Robust Chip-Level Clock Tree Synthesis

Anand Rajaram, *Member, IEEE,* and David Z. Pan, *Senior Member, IEEE*

***Abstract*—Chip-level clock tree synthesis (CCTS) is a key problem that arises in complex system-on-a-chip designs. A key requirement of CCTS is to balance the clock-trees belonging to different IPs such that the entire tree has a small skew across all process corners. Achieving this is difficult because the clock trees in different IPs might be vastly different in terms of their clock structures and cell/interconnect delays. The chip-level clock tree is expected to compensate for these differences and achieve good skews across all corners. Also, CCTS is expected to reduce clock divergence between IPs that have critical timing paths between them. Reducing clock divergence reduces the maximum possible clock skew in the critical paths between the IPs and thus improves yield. This paper proposes effective CCTS algorithms to simultaneously reduce multicorner skew and clock divergence. Experimental results on several test-cases indicate that our methods achieve 30% reduction in the clock divergence with significantly improved multicorner skew variance, at the cost of 2% increase in buffer area and 1% increase in wirelength.**

***Index Terms*—Chip-level clock tree synthesis (CCTS), multi-corner CTS, robust clock tree synthesis.**

I. Introduction

A SYSTEM-ON-A-CHIP (SoC) design can be defined as "an IC, designed by stitching together multiple stand-alone VLSI designs to provide full functionality for an application" [1]. SoC designs have become increasingly common and the trend is expected to continue in the future [2]. An attractive feature of SoC designs is the ability to reuse a given sub-component in multiple chips. The level of reuse can be different from IP to IP. This paper uses the word *IP* to denote the individual sub-blocks used in SoC designs. They are also referred to as *core* in some literature [1]. At one extreme of the reuse spectrum are hard-IPs where the exact transistor-level layout is reused in several designs. At the other end are the soft-IPs which go through the physical design/timing closure process from scratch so as to integrate the IP with the rest of the chip. This paper defines a soft-IP as the one for which netlist is available but physical information is not present as a part of the IP.

Most SoC physical design closure is done in a hierarchical fashion [1]. In such a methodology, different IPs should be integrated along with the glue logic to complete the chip-level timing closure. Timing closure in most practical chips involve verifying timing across several corners (referred to as design corners) that represent several global variation effects such as fab-to-fab, wafer-to-wafer, die-to-die variation, global voltage and temperature variations [1], [3], [4]. This chip-level timing closure includes the chip-level CTS (CCTS) step in which a chip-level clock tree is synthesized to drive all the IP-level clock trees. The primary objective of CCTS is that the full clock tree, which includes the chip-level and all the IP-level clock trees should be balanced and have less skew across all the corners. Skew in a given corner is defined as the maximum difference in the insertion delays of any two clock sinks in that corner. Reducing the skew across all corners prevents data mismatch as well as avoids the use of data lock-up latches [1]. Minimizing skew is relatively easy when considering only the nominal delay corner. However, the different IPs of an SoC are timing-closed independently by different individuals/teams, possibly using different methodologies, tools, and library cells. In such cases, achieving good skews for the entire clock tree of the chip across all the design corners is a very challenging task. This is primarily because of the possible difference in the way the delays and skews of the different clock-trees of the IPs scale, either because of difference in the clock structures or the difference in the relative significance of cell and interconnect delays between the IPs.

Another important objective for chip-level CTS is to minimize the clock divergence (see Section II-A for detailed explanation). This helps to minimize the maximum possible skew variation between the critical timing paths between the IPs and thus improves the overall yield. This also helps in faster timing closure in real designs as most clock tree analysis algorithms [5] consider the fact that process variations in common part of the clock tree do not affect the skew between a given register pair. Clock divergence reduction is a trivial problem when either the number of IPs are very small or when they do not interact significantly. Both these conditions do not apply to the SoC designs of today which have a significant number of IPs which interact in a complex way, with critical paths present between multiple overlapping pairs of IPs [1], [2].

In many complex chips, CCTS work is often custom/manual [3], [4] so as to achieve the precise skew and divergence objectives, but this is often very time consuming. Also, as the complexity and size of SoC designs increase, custom/manual chip-level CTS will become increasingly difficult. Thus, fully automated methods to address the CCTS problem are needed. Though there are a few works related to global clock distribution [6]–[9], they make the assumption that a H-tree topology is sufficient and focus on improving the quality (skew, power, and so on) of the H-tree. Similarly, works like

Manuscript received March 10, 2010; revised June 30, 2010 and October 8, 2010; accepted December 20, 2010. Date of current version May 18, 2011. This work was supported in part by NSF, SRC, and the IBM Faculty Award. This paper was recommended by Associate Editor Y.-W. Chang.

A. Rajaram is with Magma Design Automation, Austin, TX 78731 USA (e-mail: anandrajaram@ieee.org).

D. Z. Pan is with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712 USA (e-mail: dpan@ece.utexas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2106852

[10] and [11] focus on variation reduction on general clock trees but do not directly address the issues of divergence and multicorner skew reduction that are very important for CCTS problem. For rest of the paper, multicorner skew is defined as the maximum of skew among all the different corners. This paper attempts to address the CCTS problem. The key contributions of this paper are as follows:

1) a 0–1 quadratic programming (QP)-based clock pin relocation scheme for soft-IPs to reduce chip-level clock divergence;
2) an effective method to reduce the chip-level clock tree skews simultaneously across different PVT corners;
3) a dynamic programming based CCTS algorithm that simultaneously reduces clock divergence and multicorner skew.

To our best knowledge, the above contributions make the first comprehensive solution to the CCTS problem for complex SoC designs. A preliminary version of our research was published in [12]. Compared to [12], this paper has detailed explanations, experimental results with more test-cases and also a faster CCTS algorithm. It may be noted here that CCTS problem is significantly different from the IP-level CTS problem discussed in well known CTS works like [13]–[16]. In these works, the main problem is to reduce the overall delay and skew at the IP level, where there are no pre-existing clock trees. There is no consideration given to issues like divergence, multicorner skew balancing and clock pin assignment. Another key difference is their place in the overall design flow. IP-level CTS is done much before top-level chip integration and also before timing closure of the individual IPs. On the other hand, our pin-assignment algorithm will be done before IP-level CTS and our main CCTS algorithm will be used only during the top-level chip integration. The readers are referred to the work of [17] for a detailed survey of IP-level CTS algorithms.

II. Motivation and Problem Formulation

In this section, we will first discuss the significance of clock divergence, the effect of clock pin assignment on clock divergence and multicorner skew reduction using a few simple examples after which we will formulate the chip-level CTS problem. Fig. 1 shows a simple example of a chip-level CTS problem. The IPs shown might be either hard-IPs or soft-IPs. In the case of hard IPs, the clock pin location and the clock tree itself will be fixed. For soft-IPs, CTS will be done as a separate step along with IP-level timing closure and then integrated at the chip-level.

A. Significance of Clock Divergence Reduction

The significance of reducing clock divergence between registers in *timing-critical* paths is well known [17]. For a given overall delay, the lesser the divergent delay between the such register pairs, the lesser is the value of maximum skew that can be seen between them. This is because any variation in the common clock path will not impact the skew between the register pair. This is illustrated in Fig. 2. In this example, assuming all other conditions are same, Case A is better for timing yield in the presence of variation because skew variation in Case A is limited only to the variations in last clock

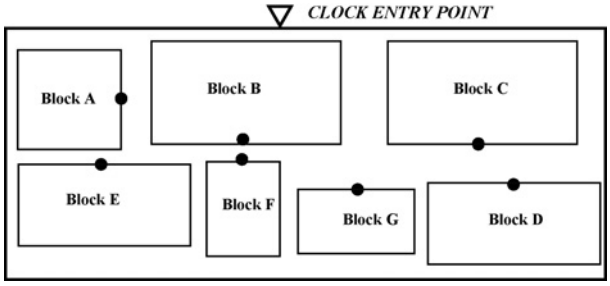

Fig. 1. Simple chip-level CTS example. The black circles represent the clock root for each IP.

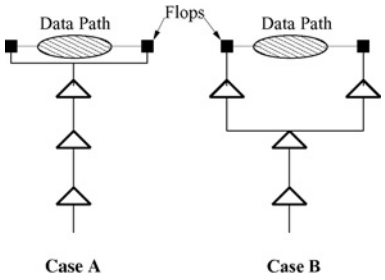

Fig. 2. Even for identical nominal skews, Case A is better than Case B because of lesser clock divergence and hence lesser skew variation.

net. However, in Case B, since the last buffer is not shared, the magnitude of possible skew variations increases, thereby impacting the timing yield in the presence of variations.

1) *Significance of Clock Divergence Reduction in CCTS:* The same principle of clock divergence reduction discussed above is also applicable at the chip-level where different IPs interact with each other instead of register pairs. In some cases, clock divergence reduction between specific IPs might be extremely important to ensure good timing yield. For example, when the clock tree divergence between two heavily interacting IPs is high, it might result in significant skew variation between all the register pairs between the IPs. If some of these register pairs were already timing-critical, the increased skew variation will only exacerbate the situation, thereby affecting the timing yield.

B. Impact of IP Clock Pin Location on Clock Divergence

Unlike hard IPs, the clock pins of the soft-IPs can be changed specific to a given chip and floorplan. This additional flexibility for the soft-IPs can be effectively used toward clock divergence reduction between critical IPs. Fig. 3 shows a simple example where the clock pin assignment might make a difference in clock divergence reducing. In this example, IPs A and B are assumed to have critical paths between them. Thus, the pin assignment in Case B is better since it reduces the clock divergence (and hence the maximum clock skew under variation) between the flip–flops in the critical path.

