# Variation Tolerant Buffered Clock Network Synthesis with Cross Links


## ABSTRACT

Clock network synthesis is a key step in the ultra deep submicron (UDSM) VLSI Designs. Most existing clock network synthesis algorithms are designed for nominal operating condition, which are insufficient to tolerate the growing variability from process, voltage and temperature (PVT) variations. In this paper, we propose a *unified* algorithm for synthesizing a robust, variation tolerant, buffered clock network with cross link insertion. We explicitly consider clock signal slew and resistive shielding effect for our variation-tolerant, *balanced* buffered clocks tree construction without sacrificing skew, buffer/wire area, and run time. We further reduce the clock skew variability with cross link insertion to our balanced buffered clock trees, which work seamlessly with cross-link methodology. Very promising experimental results are obtained on standard benchmarks, compared to previous state-of-the-art algorithms.


## 1. INTRODUCTION

Clock distribution networks (CDNs) are of great importance in any synchronous VLSI chip because the pace of almost every data transfer is determined by the clock signal. A high quality CDN is vital for obtaining good circuit performance, reduced power and high timing yield in the UDSM technologies. As one of the largest and fastest switching nets in any design, the CDN has a tremendous influence on the over all performance of the chip [2]. Realizing this importance, numerous works have been done towards solving the problem of clock network synthesis [5–18]. In the majority of these works, only traditional parameters like skew, wire/buffer area and power are considered. However, in the sub-100 nm technologies, variation effects like manufacturing variations [3, 20], temperature changes and power supply noise [4] are becoming more and more significant and therefore, it is of vital importance to address these issues in the clock network, which is especially sensitive to variations.

In a typical clock network, the variation in device parameters like gate length and oxide thickness is an important source for the unwanted skew in the UDSM technologies [21]. The interconnect variation might also account for up to 25% of the total clock skew variability in high-performance designs [22]. The contribution of interconnect variation towards unwanted skew is likely to increase as the technology scales further. However, many of the existing algorithms, including the recent ones like [15–17] do not consider such variation effects during clock tree synthesis. These algorithms typically construct the clock tree for the nominal values of device/interconnect parameters to achieve the target clock skews. The main drawback of such an approach is that even if clock skew constraints are met at design time (for nominal values of device/interconnect parameters), PVT variations can introduce unwanted clock skew during the chip fabrication, thereby affecting performance and timing yield. Recently proposed link-based clock network [1] has been shown to be very tolerant to PVT variations. However, [1] addresses only unbuffered clock networks, which cannot be used in most practical cases.

In this work, we propose an efficient algorithm for synthesizing a robust, variation tolerant buffered clock network with cross link insertion. The important contributions of this work are:

- We propose an efficient algorithm to explicitly consider clock signal slew and resistive shielding effects during the bottom-up topology construction. It helps us to overcome the inaccuracy of the simple Elmore delay model, as used in most clock tree synthesis algorithms.
- We propose a unified variation-tolerant algorithm to build a balanced buffered clock network along with wire length minimization.
- We further reduce the clock skew variability with cross link insertion to our balanced buffered clock trees. It may be noted here that adding links in a buffered clock tree is a non-trivial problem. This has been explained in section 5.

Monte Carlo simulations in SPICE using 90 nm technology parameters show that our algorithm can reduce the average skew due to variation effects by as much as 50% with no penalty in resources or runtime.

## 2. BACKGROUND AND MOTIVATION

In this section, we will briefly review existing buffered clock tree synthesis algorithms in the literature. Then, we

will point out their drawbacks, and motivations for this work.

## 2.1 Brief review of existing works

One of the pioneering algorithms for clock routing was proposed in [5], in which a zero skew clock routing was obtained by recursively merging a pair of zero skew subtrees until a single clock tree is obtained. The zero-skew principle in [5] was extended in the DME algorithm [6] for wire length minimization. However, [5, 6] addressed only the problem of an unbuffered clock tree. The problem of constructing a zero skew buffered clock tree under Elmore delay model was solved in [7, 8]. The optimal clock buffering for a given topology and buffer library was solved in [9]. A heuristic for synthesizing a low-power buffered clock tree using the Elmore delay model was proposed in [10]. Buffer and wire sizing were done so as to reduce power and maintain the zero skew property under Elmore delay. None of the above clock tree synthesis algorithms consider clock signal slew *during* the synthesis of the clock tree.

To our best knowledge, [11] was the first work that explicitly considered slew during buffer insertion. But it assumes a *given* unbuffered clock tree, which may result in very suboptimal solution compared to simultaneous buffering and clock routing [7]. In [18], a SPICE based time domain based buffer/wire sizing has been proposed which results in greatly reduced skew in SPICE. To our best knowledge, [18] is the only work that aims to reduce the actual clock skew in SPICE. However, since it directly uses SPICE for tuning the clock network, the runtime may be prohibitive.

In [12], the important problem of clock buffer load imbalance is addressed. In the previous algorithms like [7], different clock buffers at a given level from the clock source may drive different loads. The methodology in [12] attempts to solve this problem by using a clustering approach. But such a clustering and load balancing approach usually results in excessive wire length due to wire snaking when the two clusters to be merged do not have similar target delays

The effect of interconnect width variation on the clock skew has been addressed in the algorithms of [1, 13, 14]. However, the effect of device variation has not been addressed. In [15, 16], the problem for optimal buffer/wire sizing in clock network has been studied under the Elmore delay model. In [17], a new merging scheme has been proposed for prescribed skews which usually results in considerably less wirelength compared to the other algorithms. However, this algorithm results in highly unbalanced clock structure.

