells* (CLCs), such as clock gating cells (*CGC*s), multiplexers (*MUX*es) and dividers (*DIV*s), are required in the clock tree to achieve different performance and power saving requirements. To enable multi-mode operation and dynamic voltage frequency scaling (DVFS), large numbers of clocks are generated to drive flip-flops (FFs) in an SOC.[1] Balancing the clock trees of multiple clocks is challenging because timing constraints depend on clock periods, and on the process, voltage and temperature (PVT) corners. Furthermore, as on-chip variation (OCV) increases, *clock uncertainties* (derates) on the launch and capture paths can increase. Clock tree synthesis (CTS) must find optimal branching points in the clock tree to minimize clock uncertainties due to OCV on *non-common paths* [9][16][17]. Figure 1 (left) illustrates the clock balancing problem due to CLCs in a clock tree and the impact due to OCV. Due to the CLCs, the clock arrival times at FF groups are skewed. Moreover, the clock tree splits near the clock source; this leads to long non-common paths between the FF groups. As shown in Figure 1 (right), we can insert buffers to balance the clock, and optimize placement of the CLCs to reduce the non-common paths.

1.1 Motivation for Clock Tree Optimization

Given a clock tree, we represent the *top-level clock tree* as a hypergraph, $G_{top}(V_{top}, E_{top})$, in which V_{top} is a set of CLCs and the transitive fanin cells of the CLCs. E_{top} is a set of nets that connect the cells in V_{top}. Figure 2 shows a top-level clock tree

[1]Both synchronous and asynchronous clocks can exist in an SOC. Our work focuses on balancing synchronous clocks in an SOC.

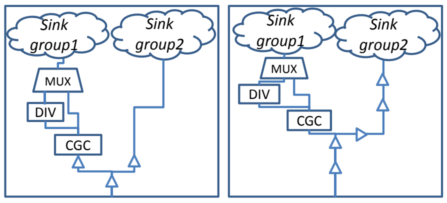


Figure 1: Clock tree synthesis problems.

with a CLC and three bottom-level buffered clock trees. In most cases, sophisticated EDA tools and CTS algorithms are able to achieve good solutions for the bottom-level clock trees. However, *achieving a good solution for the top-level clock tree can be problematic* when there are critical paths across the FF groups between different bottom-level clock trees. The requirements to balance the top-level clock are not obvious due to the complex structure of the tree (see Figure 6). Fixing the critical paths across the FF groups can be difficult at the bottom-level clock trees due to tight timing constraints among FFs within the same group. To optimize timing across FF groups, we propose to **balance the top-level clock tree while preserving the bottom-level clock trees**. For example, in Figure 2, if we increase the delay $d(1,2)$ on the net between *pins 1* and *2* from 2ns to 4ns, we can change the skew between *FF groups 1* and *2* from 2ns to 0ns, thereby meeting the timing target of *critical path A* which has a clock period of 3ns. Note that varying the delay on the top-level clock tree does not affect *critical path B* (but, the OCV derating on a longer top-level path will be larger), which has both its launch and capture FFs in the same group. Therefore, we only need to consider the requirements to balance clock across FF groups, thereby simplifying the top-level clock tree optimization problem. Since problems arise in the top-level tree due to CLCs, our work focuses on optimizing the placement of CLCs and insertion of buffers in the *top-level clock tree*.

1.2 Previous Work

Rajaram and Pan [16] propose CTS algorithms to optimize the chip-level clock tree across different PVT corners. They use quadratic programming to reallocate clock pins of IP blocks to reduce non-common paths in the chip-level clock tree. After clock pins are reallocated, buffers are inserted up to each pin, and subtrees are merged recursively in the same manner as the *deferred-merge embedding* (DME) algorithm [6]. The algorithm only inserts buffers that minimize the difference in clock latency among subtrees across PVT corners. Although the chip-level CTS work in [16] accounts for delay variation across PVT corners and timing penalty on non-common paths, it does not consider CLCs, timing between FF groups, or wirelength, all of which make CTS a challenging task. As illustrated in Figure 1, the placement of CLCs should also be considered during CTS as it can significantly affect the non-common paths in the tree. Other works [20][18] seek to minimize the effect of OCV during CTS, but do not address the issues of CTS with CLCs across multi-corner and multi-mode (MCMM) scenarios. Lung et al. [12] propose a linear programming (LP) based clock skew optimization [8] which accounts for delay variation across PVT corners. They also present a method to map the required delays obtained from the LP to actual circuits. While mapping delays, they use updated timing information to dynamically adjust buffer delays. Although this work addresses the MCMM clock skew minimization problem, it does not consider the effects of non-common paths and CLC placement. There are many previous works on buffer insertion for CTS (e.g., [1][4]), but they do not consider clock trees with CLCs which have different timing requirements depending on the operating modes and FF groups. Papa et al. [15] minimize worst negative slack (WNS) at a single PVT corner by optimizing the placement and buffering of datapaths. They do not consider multiple PVT corners and they do not balance the top-level clock trees.

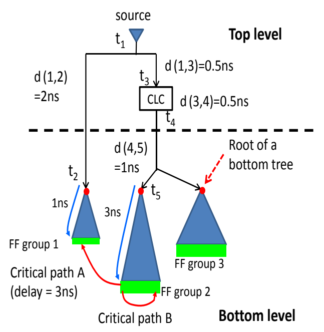


Figure 2: Example of balancing a clock tree by varying $d(i,j)$.

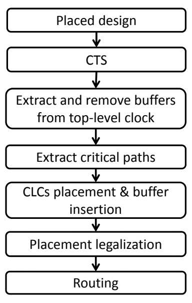

Placed design
CTS
Extract and remove buffers from top-level clock
Extract critical paths
CLCs placement & buffer insertion
Placement legalization
Routing

Figure 3: Overview of our CTS flow.