C. Measuring Divergence

In this section, we explain briefly as to how clock divergence can be measured for a given clock tree. Consider Fig. 4 in which a simple four sink clock tree is shown. Points A–D represent the four sinks and points P1 and P2 are the internal nodes of the clock tree. If we consider only a single pair of sinks, measuring divergence is trivial as we only need to

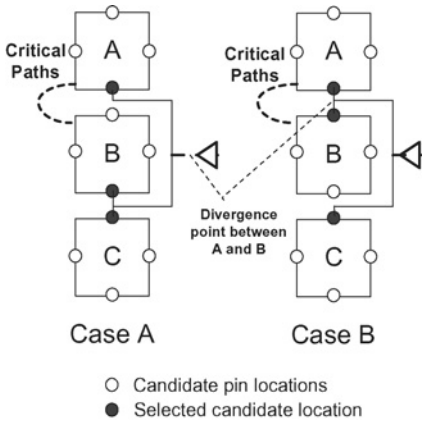

Fig. 3. Importance of clock pin assignment for IPs. Case A and Case B differ in the clock pin location for IP B, which affects CTS. If IPs A and B have critical paths between them, Case B will result in better yield because of reduced clock divergence between A and B.

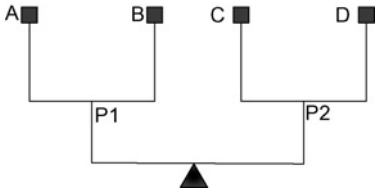

Fig. 4. Example for measuring divergence.

know the sum of clock delays that is not shared by the given sink pairs. However, when considering more than one sink pair, such a direct measurement of divergence is not correct because not all sink pairs are equally critical from timing perspective. For example, while considering all the four sinks of Fig. 4, there are six potential sink pairs and thus we need to consider the relative importance of each pair while calculating the divergence for the entire clock tree.

The relative importance of the different sink pairs can be represented by a pairwise weights proportional to the timing criticality of the path between the sink pair. If there is no valid timing path between a pair of sinks, then the corresponding weight is zero. This concept can be easily extended as more clock sinks and timing paths are added. Similarly, clock divergence at the chip-level can be measured as the weighted sum of clock divergence between the clock trees of the different IPs. The weight used for a pair of IPs will be proportional to the timing criticality of all the paths between the pair. Please note that the timing criticality information can be obtained directly from the timing analysis usually done with ideal clocks just before CTS. The actual weights might be made proportional to either the worst negative slack or the total negative slack of all paths between the given pair of IPs. Thus, for a given chip-level clock tree with N IPs, the value of divergence can be expressed as

$$divergence = \sum_{\forall i,j} W_{i,j} * \left({D_i}^F + {D_j}^F - 2 * {D_{i,j}}^C \right). \quad (1)$$

In the above equation:

1) $1 \leq i, j \leq N, i \neq j$; i and j denote the IP numbers;

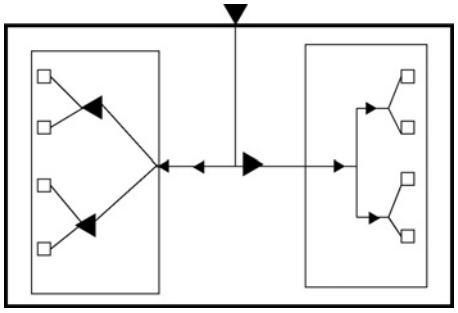

Fig. 5. Simple example illustrating difficulty of balancing two different IPs. The clock tree delays of the two IPs will scale differently across different corners due to different buffer sizes and interconnect lengths.

2) ${D_i}^F$ is the average insertion delay for all registers in IP i from the root of the chip-level clock tree;
3) ${D_{i,j}}^C$ is the insertion delay of common clock path between IP i and IP j;
4) $W_{i,j}$ is weight that is proportional to the timing criticality of timing paths between the IPs i and j, obtained from the timing analysis done before CTS.

D. Multicorner Skew Reduction Problem

In the chip-level CTS problem, each of the sub-trees shown in Fig. 1 are assumed to have full clock trees in them with fixed clock input pins. In addition, we also know the delay/skew of each of the clock trees across all the PVT corners. This information will be necessary for balancing the chip-level clock tree across all PVT corners. To understand the difficulty in reducing the skews at the chip-level across multiple design corners, consider Fig. 5 where only two IPs are present. The squares in the IPs represent clock sinks. The left-side IP has bigger buffers with longer interconnects and the right-side IP has smaller buffers with shorter interconnect. Let us assume that clock trees of both the IPs have identical delays in the nominal corner. However, their delays across different design corners will be different, mainly because of the difference in the interconnect lengths and buffer sizes. To balance these two clock trees across all corners, the chip-level clock tree should be built such that the differences in the delays, *across all corners*, between the two clock trees gets exactly (or nearly) compensated at the chip-level. In our example, we can attempt to do this by driving the left-side IP with small buffers and short interconnect and the right-side IP with bigger buffer and longer interconnect as shown in Fig. 5. In most SoC designs, there will be several IPs having clock trees with significant differences in their size, structure, buffer sizes used and interconnect lengths. Thus, synthesizing a chip-level clock tree that can simultaneously reduce the skew across all corners by accounting for these differences while not significantly increasing the overall delay is a challenging problem.

1) *Problem Formulation:* We formulate the overall CCTS problem into the following two sub-problems.

a) *Given:* chip-level floorplan and criticality of clock divergence between all IP pairs.
Problem: select the clock pin locations of all soft-IPs to reduce clock divergence between critical IP pairs.

b) *Given:* all information from the previous step and also information on clock tree delays/skews across all corners for each IP.
Problem: obtain a chip-level clock tree such that the skews and delays across all corners are reduced, while simultaneously reducing the weighted sum of clock divergence between all the IP pairs. The value of weight for a given IP pair is directly obtained from the number and timing criticality of paths between them. In general, the more paths a given IP pair and the higher the timing criticality of those paths, the higher the value of the weight for that pair.

2) *Tradeoff Between Divergence Reduction and Delay Reduction:* In some cases, we might be able to achieve lesser clock divergence by increasing the overall delay of the clock tree and vice-versa. One simple way to quantify this tradeoff is to use a scaling factor that will determine the percentage of delay increase that can be tolerated for a given reduction in clock divergence. Using this scaling factor, we can define the overall cost as follows:

$$Cost = x * Max\_Delay + (1 - x) * DIV\_COST \quad (2)$$

where $DIV\_COST = \sum_{\forall i,j} W_{i,j} * \left({D_i}^F + {D_j}^F - 2 * {D_{i,j}}^C \right)$.

In the above equations:

1) x: variable with value between 0 and 1 to quantify delay and divergence tradeoff;
2) Max_Delay: maximum delay to any sink in the entire clock tree;
3) DIV_COST: clock divergence cost between all IPs pairs;
4) i,j: the IP numbers, with $1 \leq i, j \leq N, i \neq j$;
5) $W_{i,j}$: criticality of clock divergence between IPs i; j;
6) ${D_i}^F$: average delay from clock root to the flip–flops in IP i;
7) ${D_{i,j}}^C$: the maximum *shared or common* delay between any two IPs i, j;
8) all the delay information are with respect to the nominal corner values.

Thus, the objective of the CCTS problem is to get a chip-level clock tree that can minimize the above cost function while simultaneously reducing the skews across all corners. It may be noted here that the above formulation assumes that all the logic in the complete chip is divided into IPs on which CTS has been completed. In many practical situations, glue logic to integrate the different IPs will also be present at the chip level. However, such situations can also be handled by the above formulation by dividing up the glue-logic itself into one or more virtual IPs and doing separate CTS on them from a common set of clock source points. At this point, we can apply the above formulation.

III. Clock Pin Assignment Algorithm for Clock Divergence Reduction

Given a floorplan and criticality of clock divergence between all IP pairs, the clock pin assignment aims to identify the location of all the clock pins of each soft-IP *even before any CTS is done on them*. For example, this step may be done after the floor-planning stage of the chip design and before the timing closure of the individual IPs starts. We restrict the possible clock pin locations to the mid points of one of the four sides of each IP. This minimizes the distance between the clock pin and the farthest register and can result in reduced clock tree delay. When the flop distribution is not uniform within a given IP or when there are multiple clocks present in a given IP, we locate each clock pin such that it divides the sink distribution it drives into roughly two equal halves, either in the horizontal or vertical direction. Under this assumption, clock pin assignment problem can be formulated as follows:

$$\begin{aligned} &Minimize \sum {x_i}^p * {x_j}^q * W_{i,j} * Top\_Level\_Dist({B_i}^p, {B_j}^q) \\ &s.t. \sum {x_i}^p = 1, \quad {x_i}^p \in \{0, 1\} \\ &where\ 1 \leq i, j \leq N, i \neq j; \quad 1 \leq p \leq 4; \quad 1 \leq q \leq 4. \end{aligned} \quad (3)$$

In the above equations:

1) i and j denote IP numbers;
2) p and q denote one of the four pin locations on a given IP;
3) binary variable ${x_i}^p$ represents if a given pin location p is selected for IP i;
4) ${B_i}^p$ denotes the IP i with pin location at p;
5) $W_{i,j}$ denote the criticality of the paths between IPs i and j;
6) $Top\_Level\_Dist({B_i}^p, {B_j}^q)$ represents the Manhattan distance between pin location p of B_i and q of B_j.

The conditions that each of the variables ${x_i}^p$ should be either 0 or 1 and that the sum of all the variables for a given IP should exactly be 1 makes sure that exactly one pin location is selected for each IP. The cost function being minimized is the weighted sum of distances between all the clock pins of all IP pairs where the weight is the criticality of the paths between a given IP pair. Minimizing the distance between two pins will directly increase the chances of clock delay sharing between the two IPs. The only variables in the above optimization problem are ${x_i}^p$ and since they can only take values of either 0 or 1, the above problem is a 0–1 quadratic programming problem. Though this problem is NP-hard, efficient heuristics are available to solve this problem [18]. It may be noted here that, though prior work [19] solves a similar problem, the formulation is not suitable when different IP pairs have different criticality values.