1.3 Our Work

To address the top-level CTS problem mentioned above, we propose a *new* CTS flow that accounts for the effects of CLCs as well as delay variations due to MCMM and OCV. The basic idea of our approach is to *automatically* identify the requirements to balance clocks based on the timing critical paths and use them to drive the CTS. The flow shown in Figure 3 starts with a placed design and performs *conventional CTS* to obtain a clock tree. We then extract the top-level clock tree (see Algorithm 1) and remove buffers in the top-level clock tree. Within the remaining (bottom-level) clock trees, we extract timing-critical FF-to-FF paths to identify the timing requirements for clock balancing. Based on these requirements, we construct a linear program (LP) to optimize the placement of CLCs and the delay on nets (achieved by inserting buffers) in the top-level clock tree. Unlike the routing algorithm proposed by Oh et al. [13] which minimizes the total wirelength of a routing tree, we include CLCs and Steiner point locations as variables in the LP, so that the LP-based optimization can account for the cost of non-common paths. With the physical locations of CLCs and Steiner points of the routes, we insert buffers in the top-level clock tree, legalize the placement and route the clock tree. The advantages of our methodology are as follows.

- Preserving the bottom-level clock trees affords more accurate timing information for the top-level clock tree optimization.[2]
- Since the top-level clock tree has many fewer instances, we can perform runtime-intensive optimizations which cannot be practically applied to the bottom-level clock tree.
- Introducing our new top-level clock tree placement optimization enables fixing of suboptimal CLC placements which have already been determined during the preceding placement stage.
- Buffer insertion and CLC placement optimization can achieve reductions of non-common path timing penalties, which are not achievable using local/incremental optimizations.

The key contributions of our work are summarized as follows.

- We propose a new automated clock tree synthesis methodology that optimizes the CLC placements and buffer insertion in the top-level clock tree.
- We propose an LP-based clock tree optimization method which accounts for routing resources (i.e., wirelength), circuit timing and the impact of non-common paths.

[2]In this work, we optimize only the top-level clock tree. Joint optimization of the top- and bottom-level trees is a direction of ongoing work.

- Our method improves WNS by up to 320ps, and reduce the top-level clock wirelength by up to 50% compared to a default CTS flow.
- As part of our validation process, we develop generators for testcases that represent clock tree structures typically found in high-speed IPs (e.g., graphics accelerators) and real-world SOCs.

In the remainder of this paper, Section II describes our top-level clock tree optimization methodology. Section III describes experimental setup and our experimental results. In Section IV, we summarize our work and outline directions for future research.

2. CLOCK TREE OPTIMIZATION

We now explain the top-level clock tree optimization problem and our approach. In the following, we use *condition*, k, to denote that a timing value is specific to a PVT corner, clock group and timing analysis type (setup or hold). For example, with two PVT corners, two operating modes, two clock groups and two timing analysis types, k will range from $1,2,\ldots,16$.

2.1 Problem Statement

Formally, the top-level CTS problem is defined as follows.
Objective: Minimize the weighted sum of (i) worst negative slack, (ii) total negative slack (TNS), (iii) clock uncertainty and (iv) wirelength of a clock tree [16].
Input: Placed design; list of CLCs; timing constraints (.sdc).
Output: An optimized placement of CLCs and clock buffers, clock routing of the top-level clock tree.

We model the cost of clock uncertainty $Z_k(a,b)$ on a critical path between FFs a and b as the sum of delays of the non-common launch and capture clock paths in the critical path. The non-common path delays are normalized to the clock period (CP) of the path using factor α_k.

$$Z_k(a,b) = \alpha_k\{\sum_{i\in h_a, j\in h_b} d_k(i,j) - \sum_{i,j\in(h_a\cap h_b)} 2d_k(i,j)\} \qquad (1)$$
$$\alpha_k = 1/\text{CP at condition } k$$

where h_a denotes a launch/capture path from a clock source to FF a, and $d_k(i,j)$ is the delay between pin i and j.

2.2 Our Approach

We formulate the top-level clock tree balancing problem as a linear program by assuming that we can vary (i) the delay $d_{ref}(i,j)$ from an output pin i to its fanout input pin j at a *reference condition*;[3] (ii) locations of CLCs; and (iii) Steiner points in the clock net (for a given topology). Although wire delay is normally nonlinear with respect to wirelength, we approximate $d_{ref}(i,j)$ as a linear function of distance between pin i and j assuming buffer insertion (as noted in, e.g., [15], the delay of a net with uniformly spaced buffers is linearly proportional to the number of stages).[4]

The main objective of the LP is to minimize the weighted sum of worst negative slack S_{wns}, the total negative slack S_{tns}, non-common paths, $Z_k(a,b)$, and total wirelength $U(i,j)$.[5] Note that we weight the $Z_k(a,b)$ proportional to its original *negative* slack (i.e., $1-s_k^0(a,b)$) such that the LP focuses on reducing the non-common path delay on timing paths. The critical paths and their original slacks $s_k^0(a,b)$ are extracted after the buffer removal step in Figure 3 by performing static timing analysis (STA).

To represent negative slack $s_k{}'(a,b)$ in the LP, we use Constraints (3) and (4) such that $s_k{}'(a,b)=0$ when $s_k(a,b)>0$. S_{wns} and S_{tns} are defined in Constraints (5) and (6), respectively. Since circuit designers may treat hold and setup slacks differently, we use a

[3]The reference condition is $\{$SS process corner, $0.85V, 125°C\}$.

[4]A buffered net has relatively linear delay vs. distance even in advanced technology nodes. For example, the stage delay in a uniformly buffered-chain is almost the same except for the first few stages. Adding an additional stage will increase the delay by a fixed amount. To account for the non-linearity within a single stage delay, our buffer insertion algorithm detour wires to match the required delay obtained from our LP.