A. Impact of Pin Assignment on Delay at the IP-Level

The above formulation ignores the impact of clock pin assignment on the IP-level clock tree delays, which might end up increasing the overall delay or even clock divergence. However, the formulation can be made to account for IP-level clock tree delays by introducing an additional weighting term of the form ${K_i}^p$ that denotes the criticality of assigning the pin location p for IP i with regards to the IP-level clock tree. For example, if all four sides are equally acceptable for the IP-level CTS of IP i, then the value of ${K_i}^p$ will be identical for all four values of p. If on the other hand, we want to make a particular pin location more likely, we can increase the corresponding scaling factor. The relative values for these

factors may be obtained by a weighted sum of distances of all the registers from each of the four pin locations. Thus, the objective function for equation 3 can be modified as

$$Min \sum \frac{x_i{}^p * x_j{}^q * W_{i,j} * Top\_Level\_Dist(B_i{}^p, B_j{}^q)}{(K_i{}^p + K_j{}^q)} \quad (4)$$

where the new term $K_i{}^p$ can be increased to give more weightage to a particular location p for any IP i. In practice, $K_i{}^p$ for a given IP i can be obtained by estimating the insertion delay in the IPs can be modeled as a function of the pin placement. For example, we can assume that the maximum delay in the IP is roughly proportional to the distance of the farthest clock sink from the clock pin location. The precise details of such modeling schemes will depend on the CTS algorithm used for the IP-level CTS. Since our objective is to consider the chip-level clock balancing requirements even before CTS on any of the IPs is completed, even a rough modeling of IP-level delays will be sufficient for our purpose.

IV. Multicorner Skew Reduction Algorithm

In this section, we will address the problem of merging any two clock trees such that their combined skews across all the corners are reduced. This problem can be divided into two categories. In the first, the clock pins are located very close to each other *and* their delays across all corners are very similar. In this case, the multicorner skew balancing is trivial since it is possible to merge the clock pins with just interconnect without adding an extra buffer level. In the second case, the clock pins are far apart *and/or* they have significantly different delays across the corners. In such situations, we need to add one or more 1-fanout buffer stages (with appropriate buffer sizes/interconnect lengths) to the root of the sub-tree with lesser delay to reduce the multicorner skew between the two sub-trees. Thus, to reduce the multicorner skew between any two sub-trees for the non-trivial situation, we need a method to select the appropriate number of buffer stages and the size/lengths of the buffers/interconnects to be used to merge the clock pins of the two IPs. In future discussions, we call the selection of appropriate buffer size/interconnect length as selection of a *buffer configuration*.[1] Fig. 6 shows examples of *buffer configuration* for both 1-fanout and 2-fanout situations. Please note that adding a buffer configuration implies adding only $BUF1$ on top of existing sub-tree(s) at appropriate distances from the current root(s) of the sub-tree(s). For example, if we add a buffer configuration to a given sub-tree, it means adding $BUF1$ in Fig. 6(a) at a distance of L_0 from the current root of the sub-tree, which is denoted by $BUF2$. The Cap_1 in the figure is the effective capacitance of the sub-tree driven by $BUF2$. Similarly, if we merge two sub-trees using a fanout-of-2 buffer, it means adding $BUF1$ in Fig. 6(b) at a distance of L_0 from the merge point of the two sub-trees. The distance of the two sub-trees from the merge point are denoted in Fig. 6(b) by L_1 and L_2. To summarize, the problem of multicorner skew balancing of a given pair of IPs can be translated to the problem of picking the right buffer configurations to be added on top of the slower sub-tree to bring the multicorner skews between the sub-trees to desired levels.

[1]Please note that interconnects of different widths and spacings can also be considered in the same framework, similar to different buffer sizes.

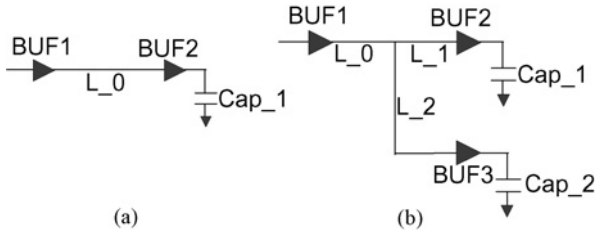

Fig. 6. Buffer configurations used for multicorner delay characterization. (a) Single fanout case. (b) Double fanout case.

A. Special Properties of CCTS Problem

To solve the problem of picking the right buffer configurations for multicorner skew reduction, following special properties of CCTS problem can be exploited.

1) Unlike CTS on a flat design, the CCTS problem will have just a hand full of end points (clock pins of IPs) that are much more spread apart in distance than typical registers. This is because the number of IPs in a typical SoC will be orders of magnitude lesser than the number of flip–flops in the whole design.
2) Since the clock pins of the IPs are far away from each other, the *typical* fanout for a buffers in the chip-level clock tree will be considerably less compared to the IP-level clock trees. In most practical cases, this can be as low as 1 or 2.

B. Steps to Choose Buffer Configurations for Multicorner Skew Reduction

In order to distinguish between the different buffer configurations and select the right set of configurations to achieve multicorner skew reduction, we can follow the following steps.

1) Restrict the maximum fanout for any chip-level clock buffer to just 1 or 2. Thus, the buffer configurations that will be added on top of sub-trees while merging them will be as shown in Fig. 6(a) or (b). The clock power/area penalty due to this restriction will be negligible because the fanout of most buffers is expected to be small anyway. Also, the number of chip-level clock buffers will be small compared to the total number of buffers of all the IPs combined.
2) The fanout restriction drastically reduces the number of possible buffer configurations, enabling us to do the multicorner delay characterization for each configuration quite easily. For example, in Fig. 6(a), the input slew (in 5 ps increments), buffer type of the driver ($BUF1$), interconnect lengths (L_0) (in 25 μm increments) and the load buffer type ($BUF2$) are the variables. Since this is a simple circuit, the complete multicorner delay characterization of all possible configurations *and* across all corners typically takes just a few minutes. This is similar to the typical cell delay characterizations used in ASIC designs, with the added explicit variables of interconnect length and load cell being driven. Similarly, we

characterize all possible 2-fanout configurations using the template in Fig. 6(b).

3) The next step is to get what we define as cross-corner delay ratios (CCDR) for every buffer configuration. For each buffer configuration, we first get the SPICE delays across different corners obtained from characterization. Then, we normalize (divide) the delays across all K corners with the nominal-corner delay of that configuration. After the normalization, each buffer configuration will have a *vector* of K numbers, corresponding to K corners, called its CCDR. Obviously, the ratio number corresponding to the nominal corner will always be 1. This normalization helps us to compare the *relative* cross-corner scaling of different buffer configurations and choose the appropriate one for merging any given sub-tree pair. For example, if a buffer driving a 500 μm interconnect has delays of 50, 100 and 200 ps in the fast, nominal and slow corners respectively, then the delay vector for this configuration will be (50, 100, 200). To obtain the CCDR vector, we divide each element in this vector by the nominal corner delay of 100. Thus, the CCDR for this case will be (0.5, 1.0, 2.0). If another buffer driving a 300 μm load has a CCDR vector of (0.4, 1.0, 1.8), then we can conclude that the second configuration relatively speeds up the fast and slow corners than the first configuration. Thus, the CCDR vector for a given sink of tree can be defined as

$$CCDR = [D_1/D_{\text{nominal}}, ...D_i/D_{\text{nominal}}, ...D_K/D_{\text{nominal}}] \tag{5}$$

where D_i represents the ith corner and D_{nominal} is the arbitrarily chosen nominal corner among the available K corners. Please note that CCDR for a given sub-tree can be defined in the same way using either the maximum delays in different corners or using the average delays in different corners.

The concept of CCDR described above is used in our multicorner sub-tree merging heuristic shown in Fig. 7. The basic idea behind this heuristic can be explained by a simple example. Let A and B be two sub-trees that we want to balance across three corners—fast, nominal, and slow. Let the delays for the two IPs in the three corners be A(50, 100, 200) and B(40, 100, 220), respectively. Such differences in delay scaling across corners can happen when different clock buffer types, CTS tools/methodologies are used in the two IPs. If the two clock trees are merged using a *zero-nominal-skew* chip-level clock tree, then the merged tree will have zero skew at nominal corner, but higher skews at the fast and slow corners. In order to achieve good skews across all three corners, we should build the chip-level tree such that $del\_to(A, \text{nominal}) \simeq del\_to(B, \text{nominal})$ and $del\_to(A, \text{fast}) < del\_to(B, \text{fast})$ and $del\_to(A, \text{slow}) > del\_to(B, \text{slow})$, where $del\_to(A, \text{nomi} - \text{nal})$, and so on represent the chip-level clock-tree delay to the clock pin of A in the nominal corner. Chip-level clock trees with such precise cross-corner delay scaling requirements can be constructed by selecting the buffer configurations with appropriate CCDR.

Procedure: $Multi\_Corner\_Subtree\_Balance(S_A, S_B)$
Input: Location, delay for both sub-trees S_A & S_B.
Output: New sub-tree S_C combining S_A & S_B.

1. Get CCDRs of S_A, S_B w.r.t. nominal corner;
2. Set $Sub\_trees\_not\_close = 1$ if skew of S_A and S_B are not close;
3. S_P = Sub-tree with min nominal-corner delay;
4. S_Q = Sub-tree with max nominal-corner delay;
5. While ($Sub\_trees\_not\_close == 1$)
   (i) Select the best buffer configuration to be added to S_P to brings the CCDR values ***closest*** to S_Q without exceeding its delay. That is, we want to add delay to S_P without making it slower than S_Q in the nominal corner.
   (ii) Update delay and CCDR information for S_P;
   (iii) If skew across corners $<$ limit $Sub\_trees\_not\_close = 0$;
6. Using length of the selected buffer configurations, get the Manhattan ring for S_P within which the root could be located;
7. If (Manhattan ring of S_P intersects with root point of S_Q)
   Root = root point of S_Q;
   Do wire-snaking to preserve skews;
   Else
   Select point on Manhattan ring of S_P closest to root point of S_Q and merge them using a simple symmetric (0 skew) tree;
8. Name the new sub-tree as S_C and return S_C;

Fig. 7. Multicorner skew balancing heuristic.

This is the key idea behind our multicorner sub-tree balancing heuristic shown in Fig. 7.

In the above procedure, the sub-tree with lesser nominal corner delay is denoted by S_P and the other is denoted by S_Q. We evaluate the impact of adding each of the potential buffer configurations to S_P on its CCDR and finally select the configuration that results in bringing the CCDR vectors of S_P and S_Q closest in terms of the least-squares distance between them. This process is repeated till the delays of both the sub-trees are fairly close to each other across all corners. At this point, the exact configurations to be added at the roots of both sub-trees A and B to minimize their multicorner skew are available. However, the location of the merging point of the two sub-trees is still not yet fixed. For the sub-tree S_P, the total lengths of all the interconnects added with buffer configuration gives the radius of the Manhattan ring within which its root pin is to be located. If the Manhattan ring of sub-tree S_P intersects with the root pin of S_Q, then the current root of S_Q can be selected as the merged root with appropriate wire-snaking to preserve the skews. If the Manhattan rings do not overlap, it means that though the two sub-trees have similar delays, we need to add more buffer levels to both of them to physically merge them. To achieve this, we identify the closest point/segment on the Manhattan ring of S_P to the root of S_Q and merge them with a perfectly symmetric tree. This will ensure that the multicorner skew balancing already completed between the two sub-trees is not affected. It may be added here that exact location of the root of the merged sub-tree can be *deferred* in the same manner as in the DME algorithm.

It shall be noted that the above multicorner sub-tree balancing procedure inherently assumes the following procedure.

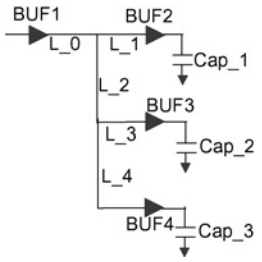

Fig. 8. Buffer configuration for a 3-fanout case.

1) The skew target in each corners is bigger than at least the delay of the smallest buffer in that corner. Otherwise, the skew condition in line 5-(iii) of the algorithm will never be met and the loop will go on indefinitely.
2) All the buffer sizes used at the IP level CTS are available for use at the chip-level CTS. Otherwise, there might be some buffer sizes that scale differently from others across corners which can not be compensated at the chip level.

The above procedure is suitable only in the limited context of chip-level CTS and is inefficient in terms of buffer resources for CTS on a flat design. Since the number of IPs will be several orders of magnitude less than the number of flip–flops in the design, the chip-level CTS can afford to adopt the above approach. It may be noted here that the restriction of the number of fanouts to 1 or 2 can be relaxed by increasing the number of buffer configurations that are characterized. For example, Fig. 8 shows how the concept can be extended to a 3-fanout case. Compared to a 2-fanout case, the 3-fanout case has more variables to be changed during characterization. This results in a significant increase in the number of buffer configurations to be evaluated. These new buffer configurations can be used in situations where there are multiple sub-trees with similar delays located very close to each other such that a single buffer can drive them. Also, the procedure in Fig. 7 needs to be modified to account for the fact that more than two sub-trees can be merged simultaneously. One way to do this merger is as follows. Given k sub-trees to merge, find the best k-fanout buffer configuration to bring their CCDRs closer to each other and merge them. If we cannot merge the k sub-trees with any available buffer configuration, then we can add more single fanout buffer configurations on top of the sub-trees to bring them closer to each other—-both in terms of physical distance and also in terms of their multicorner delays. However, with successive relaxation to the fanout limit, the gain in terms of reduced buffer area will diminish while the runtime will increase since a much larger number of buffer configurations should be evaluated.

Example: Finally in this section, we would like to give a simple example that illustrates the algorithm in Fig. 7. Let us assume, for the sake of simplicity, that there are only two corners, nominal and slow. We denote the delay in these two corners of a given tree or a sink in a tree as an ordered pair of numbers like (10, 20), all numbers in picoseconds (ps). Let there be exactly three different buffer configurations, each with the same buffers but with different interconnect lengths. We denote them by BC_A, BC_B, BC_C. Let the delays of these three buffer configurations be (4.9, 8), (5, 10), and (5.3, 12), respectively. This means, for very similar delays in the nominal corner, these three buffer configurations have significantly different delays in the slow corner. This can happen since they are driving different interconnect lengths. Now, let S_P sub-tree of Fig. 7 have delays of (19.7, 38) and S_Q sub-tree have delays of (25, 50). Now, S_Q has higher delay than S_P, so we will have to recursively add buffer configurations on top of S_P till the delays of the two sub-trees are very close. Let us here assume that the skew requirement for the algorithm to stop is 0 ps. Since the skew between S_P and S_Q is not zero, the algorithm will enter the while loop of Fig. 7. Now, we have to select the best buffer configuration from the three available configurations to add to S_P. We iteratively go through each of the buffer configurations and find the best configuration to be added to S_P to bring its CCDR closest to that of S_Q. From among the three configurations, we can see that adding the buffer configuration BC_C not only brings the CCDR of S_P to the same values as S_Q, but also brings down the skew to 0ps. Thus, after adding the buffer configuration BC_C to S_P, its delay is identical to that of S_Q. At this point, the algorithm will exit the while loop in Fig. 7 and proceed to physically merge the two sub-trees.

V. Chip-Level CTS Algorithms

In this section, we discuss four different chip-level CTS algorithms with varying degrees of complexity. Please note that only the *dynamic programming based algorithm* is newly proposed in this paper. The other three algorithms are simple modifications of existing CTS works used for comparison.

A. Single-Corner Approach

This algorithm is a direct application of existing CTS algorithms to the CCTS problem in which only one corner delays are used. The algorithm recursively merges sub-tree nodes which are the nearest neighbors in a manner similar to that of well known CTS algorithms [13]–[16]. If a given sub-tree cannot be merged with any other sub-tree without violating the slew limits, a buffer is added on top of the sub-tree to extend the possible merging region for the sub-tree. The buffer sizes for merging two sub-trees are chosen in such a manner to reduce the total amount of buffer area added. *The results from this approach will be used as the baseline for rest of the algorithms.*

B. MultiCorner Approach

This approach is identical to the single-corner approach with one key difference: the consideration of multicorner skews. During the process of merging two sub-trees, the method described in Fig. 7 is used instead of using only one corner delay. At each step, the sub-trees that are closest to each other are merged recursively till only one sub-tree remains. The results from this approach will be used to do the cost versus benefit analysis of multicorner skew reduction.

C. Greedy CCTS Algorithm

This algorithm is a simple modification of the work of [20] in which every sub-tree merger is done to minimize the cost (wirelength or buffer area) of that merger. In our

modification, the merging cost as defined by (2) instead of wirelength. During each iteration, the merging cost of all possible pairs are evaluated and only the best pair is selected for the actual merger. The selected pair is then merged using the multicorner skew reduction method of Fig. 7. This is done repeatedly till all the different sub-trees are merged. Since [20] is one of the best algorithms for prescribed-skew CTS, the results from this approach will help us determine if existing prescribed skew (useful skew) CTS algorithms can be modified for solving the CCTS problem.

D. Dynamic Programming Based CCTS Algorithm

First, we would like to briefly describe why the CCTS problem is amenable to dynamic programming approach. A key requirement for a problem to be solvable by dynamic programming is that it exhibits optimal substructure [21]. That is, the optimal solution to a problem should contain the optimal solutions to the sub-problems also. This is clearly satisfied in the CCTS problem. For example, without loss of generality, let the final optimal [in terms of (2)] chip-level tree have a fanout of two at the root. The final cost of the full tree itself can be re-written as a sum of the costs of the two sub-trees and the cost of their merger. Since the whole cost is optimal, it follows that the cost of merging the given two sub-trees should also be optimal. This, in turn, implies that the cost of each the two sub-trees are optimal as well. Another feature of CCTS problem that makes dynamic programming suitable is the presence of overlapping sub-problems. For example, if two IPs are very close to each other with identical delays, then the cost of merging them to form a sub-tree will be very small. This might allow this sub-tree to be a part of many bigger sub-trees considered simultaneously. Thus, our dynamic programming based CCTS algorithm, shown in Fig. 9, follows the same general outline of typical dynamic programming solutions.

For subsequent discussions, we use the following terminologies. An *active sub-tree* is one that has not yet been eliminated/pruned from subsequent merging operations. The list of active sub-trees directly correspond to the current list of sub-trees considered as a solution to the CCTS problem. A *new* sub-tree in the list of active sub-trees is one that has not gone through even a single round of mergers with other active sub-trees.

1) *Overall Algorithm:* In our top-level algorithm given in Fig. 9, the basic idea is to start with individual IPs with zero cost as partial solutions to the CCTS problem. Each of the partial solutions to our CCTS problem is characterized by two metrics: the IPs covered by each solution and the cost of building that sub-tree according to step 2.[2] These partial solutions are recursively merged to form bigger solutions until one or more solutions contain all the IPs in the CCTS problem. When many viable solutions containing all the IPs are available, the one that costs the least to build will be chosen as the final solution.

A naive recursive merger of sub-trees will result in an exhaustive enumeration of all possible solutions and will result in exponential runtime with respect to the number of IPs in the design. To prevent this, our algorithm uses two effective pruning methods, *Pre_Eliminate* (Fig. 10) and *Post_Eliminate* (Fig. 11), that drastically reduce the number of solutions considered without sacrificing the quality of the final results. Now, let us discuss the details of Fig. 9. In step 1, all the clock pins of IPs are marked as *new* and *active sub-trees*. Each of these solutions will have a zero cost since we have not done any mergers yet. Step 2 of Fig. 9 is the core part of our algorithm in which we iteratively combine existing sub-trees to progressively get bigger sub-trees, eventually getting one or more solutions that drive all the IPs. In each iteration of step 2, we use the Pre_Eliminate procedure to get the new set of valid sub-tree pairs that are considered for merger in the next iteration. The valid sub-tree pairs are merged using the multicorner sub-tree balancing algorithm of Fig. 7 at the end of which, each merged sub-tree will have a specific cost as defined by (2). Each of these merged sub-trees represent a new bigger sub-tree formed by combining existing sub-trees.

Next, we use the Post_Eliminate procedure to eliminate any sub-optimal solution from the list of all the current active sub-trees and the newly generated sub-trees. Then, we mark the status of all the original sub-trees as *old* since we have completed one round of mergers among them. All the newly created sub-trees have not yet been merged with the other sub-trees. So we mark their status as *new*. The status values of the active sub-trees are used in the next round of Pre_Eliminate procedure. This procedure continues till there are no more newly created sub-trees from existing solutions, at which point we choose the minimum cost complete solution as the final solution to the CCTS problem. Interested readers may also refer to [22] that gives a simple animated example of how the overall dynamic programming based algorithm works together with the elimination steps.