[5]Our objective function is different from [16]. They do not consider wirelength and the timing between FF groups.

weight $\gamma_k \geq 0$ to set the ratio of importance (i.e., normalization ratio) of setup and hold slacks. The value of γ_k can be different for hold or setup analysis, as indicated by the condition k. We represent the timing slacks $s_k(a,b)$ for each timing-critical path between FFs a and b as a function of the original slack, original clock skew $\lambda_k(a,b)$, and the clock arrival times $(t_{ref}(a))$ in Constraint (7). Because delay and slack vary according to PVT corners and timing analysis type, we normalize the slacks across different conditions to a reference corner by using scaling factors η_k, following the approach in [12]. $\zeta = 1$ if the path is a setup-critical path and $\zeta = -1$ if the path is a hold-critical path. $t_{ref}(a)$ is the sum of delays along the path h_a (Constraint (8)).

Objective:

$$\text{Min } -w_{wns}\cdot S_{wns} - w_{tns}\cdot S_{tns} + w_{wl}\cdot \sum_{e(i,j)\in E_{top}} U(i,j)$$
$$+w_{ncp}\cdot \sum_{k,a,b}(1-s_k^0(a,b))\cdot Z_k(a,b) + w_{dis}\cdot \sum_i M(i,i_0) \tag{2}$$

Subject to:

$$s_k{}'(a,b) \leq \alpha_k \cdot s_k(a,b),\ \forall a,b,k \tag{3}$$
$$s_k{}'(a,b) \leq 0, \qquad \forall a,b,k \tag{4}$$
$$S_{wns} \leq \gamma_k \cdot s_k{}'(a,b), \forall a,b,k \tag{5}$$
$$S_{tns} = \textstyle\sum_{a,b,k}\gamma_k \cdot s_k{}'(a,b) \tag{6}$$
$$\eta_k \cdot s_k(a,b) = \eta_k \cdot (s_k^o(a,b) - \lambda_k(a,b)) + \zeta(t_{ref}(a) - t_{ref}(b)) \tag{7}$$
$$t_{ref}(a) = \sum_{i,j\in h_a} d_{ref}(i,j) \tag{8}$$
$$d_{ref}(i,j) \geq \beta_{ref}\cdot U(i,j) \tag{9}$$
$$Z_k(a,b) = \alpha_k\{\sum_{i\in h_a, j\in h_b} d_k(i,j) - \sum_{i,j\in(h_a\cap h_b)} 2d_k(i,j)\} \tag{10}$$
$$M(i,j) = m_x(i,j) + m_y(i,j) \tag{11}$$
$$m_x(i,j) \geq (p_x(j) - p_x(i)), m_x(i,j) \geq 0 \tag{12}$$
$$m_y(i,j) \geq (p_y(j) - p_y(i)), m_y(i,j) \geq 0 \tag{13}$$
$$M(i,i_0) = m_x(i,i_0) + m_y(i,i_0) \tag{14}$$
$$m_x(i,i_0) \geq (p_x(i) - p_x(i_0)), m_x(i,i_0) \geq 0 \tag{15}$$
$$m_y(i,i_0) \geq (p_y(i) - p_y(i_0)), m_y(i,i_0) \geq 0 \tag{16}$$
$$0 \leq p_{\{x,y\}}(i) \leq F_{\{x,y\}} \tag{17}$$

The values of $\lambda_k(a,b)$ and the cell delays in $d_{ref}(i,j)$ are constants in the LP, and are extracted from STA reports after the buffer removal step in our flow. In Constraint (9), we model the delay $d_{ref}(i,j)$ between pins i and j as a linear function of the Manhattan distance $U(i,j)$ between the pins. β_{ref} is a conversion factor to convert the Manhattan distance to delay at the reference condition. We obtain the value of β_{ref} using the optimal repeater length method in [2]. The value of β_{ref} is 30ps per 100μm for a 8X buffer in the 28nm foundry FDSOI standard cell library that we use in our experiments. We calculate $Z_k(a,b)$ in Constraint (10). The Manhattan distances are calculated by using Constraints (11)–(13). The location of a pin i is specified by variables $p_x(i)$ and $p_y(i)$, which represent the x and y coordinates of the pin. The bounds for $p_x(i)$ and $p_y(i)$ are specified in Constraint (17). F_x and F_y are the upper bounds for the pin coordinates along the x and y axes, i.e., the dimensions of the design's floorplan.

To avoid unnecessary cell displacements, we add a *displacement cost* $M(i,i_0)$ in the objective function [15]. The displacement cost is defined as the sum of Manhattan distances between the original cell locations ($[p_x(i_0), p_y(i_0)]$) and their corresponding cell locations ($[p_x(i), p_y(i)]$) after optimization. $M(i,i_0)$ is calculated using Constraints (14)–(16). Since the displacement cost will force the LP to "pull" the cells to their original locations, we use a very small weighting factor ($w_{dis} = 0.001$) as the cell displacement cost.

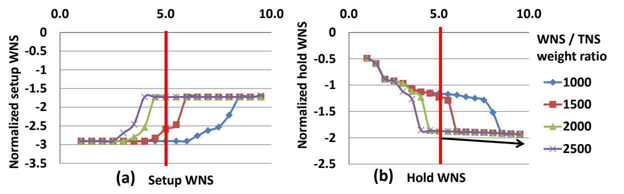


Figure 4: Normalized (a) setup WNS and (b) hold WNS obtained by solving the LP for different γ_k and w_{wns}/w_{tns}.

We apply uniform weights for TNS and non-common path delays, i.e., $w_{tns} = 1$, $w_{ncp} = 1$. Since the typical values of total wirelength in a top-level clock tree is much larger than the timing slacks we set $w_{wl} = 0.001$ such that the cost in the LP is not dominated by the wirelength.