Procedure: *Dynamic_Programming_Top_Level_CTS*
Input: Location and delay information for all IPs.
Output: Chip-level clock tree with min delay, divergence

```
1. Initialize
   a. Mergers_Completed = 0;
   b. Active_SubTrees = Clock Pins of all IPs;
   c. For each subtree ∈ Active_SubTrees
        Status(subtree) = new;
2. While ( Mergers_Completed == 0 )
   a. Valid_Pairs = Pre_Eliminate(Active_SubTrees);
   b. Generate merging cost of all Valid_Pairs using Eq.(2);
   c. New_SubTrees = Subtrees from each pair in Vaid_Pairs;
   d. Potential_SubTrees = Active_SubTrees + New_SubTrees;
   e. Deleted_SubTrees = Post_Eliminate(Potential_SubTrees);
   f. New_Additions = Potential_SubTrees - Deleted_SubTrees
                      - Active_SubTrees;
   g. For each subtree ∈ Active_SubTrees
        Status(subtree) = old;
   h. For each subtree ∈ New_Additions
        Status(subtree) = new;
   i. Active_SubTrees = Potential_SubTrees - Deleted_SubTrees;
   j. if (Number of New_Additions == 0)
        Mergers_Completed = 1;
3. Pick sub-tree in Active_SubTrees with all IPs
   and minimum cost;
```

Fig. 9. Dynamic programming based approach to chip-level CTS. The sub-steps are highlighted and explained separately in Figs. 10 and 11.

[2]We ignore the impact of input pin capacitance since in the chip-level CTS context, the wire capacitance dominates pin capacitance. So impact of input pin capacitance on delay is small.

Procedure: $Pre\_Eliminate(Active\_SubTrees)$
Input: All active sub-trees.
Output: All pairs of sub-trees that are selected for merger.

1. α and β are user given parameters.
2. Valid_Pairs = {}; Number sub-trees from 1 to N_s;
3. For $i = 1$ *to* N_s
   For $j = i + 1$ *to* N_s
   Sub-tree pair considered: S_i and S_j;
   If (Status(S_i) == new ***OR*** Status(S_j) == new)
   If (No overlap between S_i and S_j)
   PreElim_Cost(S_i,S_j) =
   $[dist(S_i, S_j) + del\_diff(S_i, S_j)/\alpha]$ /C(S_i,S_j);
   where C(S_i,S_j) is given by Equation 6.
   PreElim_Cost(j,i) = PreElim_Cost(i,j);
4. For $i = 1$ *to* N_s
   Top_pairs_for_S_i = Top β pairs with minimum PreElim_Cost(S_i, S_j)
   Selected_Pairs = Selected_Pairs + Top_pairs_for_S_i
5. Return Selected_Pairs ;

Fig. 10. Procedure to pick valid pairs for merger from a given set of sub-trees.

2) *Pre_Eliminate Procedure:* In the Pre_Eliminate procedure shown in Fig. 10, the objective is to return only the smallest number of valid pairs for next round of merger without impacting the quality of result. This is done by taking advantage of three key properties of the CCTS problem listed below.

a) First, any merger between two *old* sub-trees can be eliminated. This is because their merger would have been already considered when at least one of them was a new sub-tree. In other words, considering a merger of two *old* sub-trees simply means we are doing the same work again. This property is used in the first *If* condition in line 3 of Fig. 10.

b) Second, we can eliminate any sub-tree pair that have even one common IP between them. This is because the presence of an IP in a sub-tree means a given IP has been physically merged with another IP in the solution. This means, any other sub-tree with that same IP cannot be physically merged with the given sub-tree. This property is used in the second *If* condition in line 3 of Fig. 10.

c) Third, any merger between sub-trees that are too far away either in terms of delay or distance between their roots is likely to be sub-optimal when other alternatives with better delay or distance matching is available. This property is made use of in the calculation of *PreElim_Cost* in Fig. 10. This cost measures the desirability of merger between any given two sub-trees that do not overlap. This cost is proportional to the physical distance between the roots of the sub-trees ($dist(S_i, S_j)$) and the delay difference between the sub-trees ($del\_dist(S_i, S_j)$). It is also inversely proportional to the number of critical timing interactions between the IPs in the two sub-trees. This last effect is captured by

$$C(S_i, S_j) = \sum_{\forall p,q} W(p, q) \tag{6}$$

where p = all the IPs in S_i, q = all the IPs in S_j and $W(p, q)$ denote the timing criticality between IPs p and q.

Procedure: $Post\_Eliminate(Potential\_SubTrees)$
Input: Full list of potentially valid subtrees.
Output: Sub-treesPairs that can be pruned because of existence of other dominating sub-trees.

1. User given parameter γ; $N1$ =0;
2. Elim_SubTrees = {};
   Number Potential_SubTrees from 1 to N_s;
3. For $i = 1$ *to* N_s
   For $j = i + 1$ *to* N_s
   P = IPs driven by sub-tree i;
   Q = IPs driven by sub-tree j;
   If (P $\subseteq$ Q AND cost(sub-tree i) $\geq$ cost(sub-tree j))
   Elim_SubTrees = Elim_SubTrees + sub-tree i;
   If (Q $\subseteq$ P AND cost(sub-tree j) $>$ cost(sub-tree i))
   Elim_SubTrees = Elim_SubTrees + sub-tree j;
4. Valid_Sub_Trees = Valid_Sub_Trees - Elim_SubTrees;
5. Sort sub-trees in Valid_Sub_Trees per descending order of # IPs in a sub-tree and then per ascending order of cost.
6. For each sub-tree $S_i \in$ Valid_Sub_Trees
   a. If (# IPs in S_i == 1)
   must_have(S_i) = 1; next;
   b. If (must_have(S_i == 1) next;
   c. must_have(S_i) = 1;
   d. IPs_Covered = IPs in S_i
   e. While (IPs_Covered != All IPs)
   i. Proceed down Valid_Sub_Trees array and pick the next sub-tree, S_j, without IPs overlap with IPs_Covered;
   ii. must_have(S_j) = 1;
   iii. IPs_Covered = IPs_Covered + IPs in S_j;
   f. N1++;
   g. If (N1 $>$ γ) Go To Step 7.
7. Add all sub-tree with must_have(S_i) $\neq$ 1 to Elim_SubTrees;
8. Return Elim_SubTrees;

Fig. 11. Post-eliminate procedure used to eliminate dominated sub-trees.

The Pre_Eliminate procedure uses two user defined parameters that are explained next. The α parameter is used as a weighing factor between the delay difference and the physical distance between the roots of the sub-trees. It is set to be the average length of interconnect that may be driven per unit of delay using a given set of buffers and a given technology under the maximum slew constraint. It is measured in terms of distance per unit delay. The other parameter used in Fig. 10, β, is an integer and is used to control how many potential pairs are to be allowed per sub-tree. β can be any integer with values of at least 1. In our experiments, β was set to a value of 2. It may be noted that in the preliminary version of this paper [12], we used two other comparable parameters with similar motivation that directly controlled the actual value of maximum allowed delay difference and distance difference between the sub-trees. However, based on our experiments on a large number of test-cases, we find the new parameters are a lot easier to set without any need to tune the parameters for individual test-cases. Thus, using the above mentioned three properties of CCTS, the Pre_Eliminate procedure selects only a few best sub-tree pairs for consideration during the next round of mergers.

3) *Post_Eliminate Procedure:* The objective of the post-elimination procedure of Fig. 11 is to compare all the existing sub-trees and weed out any inferior solutions. A sub-tree P is inferior if there exist another sub-tree Q that covers the same set (or a super-set) of clock pins covered by sub-tree P, but has same or lower merging cost. Two sub-trees that drive different sets of IPs will never be directly compared for elimination as

one cannot fully replace the other. Once the inferior solutions are identified, they are removed from the list of active sub-trees that will be considered for the next round of sub-tree mergers. This is shown in the steps 2 to 4 of Fig. 11.

In addition to the above straight forward pruning, the procedure of Fig. 11 executes another pruning that is a bit more subtle. This is shown in the steps 5 to 7 of Fig. 11. This procedure uses a user-defined integer parameter, γ, that represents the maximum number of *independent and complete* solutions that can be present in the current set of sub-trees. We first sort all the current sub-trees in descending order of number of IPs in them. When two sub-trees have the same number of IPs in them, we sort them on the ascending value of the cost. The final sorted list of valid sub-trees represent how close each sub-tree is to the final complete solution to be chosen. For example, the top-most sub-tree has the maximum number of IPs below it with the least cost. Given this sorted list of sub-trees, we move down the list from top to bottom to select a list of sub-trees that can be used to get one complete and independent solution to the CCTS problem. This step gets repeated until the total number of independent solutions reaches γ or the list of potential full solutions runs out. Any sub-tree that is not present in any of the top γ complete solutions is added to the list of eliminated sub-trees. The list of sub-trees eliminated by the Post_Eliminate procedure are dropped from subsequent iterations of the algorithm in Fig. 9.

The second pruning procedure drastically reduces the overall runtime with little impact on the final results. This is because a sub-tree that is not a part of the top γ final solutions can be eliminated with little risk as long as γ is sufficiently large. For example, in our experiments, we set γ to 200. However, keeping that sub-tree in the solution pool takes up exponentially higher runtime since it may add quite a few new solutions in the subsequent iterations without actually adding to any better results. It may be noted that in the preliminary version of this paper [12], this last pruning method was not employed. As a result, the runtime of the original algorithm does not scale as well as the new algorithm with respect to the number of IPs in the CCTS problem.

VI. Practical Considerations in CCTS

A. Generalization of Pin Assignment Algorithm

In Section III, the 0–1 quadratic programming problem was formulated assuming that the clock pins can be located in only the mid-points of the four sides. In the most generic case, a given IP can have multiple candidate clock-pin locations on each of the four sides and also candidate locations on the top of IP. This situation can be easily handled by introducing two constant weight factors for each candidate location. One new factor should account for the estimated IP clock delay for each candidate location. This factor should increase proportionally with respect to the estimated delay of IP clock tree for the candidate pin location. The second factor should consider the potential routing layer difference that might arise when clock-pin locations on top of the IP are considered. Also, another straightforward modification that can be made to the method proposed in 3 is that the variables p and q that represent the number of candidate pin locations should be changed to account for the new candidate pin locations. Thus, the original formulation in Section III is applicable generally.