Figures 4(a) and 4(b) respectively show the setup and hold WNS (both normalized to their corresponding clock periods) obtained by solving the LP for different values of γ_k. As we sweep γ_k from 1 to 10, the setup WNS obtained from the LP improves but the hold WNS worsens. When we sweep the w_{wns}/w_{tns} ratio, the setup and hold WNS are not affected when $\gamma_k \leq 3$. However, when $\gamma_k > 3$, the cost in the LP is dominated by the setup WNS and increasing the w_{wns}/w_{tns} ratio will improve the setup WNS. Since the hold time violations are relatively easy to fix by inserting buffers, we prioritize setup slacks when we select the γ_k and w_{wns}/w_{tns} weight ratios. In our experiments, we use $\gamma_k = 5$ and $w_{wns}/w_{tns} = 2000$ because we experimentally observe that by increasing γ_k further does not improve the setup WNS but makes hold WNS worse (black arrow in Figure 4(b)). We use the same values of the weighting factors across all testcases. It is also possible to apply different combinations of values of weighting factors, run the flows in parallel, and choose the best CTS solution.

2.3 Implementation Heuristics

Given a design with an initial clock tree, $G(V,E)$, and a subset of vertices $V_{CLC} \subseteq V$ corresponding to CLCs, we extract the top-level clock net using Algorithm 1.[6] First, we create a list V_{top} of all transitive fanin cells of the CLCs. In Lines 2–4, we remove all the clock routes connected to the fanin cells. In Lines 5–12, we check each cell in V_{top}, remove all the buffers and reconnect the nets accordingly.

Algorithm 1 Extract top-level clock tree

```
Procedure Extract_top()
Input : $G(V,E)$, $V_{CLC}$
Output: $G(V_{top}, E_{top})$
1:  $V_{top} \leftarrow$ transitive fanins of all $v \in V_{CLC}$;
2:  for all $e(u,v) \in E$; $u,v \in V_{top}$ do
3:      Remove clock routing for $e(u,v)$;
4:  end for
5:  $E_{top} \leftarrow \emptyset$
6:  for $v \in V_{top}$ do
7:      if $v$ is a buffer then
8:          $(v.parent).children \leftarrow v.children$;
9:          $V_{top} \leftarrow V_{top} \setminus \{v\}$;
10:         $E_{top} \leftarrow E_{top} \cup \{e(v.parent, v.children)\}$;
11:     end if
12: end for
13: Return $G(V_{top}, E_{top})$;
```

In the top-level clock balancing problem, the LP optimizes the delays from an output pin to input pins in every net. For nets with more than one fanout, we modify the net into a binary tree by inserting Steiner points. The purpose of this step is to include the locations of the Steiner points as variables in the LP so as to optimize the non-common paths. Given a net, $G_{net}(V,E)$, and its driving pin, v_r, we apply Algorithm 2 to obtain a binary tree. In Lines 8–16, we find the pin pair that minimize the metric $\Delta L'$ which is defined as the sum of the difference in *sink latency*[7] and the delay

[6] We obtain V_{CLC} by assuming all CLCs are in the top-level clock tree.

[7] The sink latency $L(u)$ of a pin u is the maximum latency from u to any FF in the transitive fanout of u.

due to the Manhattan distance between these pins.[8] In Lines 17–25, we merge the pin pair that has minimum $\Delta L'$ by creating a new Steiner point. We define the x and y coordinates of the new Steiner point as the average of the x and y coordinates of the merged pins (Lines 21–22). The sink latency of the Steiner point is defined as the maximum sink latency of the merged pins (Line 20). The procedure *split_net()* is invoked repeatedly until all driving pins have a single connection (to a Steiner point). Figure 5 illustrates our Steiner point insertion algorithm. In the first iteration, we merge pins j_2 and j_3 because they have the smallest ΔL and Manhattan distance. Pins j_2 and j_3 are then connected to Steiner point $j_{2'}$ (red square). The location of $j_{2'}$ is defined by the average of the x and y coordinates of pins j_2 and j_3. In the second iteration, we merge pins j_1 with $j_{2'}$ because they have a smaller $\Delta L'$ even though the Manhattan distance between pins j_1 and $j_{2'}$ is larger than the Manhattan distance between pins j_4 and $j_{2'}$. In the last iteration, we merge j_4 and $j_{1'}$. Note that our algorithm selects the pins to merge based on the sum of Manhattan distance and the difference in sink latency. This is different from the algorithm in [7] which selects the pins based on Manhattan distance only. For example, the algorithm in [7] will merge j_2 and j_3, followed by j_4 and j_1. As shown in Figure 5 (the upper-right clock tree), the algorithm in [7] will lead to a clock tree that will require more buffers to be inserted (red arrows) to balance the clock latencies (green arrows) compared to the tree produced by our algorithm (the lower-right clock tree).

Algorithm 2 Create Steiner points

```
Procedure split_net()
Input : $G_{net}(V,E), v_r \in V$
Output: $G'_{net}(V',E')$
1: $V' \leftarrow V$;
2: if ($|v_r.child| < 2$) then
3:     $E' \leftarrow E$;
4: else
5:     $E' \leftarrow \emptyset$;
6:     while ($|v_r.child| \geq 2$) do
7:         $min\_\Delta L' \leftarrow \infty$;
8:         for ($u_1, u_2 \in v_r.child$) do
9:             $\Delta L(u_1,u_2) \leftarrow |u_1.L - u_2.L|$;
10:            $\Delta L'(u_1,u_2) \leftarrow \beta_k \cdot M(u_1,u_2) + \Delta L(u_1,u_2)$;
11:            if ($\Delta L'(u_1,u_2) \leq min\_\Delta L'$) then
12:                $u_{min1} \leftarrow u_1$;
13:                $u_{min2} \leftarrow u_2$;
14:                $min\_\Delta L' \leftarrow \Delta L'(u_1,u_2)$;
15:            end if
16:        end for
17:        Create a new Steiner point $u' \notin V$;
18:        $v_r.child \leftarrow v_r.child \setminus \{u_{min1}, u_{min2}\}$;
19:        $u'.child \leftarrow \{u_{min1}, u_{min2}\}$;
20:        $u'.L \leftarrow max(u_{min1}.L, u_{min2}.L)$;
21:        $p_x(u') \leftarrow (p_x(u_{min1}) + p_x(u_{min2}))/2$;
22:        $p_y(u') \leftarrow (p_y(u_{min1}) + p_y(u_{min2}))/2$;
23:        $v_r.child \leftarrow v_r.child \cup \{u'\}$;
24:        $V' \leftarrow V' \cup \{u'\}$;
25:        $E' \leftarrow E' \cup \{e(u',u_{min1}), e(u',u_{min2})\}$;
26:    end while
27:    $E' \leftarrow E' \cup \{e(v_r,u')\}$;
28: end if
29: Return $G'_{net}(V',E')$;
```

By solving the LP, we obtain cell locations, clock routes (Steiner point locations) and net delays in the top-level clock tree. Next, we insert buffers in the top-level clock tree to guide clock routing and control clock skews. For each two-pin net in the optimized top-level clock tree, we insert buffers according to the steps described in Algorithm 3. In Line 1, we initialize the variable n, which indicates the number of inserted buffers, to 1. In Lines 2–14, we calculate the number of buffers required to meet the delay target as a function of net delays and buffer delays. M_{buf} is the minimum required spacing between two buffers.[9] The **while** loop exits when the sum

[8]We convert the Manhattan distance to delay by a conversion factor β_k at the reference condition.

[9]We use $M_{buf} = 5\mu$m in our experiments.

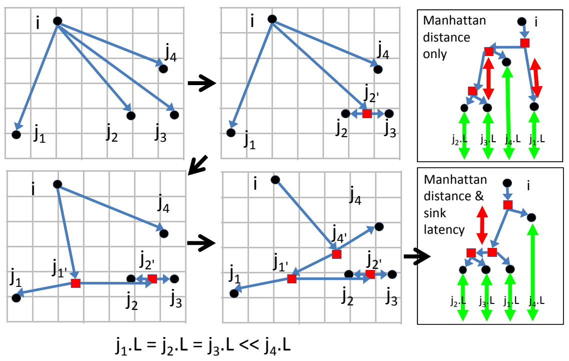


Figure 5: Steiner point creation. In each iteration, we find a pair of pins (black circles) or Steiner points with the minimum $\Delta L'$ (sum of scaled Manhattan distance and difference in sink latency) and connect them to a new Steiner point (red square).

of net and buffer delays (d_{est}) exceeds the required delay between the pins i and j (d_{req}). In Lines 15–21, we calculate the minimum wirelength required to insert n buffers. If this wirelength is less than or equal to the Manhattan distance between pins i and j, $M(i,j)$, we place the buffers in an L-shaped (y-axis first, followed by x-axis) manner. Otherwise, we place the buffers in a U-shaped manner because total wirelength is $> M(i,j)$. U-shaped placement is the general case, and L-shaped is a special case of U-shape when total wirelength is $\leq M(i,j)$.

Algorithm 3 Insert buffers

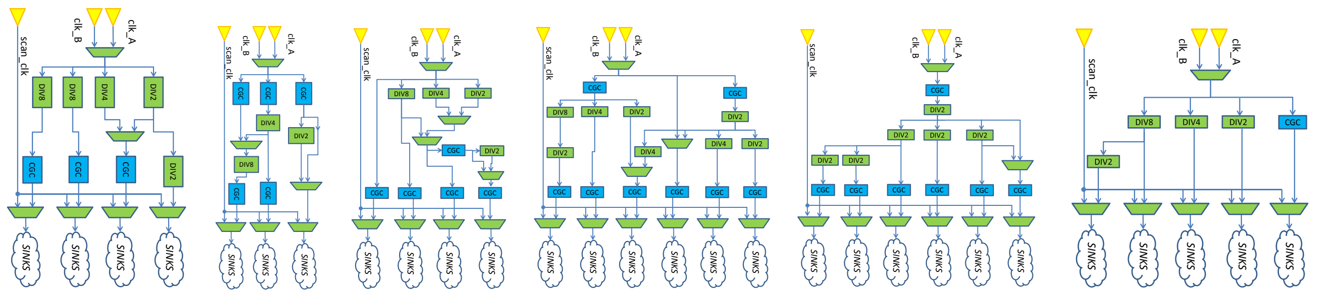
```
Procedure insert_buffers()
Input : pins $i$ and $j$, $d_{req}(i,j)$
Output: inserted buffers
1: $n \leftarrow 1$;
   // calculate number of buffers to meet required delay
2: while (1) do
3:     $l \leftarrow M(i,j)/(n+1)$;
4:     if ($l < M_{buf}$) then
5:         $l \leftarrow M_{buf}$;
6:     end if
7:     $d_{est} \leftarrow (n+1) \times d_w(l) + (n-1) \times d_g(c_{in\_buf} + c_w(l)) + d_g(c_{in}(j) + c_w(l))$;
8:     if ($d_{est} > d_{req}(i,j)$) then
9:         $n \leftarrow n-1$;
10:        break;
11:    else
12:        $n \leftarrow n+1$;
13:    end if
14: end while
15: if ($n > 0$) then
16:    if ($M_{buf} \times n > M(i,j)$) then
17:        Detour wire and place $n$ buffers in U-shape;
18:    else
19:        Place $n$ buffers in L-shape;
20:    end if
21: end if
```

3. EXPERIMENTS

To test the effectiveness of our methodology, we require testcases with complex top-level clock trees. Since existing benchmarks [10][23] typically lack complex top-level clock trees, we generate testcases based on common clock tree structures typically found in high-speed SOCs and IPs [21][22]. The clock structures of our testcases are shown in Figures 6(a)–(f). We use dual-Vt 28nm foundry FDSOI libraries and implement each testcase at two operating modes – {1.25GHz at 0.95V} and {1.667GHz at 1.20V}. We perform placement and routing (P&R) using a commercial tool and use *Synopsys PrimeTime vH-2013.06-SP2* [25] for timing analysis. Table 1 shows the timing analysis parameters in our experiments.