TABLE I
Key Test-Case Generation Parameters

Parameter	*Value*
Chip size	$0.25\,\text{cm}^2$ to $6.25\,\text{cm}^2$
No. of IPs	10–130
Aspect ratio	0.7–1.3
Hard-IP probability	0.2
# Slew limit range	90–110 ps
Technology	65 nm

B. Consideration of Blockages

A key requirement of any chip-level CTS algorithm is that it works in the presence of blockages. All the algorithms presented in our approach to the CCTS problem can be applied even for chips with blockages. For example, the clock pin assignment algorithm can be made blockage aware by measuring the distance between any two candidate pin locations using a *blockage aware global router* instead of a Manhattan estimate. Similarly, the multicorner sub-tree balancing heuristic of Fig. 7 can be modified by using the global router based distance instead of Manhattan distance. Since the dynamic programming algorithm internally uses the multicorner heuristic, that can also be used in the presence of blockages.

VII. Experimental Results

A. Test-Case Generation

To test the effectiveness of our algorithms, we need several chip-level SoC test-cases. Since obtaining test-cases from actual SoC chips is not feasible for us and since there are no known CCTS work in the literature, we generate random test-cases using the data available on SoC chips in the literature [1]–[4].

1) *Defining SoC Chip's Physical Attributes:* First, we define reasonable *ranges* for the following variables: chip size, number of IPs, size range of the IPs, aspect ratio range for IPs, and chip density. Using these, we generate random chip-level floorplans such that the chip size, number of IPs, and so on are all within the selected ranges. We also make sure that the chip density (the ratio of the chip covered by the all IPs) is within limits and that there are no overlaps between the IPs. Each IP is marked as a hard or soft IP randomly with probabilities of 0.2 and 0.8 respectively. We would like to note here that the relative probabilities of hard and soft IPs were chosen based on our prior experience with SoC chips. We are unable to find any previous work from which we can choose this number.

2) *Generating Timing Criticality Data:* To generate a realistic timing criticality information between IP pairs, we consider how the chip-level floorplan is done. A key objective of floorplanning step is to ensure IPs that interact heavily are located close to each other. However, when the interaction between the IPs become complex, placing all the IPs that interact right next to each other becomes impossible. Also, IPs that are very far away from each other rarely have a significant number of critical paths between them. To closely

TABLE II
CLOCK DIVERGENCE, DELAY, SKEW, BUFFER AREA (BA), AND WIRE LENGTH (WL) RESULTS FOR THE TEST-CASES IN TABLE IV

TC	PAM	CCTS Alg.	Divergence (μs)			Max. Delay (ns)			Skew (% of Delay)				BA (nm^2)	WL (μm)	CPU
			NN	SS	FF	NN	SS	FF	NN	SS	FF	Worst	X 1e6	X 1e6	(s)
TC1	RND	1CA	0.13	0.16	0.10	2.44	3.00	1.98	3.95	0.91	6.16	6.16	32.32	163.27	1
		MCA	0.13	0.16	0.10	2.41	2.99	1.96	2.17	2.64	2.42	2.64	32.34	163.33	1
		MC-GRD	0.11	0.12	0.10	2.41	2.99	1.96	2.25	2.38	2.63	2.63	32.45	163.66	8
		MC-DyP	0.12	0.14	0.09	2.41	2.99	1.96	2.45	2.94	2.38	2.94	32.43	163.62	10
	QP	1CA	0.12	0.15	0.09	2.44	2.99	1.97	4.22	1.08	6.18	6.18	32.25	163.06	1
		MCA	0.12	0.15	0.09	2.41	2.99	1.96	2.43	2.48	2.49	2.49	32.30	163.15	2
		MC-GRD	0.11	0.12	0.10	2.41	2.99	1.97	1.99	2.57	3.24	3.24	32.35	163.39	7
		MC-DyP	0.11	0.13	0.09	2.42	3.00	1.97	2.11	2.88	2.23	2.88	32.35	163.36	10
TC2	RND	1CA	0.50	0.63	0.40	1.79	2.22	1.42	6.46	3.07	8.72	8.72	10.99	55.83	2
		MCA	0.52	0.65	0.42	1.83	2.29	1.48	3.36	4.39	4.71	4.71	11.07	55.98	3
		MC-GRD	0.48	0.55	0.43	1.78	2.23	1.43	3.55	4.47	4.76	4.76	11.21	56.47	28
		MC-DyP	0.38	0.47	0.30	1.77	2.22	1.43	3.79	4.11	4.23	4.23	11.21	56.40	54
	QP	1CA	0.52	0.66	0.41	1.79	2.22	1.42	6.13	2.36	8.72	8.72	10.97	55.76	1
		MCA	0.53	0.67	0.43	1.93	2.42	1.56	4.67	5.03	6.39	6.39	11.04	55.88	2
		MC-GRD	0.45	0.51	0.41	1.77	2.22	1.42	3.06	5.49	4.79	5.49	11.15	56.25	23
		MC-DyP	0.35	0.44	0.29	1.78	2.24	1.44	3.09	4.02	3.64	4.02	11.15	56.19	50
TC3	RND	1CA	0.67	0.83	0.53	0.65	0.80	0.52	8.77	5.29	11.58	11.58	2.89	14.28	2
		MCA	0.70	0.86	0.57	0.65	0.80	0.53	11.83	10.75	11.69	11.83	2.93	14.36	2
		MC-GRD	0.57	0.65	0.51	0.65	0.82	0.54	11.45	12.00	13.93	13.93	3.12	14.98	39
		MC-DyP	0.50	0.63	0.41	0.66	0.83	0.54	12.36	14.20	13.39	14.20	3.03	14.65	48
	QP	1CA	0.66	0.83	0.54	0.65	0.80	0.53	12.47	8.23	17.07	17.07	2.90	14.27	1
		MCA	0.60	0.75	0.49	0.63	0.79	0.51	10.36	13.22	11.61	13.22	2.93	14.31	2
		MC-GRD	0.56	0.63	0.50	0.63	0.79	0.51	13.33	15.27	13.58	15.27	3.07	14.80	33
		MC-DyP	0.49	0.60	0.39	0.66	0.82	0.54	10.75	11.05	12.74	12.74	3.03	14.64	51
TC4	RND	1CA	1.36	1.71	1.10	0.81	1.00	0.65	10.62	6.14	12.61	12.61	6.42	32.63	3
		MCA	1.43	1.77	1.16	0.91	1.13	0.76	7.53	7.46	11.73	11.73	6.48	32.75	5
		MC-GRD	1.19	1.35	1.06	0.81	1.01	0.66	9.27	9.26	10.28	10.28	6.86	34.04	61
		MC-DyP	1.02	1.27	0.83	0.81	1.01	0.66	8.08	8.32	10.31	10.31	6.68	33.38	84
	QP	1CA	1.39	1.74	1.11	0.81	1.00	0.65	8.27	5.22	11.46	11.46	6.40	32.62	2
		MCA	1.36	1.69	1.11	0.86	1.08	0.70	9.62	11.52	11.26	11.52	6.45	32.71	4
		MC-GRD	1.20	1.37	1.07	0.80	1.01	0.65	7.11	9.75	8.96	9.75	6.78	33.79	57
		MC-DyP	1.04	1.30	0.85	0.84	1.06	0.68	14.24	14.57	15.43	15.43	6.65	33.24	86
TC5	RND	1CA	3.67	4.61	2.92	1.28	1.59	1.04	6.85	4.12	11.14	11.14	9.32	43.78	4
		MCA	3.46	4.29	2.80	1.34	1.66	1.09	6.49	7.17	6.79	7.17	9.38	43.90	4
		MC-GRD	3.18	3.60	2.84	1.37	1.71	1.12	6.46	7.17	7.18	7.18	10.08	46.13	139
		MC-DyP	2.27	2.84	1.84	1.30	1.62	1.05	6.52	7.41	6.49	7.41	9.81	45.13	148
	QP	1CA	3.59	4.53	2.86	1.34	1.66	1.09	8.21	4.91	13.85	13.85	9.33	43.79	3
		MCA	3.52	4.38	2.86	1.43	1.78	1.16	6.11	6.64	6.39	6.64	9.40	43.92	5
		MC-GRD	3.20	3.63	2.87	1.28	1.60	1.05	8.20	9.62	9.04	9.62	10.01	45.94	116
		MC-DyP	2.19	2.74	1.76	1.31	1.64	1.07	6.15	7.01	6.40	7.01	9.68	44.76	240
TC6	RND	1CA	6.42	8.02	5.17	1.14	1.40	0.92	8.04	5.16	11.45	11.45	29.35	141.97	6
		MCA	6.30	7.83	5.10	1.06	1.33	0.86	8.77	9.52	10.20	10.20	29.42	142.10	8
		MC-GRD	4.94	5.63	4.40	1.05	1.32	0.85	9.88	10.81	9.98	10.81	31.14	147.73	254
		MC-DyP	5.10	6.35	4.14	1.07	1.34	0.87	6.83	8.16	8.26	8.26	30.29	144.71	488
	QP	1CA	6.11	7.64	4.92	1.06	1.33	0.86	7.11	5.03	11.35	11.35	29.32	141.86	4
		MCA	5.96	7.42	4.82	1.04	1.32	0.84	10.36	11.89	10.46	11.89	29.40	142.01	6
		MC-GRD	5.59	6.45	4.93	1.04	1.30	0.84	9.73	11.23	10.35	11.23	31.03	147.34	218
		MC-DyP	5.28	6.59	4.26	1.06	1.33	0.85	7.51	8.39	9.54	9.54	30.12	144.13	421

Skew in a given corner is given as a percentage of corresponding corner delay.

resemble this, we generate the criticality information randomly such that the maximum value on the random number generated remains constant until a certain distance, after which it reduces gradually. Thus, the probability of having a critical path between a IP pair close to each other is higher than having them on the opposite ends of the chip.

3) *Generating IP Pin Assignments:* Clock pin assignment is done in two ways to produce two flavors of the test-cases. First, we use the pin assignment step of Section III to get one set of test-cases. Next, we randomly pick the clock pin location for all IPs to get a second set of test-cases with identical floorplan as first set, the only difference being the clock pin locations. Comparison of results between these two sets will tell us the effectiveness of our clock pin assignment algorithm.

4) *Generating IP CTS Data:* The final step in test-case generation is to mimic the IP-level CTS done on the different IPs. This should be done in such a way as to account for the potential differences in the clock trees in the IPs due to the difference in the individuals/teams, methodology, cell

TABLE III
AVERAGE VALUES FOR DIFFERENT METRICS FOR SIX TEST-CASES SHOWN IN TABLE II ALONG WITH AVERAGE AND NORMALIZED RESULTS OF ALL THE 100 TEST-CASES USED

TC	PAM	CCTS Algorithm	Divergence (μs)			Average Max Delay (ns)			Skew (% of Delay)				BA (nm^2)	WL (μm)	CPU
			NN	SS	FF	NN	SS	FF	NN	SS	FF	Worst	X 1e6	X 1e6	(s)
Avg. (6 TCs)	RND	1CA	2.12	2.66	1.70	1.35	1.67	1.09	6.57	3.35	9.31	9.31	15.21	75.29	3
		MCA	2.09	2.59	1.69	1.37	1.70	1.12	5.32	5.80	6.38	6.38	15.27	75.40	4
		MC-GRD	1.74	1.98	1.56	1.35	1.68	1.10	5.65	6.18	6.47	6.47	15.81	77.17	88
		MC-DyP	1.57	1.95	1.27	1.34	1.67	1.09	5.34	6.05	5.90	6.05	15.57	76.32	139
	QP	1CA	2.07	2.59	1.66	1.35	1.67	1.09	6.73	3.49	10.06	10.06	15.20	75.22	2
		MCA	2.02	2.51	1.63	1.39	1.73	1.13	5.89	6.68	6.62	6.68	15.25	75.33	4
		MC-GRD	1.85	2.12	1.64	1.33	1.66	1.08	5.61	7.17	6.80	7.17	15.73	76.92	76
		MC-DyP	1.58	1.97	1.27	1.35	1.69	1.10	5.59	6.36	6.32	6.36	15.50	76.06	143
Avg. (100 TCs)	RND	1CA	1.85	2.32	1.48	1.37	1.69	1.10	6.67	3.63	9.95	9.95	13.68	68.57	3
		MCA	1.87	2.32	1.51	1.41	1.75	1.14	5.47	6.25	6.53	6.53	13.76	68.77	4
		MC-GRD	1.57	1.79	1.40	1.36	1.70	1.10	5.45	6.44	6.53	6.53	14.16	70.03	121
		MC-DyP	1.35	1.67	1.09	1.36	1.70	1.11	5.90	6.52	6.93	6.93	14.00	69.44	133
	QP	1CA	1.83	2.30	1.46	1.37	1.70	1.11	6.95	3.72	10.33	10.33	13.67	68.51	3
		MCA	1.84	2.29	1.49	1.41	1.76	1.14	5.69	6.41	6.60	6.60	13.74	68.65	4
		MC-GRD	1.57	1.79	1.40	1.36	1.70	1.10	5.45	6.44	6.53	6.53	14.16	70.03	119
		MC-DyP	1.30	1.62	1.06	1.37	1.71	1.11	6.37	7.16	7.28	7.28	13.95	69.28	121
% impr. w.r.t.	RND	MC-DyP	27.1	27.8	26.2	−0.19	1.15	−1.65	40.6	34.4	30.3	40.6	−2.33	−1.27	–
1CA RND	QP	MC-DyP	29.5	30.1	28.6	−0.46	0.90	−1.90	35.9	28.0	26.8	35.9	−1.97	−1.03	–

libraries, and so on. We accomplish this by randomly selecting the clock sink density for each IP within a pre-selected range, thereby selecting the number of sinks. This number is rounded off to the nearest power of 2 and the number of H-tree levels to drive these flip–flops is obtained. Next, we select a random slew range from a tight range of valid slew. Finally, we recursively choose a random buffer size and use that to drive the H-Tree in a bottom-up fashion to meet the slew limit. Because of the use of different buffer sizes and different slew limits, the above procedure mimics the situation that arises commonly in most SoC designs. Table I shows some of the key parameters for our test-case generation script.

B. Experimental Setup and Results

We use the 65 nm model cards from [23] for generation of delays across corners. We use three device corners (NN, FF, SS) to generate the nominal, fast, and slow corners. For simplicity, we did not consider other global variations like voltage, temperature, and interconnect. As more and more corners are added, the single-corner CTS will be even worse compared to our multicorner algorithms. In other words, the skew reduction results that we are presenting here are very conservative. Also, adding these variation effects will not change the nature of results on clock divergence reduction as it uses only the nominal corner delay as a guidance for minimizing the divergence. Our buffer library consisted of 10 buffers with different sizes (transistor widths) ranging from 10 to 100 times the minimum feature size.

We generate 100 random test-cases with unique floorplans and different sizes within the ranges given in Table I. Each of these test-cases will have two flavors depending on the pin assignment strategy used—either random pin placement or QP pin placement. The four algorithms described in Section V are run on both sets of test-cases generated. Since the two test-case sets are identical in all manner other than the pin locations, a direct comparison of the results from these two sets will indicate the impact of clock pin placement method. Also, we run each of the four CCTS algorithms on all test-cases irrespective of their clock pin placement method. This will be used to compare the relative effectiveness of the four CCTS algorithms.

Table II gives detailed results of six representative test-cases out of the 100 test-cases we have generated. Table III gives the average results for the six test-cases used in Table II along with average results of all the 100 test-cases generated. The last two rows of Table III give the percentage improvement of the different parameters with respect to the baseline values from the single-corner random pin assignment (1CA RND) method. A positive number in these rows implies a reduction in value. Please note that we have used the worst values of the 1CA-RND skew to normalize all the other values in these rows. Some of the acronyms used in Tables II and III are explained next. TC denotes the Test Case for the results. PAM denotes the pin assignment method used in the test-case. This can either be the quadratic-programming (QP) based method or random pin assignment method (RND). The four CCTS algorithms described earlier are abbreviated as: singe-corner approach (1CA), multicorner approach (MCA), multicorner greedy algorithm (MC-GRD), and multicorner dynamic programming based algorithm (MC-DyP). The divergence values given are weighted sum of clock divergence between all IP pairs. The weights are proportional to the timing criticality of all the paths between the IP pairs. Please note that in Table II, all metrics except skew are absolute values. Skew in a given corner is given as a percentage of the delay in the corresponding corner. Since the delay values between the slowest corner (SS) and fastest corner (FF) can be quite different, we believe normalizing the absolute skew in each corner by the corresponding delay will tell us how significant the skew is in a given corner. We call this skew value as

TABLE IV
CHARACTERISTICS OF THE 100 RANDOM TEST-CASES GENERATED WITH A REPRESENTATIVE SIX

TC	No. of IPs	No. of Flops	X_Size (cm)	Y_Size (cm)	Aspect Ratio	Max_IP_Del (ns)	Min_IP_Del (ns)
TC1	14	589 824	2.02	2.66	0.76	2.41	1.08
TC2	30	184 320	1.63	1.76	0.93	1.77	0.27
TC3	48	48512	0.98	0.91	1.07	0.63	0.11
TC4	63	119 296	1.48	1.21	1.22	0.79	0.14
TC5	90	146 432	1.31	1.82	0.72	1.29	0.15
TC6	126	521 216	2.67	2.28	1.17	1.04	0.34
Avg(6)	62	268 267	1.68	1.77	0.98	1.32	0.35
Avg(100)	56	279 777	1.74	1.84	0.96	1.48	0.35

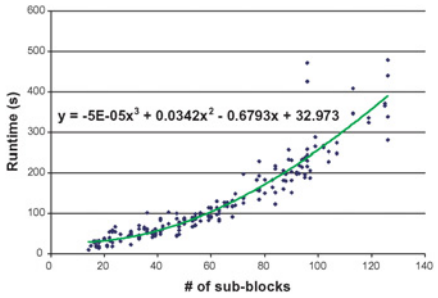

Fig. 12. Runtime of the dynamic programming based CCTS algorithm for all 100 test-cases.

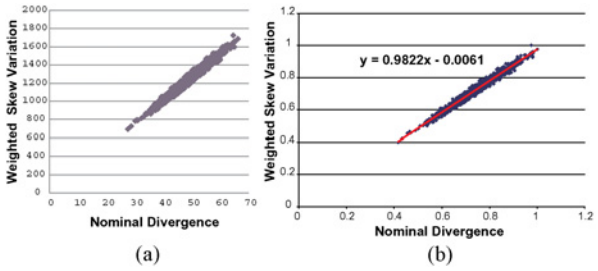

Fig. 13. Divergence and skew variation are directly correlated. (a) Absolute values. (b) Normalized values.

normalized skew. Measuring normalized skew will also help us determine the effectiveness of the multicorner approach compared to the single corner approach.

1) *Runtime:* Fig. 12 shows how the runtime of the dynamic programming based algorithm scales with respect to the number of IPs in different test-cases. The figure also shows the trendline for the runtime data, which shows that the runtime scales with complexity of $O(n^3)$ approximately.

2) *Validation of Divergence:* To demonstrate that reducing divergence is identical to reducing skew variation, we did Monte Carlo simulations on a few random test-cases. For each test-case, we also have the random weights to give the pair wise criticality of the timing paths between the IPs. Fig. 13(a) shows the results of this experiment where we have plotted the nominal corner divergence against the weighted sum of Monte Carlo skew variation in the nominal corner. Skew variation is defined as the extra skew in addition to nominal skew due to variation effects. We have assumed that both buffer and interconnect delays can vary by 10% in this experiment. For each run, we obtain the skew variation for each IP pair and use the random weights to obtain a weighted sum of skew variation. This will ensure that we measure the impact variation on skew between all IP pairs instead of just measuring the worst case skew. As we can see from Fig. 13(b), which is a normalized version of Fig. 13(a), there is almost a one-to-one correlation between divergence and skew variation since the slope of the line in Fig. 13(b) is very close to 45°. In other words, reducing divergence by x% implies a reduction in skew variation of x%. This proves the validity of our divergence metric.