3.1 Testcase Description and Generation

Testcases from Tsay [19], Kahng and Tsao [10] and ISPD-2009/2010 [23] CNS contest benchmarks lack CLCs and are insufficient to create complex top-level clock hierarchies. Kahng et al. [11] improve CTS testcases by adding CLCs (Figures 3(a) and 3(b) in [11]) but two key elements ignored: (1) combinational



(a) T1 (b) T2 (c) T3 (d) T4 (e) T5 (f) T6

Figure 6: Clock structures of our testcases.

Table 1: Timing analysis setup.

Parameter	Value
PVT corner for setup analysis at the 1.250GHz mode	SS, 0.85V, 125C
PVT corner for hold analysis at the 1.250GHz mode	FF, 1.05V, 125C
PVT corner for setup analysis at the 1.667GHz mode	SS, 1.10V, 125C
PVT corner for hold analysis at the 1.667GHz mode	FF, 1.30V, 125C
Clock uncertainty	0.15 $\times$ clock period
Maximum transition for clock paths	0.055ns
Maximum transition for data paths	0.125 $\times$ clock period
Timing derate on net delay (early/late)	0.90 / 1.19
Timing derate on cell delay (early/late)	0.90 / 1.05
Timing derate on cell check (early/late)	1.10 / 1.10

logic between FF groups and hence critical paths between FF groups; and (2) multiple clock sources. The CTS problem becomes difficult when synchronous and asynchronous clocks need to be balanced across multiple FF groups. We improve over [11] by (1) adding combinational logic with varying number of stages between FF groups, (2) adding multiple synchronous and asynchronous clocks, (3) using CLCs at different hierarchies to make the clock balancing problem very complex, (4) creating multiple top-level clock hierarchies, and (5) performing CTS with MCMM and OCV constraints.

Figures 6(a)–(f) show the six testcases T1–T6 used in our experiments. These testcases use three clock sources typically seen in SOC designs [21] and can have large fanouts (e.g., >1000 FFs). The clock source *m_clk* is from the crystal oscillator, *clk* is the output of a PLL and *scan_clk* is the test clock. Clock sources *m_clk* and *clk* are used to implement low-power modes of operation, such as DVFS. The testcases use three kinds of dividers (*DIV2, DIV4, DIV8* in figures), a glitch-free clock MUX, and integrated clock gating cells (CGCs) as CLCs. Outputs of all dividers are sources of generated clocks; the generated clocks typically drive FFs for debug/tracing, IO and other peripheral logic.

To implement variable stages of combinational logic, we use *NetGen* [26] and vary #stages from 15 to 30. To model different critical paths, we connect FFs across groups as well as within the same group using these logic stages. To obtain floorplan dimensions that resemble SOCs, we use multiple instantiations of an interface logic module (ILM) of the *jpeg_encoder* design from OpenCores [24]. We create a netlist with the top module *x5_jpeg*, in which we instantiate the *jpeg_encoder* design five times, perform SP&R and generate an ILM. Note that in this paper, we do not optimize the bottom-level clock tree. Therefore, instantiation of the same *x5_jpeg* multiple times (instead of using different modules) does not change the outcome of our experiments. We connect multiple instances of the ILM using combinational logic stages. For all CLCs, we implement custom netlists in the 28nm foundry FDSOI technology, and group FFs within the CLCs into their own skew groups so that these FFs do not affect global skew and latencies. The path latencies of FF groups are controlled by changing timing constraints and the number of stages of combinational logic between the groups. To allow a blockage-free placement region for the CLCs, we place ILM blocks (hard macros for the CTS tools) in an L-shaped manner along the periphery of the core as shown in Figure 7(a).

All testcases contain bidirectional paths, i.e., both launch and capture FFs appear in FF groups that are driven by the fastest clock and other slower clocks. In addition, the fastest clock drives around 90% of the FFs that do not belong to the ILMs. Table 2 shows #CLCs, #cells, the FFs not in ILM, FFs in the ILM, FFs at the ILM boundary, and the area of each testcase (design in table). Testcases T2, T3 and T6 contain critical paths between FFs from two different clocks, one with large latency and the other with small latency. The CTS problem is complicated by the need to balance skew between these FF groups. Testcases T1–T4 contain multiple generated clocks and reconvergent paths between these clocks. These testcases make the CTS problem complex because skew needs to be balanced between fast and slow clocks. In testcases T3–T5, the control signals of CGCs are generated by *clk*, which makes the latency of the signal to the enable pin of the CGCs very critical. Besides balancing skews, CTS also needs to balance the critical path delays of the enable signal to the CGCs along with the clock latency. To report timing paths across clocks accurately, we set the path multiplier in the Synopsys Design Constraint (SDC) [3] file for paths between all clocks.

Table 2: Benchmark designs.

Design	#CLCs	#Cells	#Flip-flops			Area
			$\notin$ ILM	$\in$ ILM	Boundary	(mm^2)
T1	17	1.93M	10K	202.7K	1.7K	3.75×3.00
T2	12	1.93M	10K	202.7K	1.7K	3.75×3.00
T3	18	1.93M	12K	202.7K	1.7K	3.75×3.00
T4	24	1.93M	12K	202.7K	1.7K	3.75×3.00
T5	18	1.93M	8K	202.7K	1.7K	3.75×3.00
T6	13	1.92M	7K	202.7K	1.7K	3.75×3.00

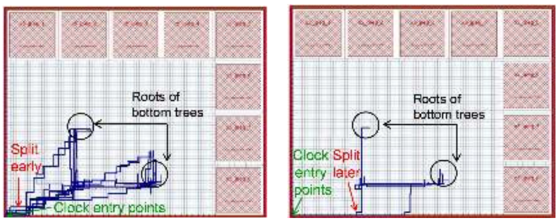
3.2 Experimental Results

Table 3 summarizes the key metrics of the clock tree before (**I** = Initial, produced by a commercial tool) and after (**O** = Optimized) applying our top-level clock tree optimization. Rows 1–14 in Table 3 show the results at the post-CTS stage, while Rows 15-28 show the results at the end of the implementation flow (after datapath routing).[10]