C. Discussions

Based on the results in Tables II and III, we can observe the following observations.

1) From the last row of Table III, we see that the MC-DyP algorithm with QP pin assignment reduces divergence by an average of around 30% [3] compared to the single-corner approach with random pin placement (1CA RND) with small impact on delay, buffer area and wirelength.
2) From the last two rows of Table III, we see that using QP pin assignment reduces the divergence by 2% on average compared to the random pin assignment. Though the nominal global skew increases very slightly (by 0.35%) with the QP pin assignment (comparing RND MC-DyP and QP MC-DyP), the overall impact of QP pin placement is still beneficial. The reason is that the nominal skew is just the global skew. So essentially, the tradeoff is between reducing clock divergence by 2% for *all* end point pairs and a very small increase in nominal skew between *one* pair of end points.
3) Comparing the worst values of normalized skews in the single corner approach with all three multicorner approaches, the multicorner methods reduces the worst case normalized skew across the three corners. For example, the single corner method using QP pin assignment results in the worst normalized skew of 10.33% compared to 7.28% for the dynamic programming approach using QP pin assignment.
4) The above reductions in clock divergence and worst normalized skew comes at an average cost of 2% buffer area and 1% wire-length.

VIII. CONCLUSION

In this paper, we addressed the chip-level CTS problem for complex SoC designs. Experimental results on several test-

[3]Average of divergence reduction in three corners.

cases showed that our algorithms are effective in simultaneous reduction of multicorner skew and clock divergence between critical IP pairs. Overall, our algorithms can achieve 30% average reduction in the clock path divergence and increased multicorner skew robustness at the cost of 2% increase in buffer area and 1% increase in wirelength.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions.

REFERENCES

[1] R. Rajsuman, *System-on-a-Chip: Design and Test*. Boston, MA: Artech House Publishers, 2000.
[2] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*, 3rd ed. Dordrecht, The Netherlands: Kluwer, 2002, p. 292.
[3] S. Agarwala, P. Wiley, A. Rajagopal, A. Hill, R. Damodaran, L. Nardini, T. Anderson, S. Mullinnix, J. Flores, H. Yue, A. Chachad, J. Apostol, K. Castille, U. Narasimha, T. Wolf, N. S. Nagaraj, M. Krishnan, L. Nguyen, T. Kroeger, M. Gill, P. Groves, B. Webster, J. Graber, and C. Karlovich, "A 800 MHz system-on-chip for wireless infrastructure applications," in *Proc. VLSI Des.*, 2004, pp. 381–389.
[4] S. Agarwala, A. Rajagopal, A. Hill, M. Joshi, S. Mullinnix, T. Anderson, R. Damodaran, L. Nardini, P. Wiley, P. Groves, J. Apostol, M. Gill, J. Flores, A. Chachad, A. Hales, K. Chirca, K. Panda, R. Venkatasubramanian, P. Eyres, R. Veiamuri, A. Rajaram, M. Krishnan, J. Nelson, J. Frade, M. Rahman, N. Mahmood, U. Narasimha, S. Sinha, S. Krishnan, W. Webster, B. Due, S. Moharii, N. Common, R. Nair, R. Ramanujam, and M. Ryan, "A 65 nm C64x+ multi-core DSP platform for communications infrastructure," in *Proc. IEEE ISSCC*, Feb. 2007, pp. 262–264.
[5] V. Wason, R. Murgai, and W. W. Walker, "An efficient uncertainty and skew-aware methodology for clock tree synthesis and analysis," in *Proc. VLSI Design*, 2007. pp. 271–277.
[6] J. Rosenfeld and E. G. Friedman, "Design methodology for global resonant H-tree clock distribution networks," *IEEE Trans. Very Large Scale Integr.*, vol. 15, no. 2, pp. 135–148, Feb. 2007.
[7] A. Kapoor, N. Jayakumar, and S. P. Khatri, "A novel clock distribution and dynamic de-skewing methodology," in *Proc. ICCAD*, 2004, pp. 626–631.
[8] P. J. Restle, T. G. McNamara, D. A. Webber, P. J. Camporese, K. F. Eng, K. A. Jenkins, D. H. Allen, M. J. Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovick, B. L. Krauter, and B. D. McCredie, "A clock distribution network for microprocessors," *J. Solid-State Circuits*, vol. 36, no. 5, pp. 792–799, May 2001.
[9] S. A. Butt, S. Schmermbeck, J. Rosenthal, A. Pratsch, and E. Schmidt, "System level clock tree synthesis for power optimization," in *Proc. DATE*, 2007, pp. 1677–1682.
[10] U. Padmanabhan, J. M. Wang, and J. Hu, "Robust clock tree routing in the presence of process variations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 8, pp. 1385–1397, Aug. 2008.
[11] A. Rajaram, J. Hu, and R. Mahapatra, "Reducing clock skew variability via cross links," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 1176–1182, Jun. 2006.
[12] A. Rajaram and D. Z. Pan, "Robust chip-level clock tree synthesis for SOC designs," in *Proc. IEEE/ACM DAC*, Jun. 2008, pp. 720–723.
[13] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," in *Proc. DAC*, 1993, pp. 612–616.
[14] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese, and A. B. Kahng, "Zero skew clock routing with minimum wire-length," *IEEE Trans. Circuits Syst. II: Analog Digital Signal Process.*, vol. 39, no. 11, pp. 799–814, Nov. 1992.
[15] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, "Bounded-skew clock and Steiner routing," *ACM TODAES*, vol. 3, no. 3, pp. 341–388, Jul. 1998.
[16] R.-S. Tsay, "Exact zero skew," in *Proc. IEEE/ACM ICCAD*, Nov. 1991, pp. 336–339.
[17] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proc. IEEE*, vol. 89, no. 5, pp. 665–692, May 2001.
[18] *MATLAB*. Available: http://www.mathworks.com/products/optimization
[19] J. Jiang, "Pin allocation for clock routing," in *Proc. 2nd Int. Conf. ASIC*, Oct. 1996, pp. 35–38.
[20] R. Chaturvedi and J. Hu, "An efficient merging scheme for prescribed skew clock routing," *IEEE Trans. Very Large Scale Integr.*, vol. 13, no. 6, pp. 750–754, Jun. 2005.
[21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2009.
[22] A. K. Rajaram. Available: http://www.cerc.utexas.edu/$\sim$anandr/DAC08_CCTS.ppt
[23] Arizona State University. Available: http://www.eas.asu.edu/∼ptm

Anand Rajaram (S'04–M'09) received the B.E. degree in electrical and electronics engineering from Anna University, Chennai, India, the M.S. degree in computer engineering from Texas A&M University, College Station, in 2004, and the Ph.D. degree in computer engineering from the University of Texas, Austin, in 2008.

From 2004 to 2008, he was with the Dallas DSP Group, Texas Instruments, Dallas, where he worked on high speed clock network synthesis and analysis on high-performance DSP chips. Since 2008, he has been with Magma Design Automation, Austin, working on various aspects of physical design automation. He has published more than 18 refereed papers in international conferences and journals. His current research interests include variation-aware physical design and clock network synthesis and analysis.

Dr. Rajaram's papers at the Design Automation Conference in 2004 and the Asia and South Pacific Design Automation Conference in 2008 were nominated for Best Paper Awards and his paper at the Design, Automation and Test in Europe in 2009 received the Best IP Paper Award.

David Pan (S'97–M'00–SM'06) received the Ph.D. degree in computer science from the University of California, Los Angeles, in 2000.

From 2000 to 2003, he was a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY. He is currently an Associate Professor and the Director of the UT Design Automation Laboratory, Department of Electrical and Computer Engineering, University of Texas, Austin. He has published over 120 refereed papers in international conferences and journals, and is the holder of seven U.S. patents. His current research interests include nanometer very large scale integration (VLSI) physical design, design for manufacturing, vertical integration of technology, design and architecture, and design/computer-aided design (CAD) for emerging technologies.

Dr. Pan has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS (TCAD), IEEE TRANSACTIONS ON VERY LARGE SSCALE INTEGRATION SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-PART I, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-PART II, IEEE CAS SOCIETY NEWSLETTER, and the *Journal of Computer Science and Technology*. He was a Guest Editor of the TCAD Special Section on the International Symposium on Physical Design in 2007 and 2008. He serves as the Chair of the IEEE CANDE Committee and the ACM/SIGDA Physical Design Technical Committee. He is on the Design Technology Working Group of the International Technology Roadmap for Semiconductor. He has served on the technical program committees of major VLSI/CAD conferences, including ASPDAC (Topic Chair), DAC, DATE, ICCAD, ISPD (Program Chair), ISLPED (Exhibits Chair), ISCAS (CAD Track Chair), ISQED (Topic Chair), GLSVLSI (Publicity Chair), SLIP (Publication Chair), ACISC (Program Co-Chair), ICICDT (Award Chair), and VLSI-DAT (EDA Track Chair). He was the General Chair of ISPD 2008 and ACISC 2009. He is a member of the Technical Advisory Board of Pyxis Technology, Inc. He has received a number of awards for his research contributions and professional services, including the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award thrice in 2000 and 2008, the IBM Faculty Award thrice from 2004 to 2006, the UCLA Engineering Distinguished Young Alumnus Award in 2009, the Best Paper Award from ASPDAC in 2010, the Best Interactive Presentation Award from DATE in 2010, the Best Student Paper Award from ICICDT in 2009, the IBM Research Bravo Award in 2003, the SRC Techcon Best Paper in Session Award in 1998 and 2007, the Dimitris Chorafas Foundation Research Award in 2000, the ISPD Routing Contest Awards in 2007, the eASIC Placement Contest Grand Prize in 2009, five Best Paper Award Nominations (from ASPDAC, DAC, ICCAD, ISPD), and the ACM Recognition of Service Award in 2007 and 2008. He was an IEEE CAS Society Distinguished Lecturer from 2008 to 2009.