Post-CTS stage: Our optimization flow reduces the total wirelength of the top-level clock tree by 53% to 68% across all six testcases. Figure 7 shows that wirelength reduces because our flow clusters the CLCs such that the clock tree does not split near the clock entry points. The large wirelength reduction suggests that the initial CLC placements by EDA tools may not be aware of the CTS requirements. The smaller wirelength enables the optimized clock tree to also reduce the number of buffers. In testcases T4 and T5, the number of buffers is larger, as our optimization flow inserts more buffers in the clock tree to improve timing slack. To estimate switching power, we extract gate and wire capacitances of the top-level clock tree. Rows 5–6 in Table 3 show that our flow can reduce the switching power in the top-level clock tree by 12% to 40% for all testcases, including testcases T4 and T5, where the number of buffers increases.

Our flow also improves the setup WNS and TNS by up to 550ps and 255ns, respectively (Rows 7–10). Hold WNS and TNS are also improved except for testcase T6, in which the hold WNS and TNS worsen by 110ps and 780ps, respectively (Rows 11–14). Our optimization flow can worsen hold WNS and TNS because we focus on improving the setup slacks ($\gamma_k = 5$). The tradeoff between setup and hold slacks is based on the following assumptions: (1) hold time violations are easier to fix in post-CTS implementation

[10]We apply the default clock tree optimization, routing and design optimization commands in the EDA tool after CTS. We do not compare our work with previous work as their algorithms cannot be applied to our testcases.



(a) Initial clock tree. (b) Optimized clock tree.

Figure 7: Initial (a) and optimized (b) clock trees for testcase T6. Wiring of the top-level clock trees is shown in black. Our flow splits common paths farther from the clock root compared to the initial clock tree. As a result, the total wirelength in the top-level clock tree is reduced from 45mm to 22mm.

Table 3: Post-CTS results. I: Initial, O: Optimized.

Testcase:			T1	T2	T3	T4	T5	T6
Post-CTS								
1	Top-level	I (um)	18086	19261	41476	38830	34009	36052
2	wirelength	O (um)	8442	8614	13193	14389	14186	15104
3	Total-level	I	163	210	361	298	322	253
4	buffers	O	152	167	242	301	421	226
5	Switching	I (uW)	875	1018	1639	1515	1557	1315
6	power	O (uW)	590	692	969	1210	1360	987
7	Worst	I (ns)	-0.05	-0.10	-0.37	-0.65	-0.55	-0.32
8	setup WNS	O (ns)	-0.04	0.00	-0.36	-0.55	0.00	-0.20
9	Total	I (ns)	-0.41	-0.25	-48.47	-1034.38	-8.39	-40.56
10	setup TNS	O (ns)	-0.17	0.00	-45.47	-779.46	0.00	-12.78
11	Worst	I (ns)	0.00	0.00	-0.40	-0.04	0.00	-0.04
12	hold WNS	O (ns)	0.00	0.00	-0.40	-0.01	0.00	-0.15
13	Total	I (ns)	0.00	0.00	-130.12	-0.21	0.00	-0.09
14	hold TNS	O (ns)	0.00	0.00	-128.23	-0.05	0.00	-0.87
Post-datapath routing								
15	Top-level	I (um)	26261	30779	58223	50432	48761	44794
16	wirelength	O (um)	15750	19097	33982	27342	28570	22051
17	Total-level	I	163	215	357	300	322	252
18	buffers	O	152	170	248	306	427	226
19	Switching	I (uW)	885	1100	1748	1592	1616	1337
20	power	O (uW)	638	729	1042	1220	1374	968
21	Worst	I (ns)	-0.03	0.00	-0.05	-0.58	-0.32	-0.19
22	setup WNS	O (ns)	0.00	0.00	0.00	-0.46	0.00	-0.18
23	Total	I (ns)	-0.05	0.00	-0.06	-883.50	-3.03	-10.81
24	setup TNS	O (ns)	0.00	0.00	0.00	-609.28	0.00	-1.10
25	Worst	I (ns)	0.00	0.00	-0.37	-0.04	0.00	0.00
26	hold WNS	O (ns)	0.00	0.00	-0.10	-0.11	-0.04	-0.05
27	Total	I (ns)	0.00	0.00	-19.82	-0.14	0.00	0.00
28	hold TNS	O (ns)	0.00	0.00	-5.46	-0.78	-0.08	-0.33
29	Total timing paths in LP		16K	20K	72K	40K	28K	11K
Runtime (minutes)								
30	Extract timing		45	37	176	71	71	25
31	Formulate LP		36	26	165	51	36	9
32	Place & legalization		8	4	6	5	6	5
33	Clock routing		7	4	5	4	5	5
34	Total		96	71	352	131	118	44

stages, and (2) some of the hold time violations are fixed by the increased wire delays in the routing stage.

In Rows 30–34 of Table 3, we report runtimes of the main procedures in our optimization flow. We spend most of the time to extract timing information and to formulate the LP.[11] CLC placement, buffer insertion, legalization and routing only take 10 minutes in total because there are not many cells in the top-level clock tree. The total runtime is 135 minutes on average. Testcase T3 has a higher runtime because it has more timing-critical paths than other testcases (Row 29).

Post-datapath routing stage: To study the benefits of our optimization flow, we also compare the post-routing results between the initial and the optimized clock trees. The results in Table 3 show that all designs with the optimized clock tree have the same or improved setup WNS compared to the designs with the initial clock tree (Rows 21–24). The improvement in setup WNS at the post-routing stage is up to 320ps. Although some testcases with the optimized clock tree have worse hold slacks (i.e., testcases T4, T5 and T6), the differences are less than 100ps. The results in Rows 15–16 shows that our optimization flow reduces the total wirelength by 38% to 51% across all six testcases. The improvements are smaller as compared to the post-CTS stage because the total wirelength of the initial and optimized clock trees both increase at the post-routing stage due to wiring of the signal nets. Total number of buffers and switching power at the post-routing stage are similar to values seen at the post-CTS stage.

[11] Solving the LP takes less than 30 seconds.

4. CONCLUSIONS

Designing a balanced top-level clock tree with multiple clock sources is very complex as we need to consider MCMM, OCV and timing constraints across FF groups. We develop a CTS methodology that optimizes CLC placement and buffer insertion, and that minimizes non-common paths between FF groups. We formulate the top-level CTS problem as the minimization of a weighted sum of WNS, TNS, clock uncertainty due to OCV and wirelength. We solve this problem using LP and develop heuristic flows to insert Steiner points and buffers, which are required elements of a top-level CTS solution. We also develop generators for testcases that resemble clock tree structures typically found in high-speed SOCs. We validate our optimization flow on testcases from our generators and achieve up to 51% reduction in wirelength for the top-level clock tree, and 320ps improvement in WNS, compared to a leading commercial CTS tool. Our future work includes (i) handling obstacles, (ii) accounting for *optimal* buffering solutions, (iii) creating testcases to capture other important SOC elements such as memory controller and multimedia blocks, and (iv) joint optimization of the top- and bottom-level clock trees.

5. REFERENCES

[1] C. J. Alpert, M. Hrkic, J. Hu, A. B. Kahng, J. Lillis, B. Liu, S. T. Quay, S. S. Sapatnekar, A. J. Sullivan and P. Villarrubia, "Buffered Steiner Trees for Difficult Instances", *Proc. ISPD*, 2001, pp. 4-9.

[2] H. B. Bakoglu, *Circuits, Interconnects, and Packaging for VLSI.* Reading, MA: Addison-Wesley, 1990.

[3] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, Springer, 2009.

[4] Y.-Y. Chen, C. Dong and D. Chen, "Clock Tree Synthesis Under Aggressive Buffer Insertion", *Proc. DAC*, 2010, pp. 86-89.

[5] C. Chen, C. Kang and M. Sarrafzadeh, "Activity-Sensitive Clock Tree Construction for Low Power", *Proc. ISLPED*, 2002, pp. 279-282.

[6] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese and A. B. Kahng, "Zero Skew Clock Routing with Minimum Wirelength", *IEEE Trans. on Circuits and Systems* 39(11) (1992), pp. 799-814.

[7] M. Edahiro, "A Clustering-Based Optimization Algorithm in Zero-Skew Routings", *Proc. DAC*, 1993, pp. 612-616.

[8] J. P. Fishburn, "Clock Skew Optimization", *IEEE Trans. on Computers* 39(7) (1990), pp. 945-951.

[9] E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits", *IEEE Proceedings*, 89(5) (2001), pp. 665-692.

[10] A. B. Kahng and C.-W. A. Tsao, "VLSI CAD Software Bookshelf: Bounded-Skew Clock Tree Routing", Version 1.0, 2000. *http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/*

[11] A. B. Kahng, B. Lin and S. Nath, "High-Dimensional Metamodeling for Prediction of Clock Tree Synthesis Outcomes", *Proc. SLIP*, 2013.

[12] C.-L. Lung, H.-C. Hsiao, Z.-Y. Zeng and S.-C. Chang, "LP-Based Multi-Mode Multi-Corner Clock Skew Optimization", *Proc. VLSI-DAT*, 2010, pp. 335-338.

[13] J. Oh, I. Pyo and M. Pedram, "Constructing Lower and Upper Bounded Delay Routing Trees Using Linear Programming", *Proc. DAC*, 1996, pp. 401-404.

[14] U. Padmanabhan, J. M. Wang and J. Hu, "Robust Clock Tree Routing in the Presence of Process Variations", *IEEE Trans. on CAD* 27(8) (2008), pp. 1385-1397.

[15] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. J. Alpert and I. L. Markov, "RUMBLE: An Incremental Timing-Driven Physical-Synthesis Optimization Algorithm", *IEEE Trans. on CAD* 27(12) (2008), pp. 2156-2168.

[16] A. Rajaram and D. Z. Pan, "Robust Chip-Level Clock Tree Synthesis", *IEEE Trans. on CAD* 30(6) (2011), pp. 877-890.

[17] V. Ramachandran, "Construction of Minimal Functional Skew Clock Trees", *Proc. ISPD*, 2012, pp. 119-120.

[18] J.-L. Tsai, "Clock Tree Synthesis for Timing Convergence and Timing Yield Improvement in Nanometer Technologies", *Ph.D. Thesis*, Electrical and Computer Engineering, University of Wisconsin-Madison, 2005.

[19] R.-S. Tsay, "Exact Zero-Skew", *Proc. ICCAD*, 1991, pp. 336-339.

[20] D. Velenis, M. C. Papaefthymiou and E. G. Friedman, "Reduced Delay Uncertainty in High Performance Clock Distribution Networks", *Proc. DATE*, 2003, pp. 68-73.

[21] Broadcom Corporation (networking infrastructure physical design principal engineer), *personal communication*, November 2013.

[22] Samsung Electronics Corporation (System LSI application processor principal engineer), *personal communication*, November 2013.

[23] *ISPD CNS Contest.* http://ispd.cc/contests/09/ispd09cts.html

[24] *OpenCores.* http://opencores.org

[25] *Synopsys PrimeTime User's Manual.* http://www.synopsys.com/Tools/Implementation/Signoff/PrimeTime/Pages/

[26] *UC Benchmark Suite for Gate Sizing.* http://vlsicad.ucsd.edu/SIZING/bench/artificial.html