The balanced clock structure issue has been addressed in works like [7, 12]. But both of them perform one form of load balancing or the other, they result in excessive total wire lengths when compared to the results of algorithm in [17]. But the algorithm in [17] constructs a highly unbalanced clock tree, inserting buffers only for meeting slew requirements.

## 2.2 Motivations for this work

**Explicit slew consideration during CTS**: As discussed in the previous section, most of the existing buffered CTS algorithms use Elmore delay. However, Elmore delay is inaccurate for the DSM technologies mainly because it ignores the resistive shielding effect [19]. Oftentimes, a buffered clock tree built using Elmore delay results in considerable skews in SPICE simulations [18]. Except in [11], the clock signal slew is not considered during the clock tree synthesis stage in most algorithms. A few algorithms like [17] use an upper bound of capacitance that can be driven by a buffer and insert buffers whenever the bound is exceeded. However, this may result in differential buffer loading and extra skews in SPICE simulations. Hence, a systematic approach to addressing signal slew of clock signals is necessary.

**Balanced nature of clock tree for variation tolerance**: We define a balanced clock tree as one in which identical buffers are inserted at a given level from the clock source. Moreover, in a balanced buffered clock tree, the number of buffer levels from the clock source to each clock sink will be the same. Figure 1 (a) shows an example of unbalanced clock tree where the number of buffers from the clock source S to the sinks A and B is two while for sinks C and D it is one. However, in the case of figure 1 (b), all the sinks have equal number of buffers starting from the clock source. This makes it a balanced clock tree, assuming that the sizes of buffers B1 and B2 are the same.

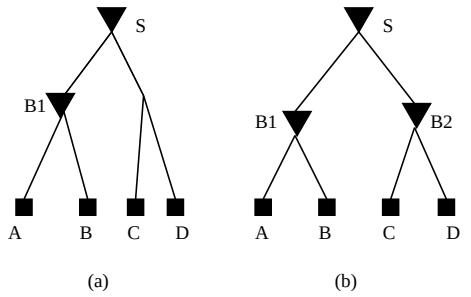


**Figure 1: (a) An example of an unbalanced buffered clock tree; (b) An example of a balanced buffered clock tree.**

A key advantage in having a balanced clock structure is that it will be much more tolerant to variation. For example, if the nominal delays from source to sinks in both figure 1 (a) and (b) is 100 ps and if the nominal gate delay is 20ps. Under nominal conditions, both figure 1 (a) and (b) will have identical skews. However, when variational effects become more and more severe in nanometer designs, the scaling of the device delays and interconnect delays need not match. For example, due to certain change in operating temperature or voltage, the device delay doubles and interconnect delay becomes one half of the nominal values. Then the balanced structure is much more tolerant to the variation.

In the UDSM technologies, it is becoming increasingly difficult to capture all the different variation effects. This further motivates us to use the balanced clock structure to improve tolerance to variations. Many recent clock tree synthesis algorithms like [15, 16, 18] use buffers of different sizes and tune them in such a way that the skew and delay targets are met at the nominal values of device and interconnect parameters. However, due to the PVT variations, significant skew can be generated in such clock trees.

**Link insertion**: In [1], a non-tree clock network with high tolerance to skew variation effects has been proposed. The non-tree is obtained by inserting cross links in a given clock tree at suitable places. It has been shown in [1] that such a non-tree is significantly more tolerant to variation

effects than a regular tree, while the increase in wirelength consumption is very small. However, the methodology proposed in [1] for link insertion is applicable only for unbuffered clock trees. As a result, the practical applicability of this clock tree is reduced. In this work, we address the problem of link addition for buffered clock trees.

The motivation of this work is to consider the above issues in a unified variation-tolerant CDN framework.

# 3. METHODOLOGY FOR BACKWARD PROPAGATION OF SLEW

In this section, we will review the concept of iterative delay evaluation of [19]. The methods of [19] is mainly for delay analysis and cannot be directly applied for clock tree synthesis. To overcome this drawback, we propose our algorithm for backward slew propagation which can be applied during clock tree synthesis.

## 3.1 Iterative delay and slew evaluation

Ideally, we would like to have a delay evaluation procedure that is as efficient and elegant as Elmore delay while accounting for resistive shielding and signal slew effects. The iterative delay estimation procedure of [19] is such a delay model, used in IBM's physical design closure tool. The procedure explicitly considers the signal slew in delay evaluation and accounts for the interdependence between the input signal slew of a node and the *effective* load seen by the node. However, the procedure is mainly for delay evaluation. In this paper, we extend it for the purpose of clock tree synthesis by introducing the notion of *required slew* similar to the concept of *required skew*.

Consider the figure 2 of a simple RC network connecting nodes $v$ and $a$. An input ramp voltage with a signal transition time of $t_v$ is applied at the node $v$. The transition time at the output node of the RC segment, namely node $a$ is given by $t_a$. According to Elmore delay, the total down stream capacitance *seen* by the node $v$ is $C$. However, because of the resistive shielding effect of the resistance $R$, only a fraction of the capacitance $C$ is actually seen by the node $v$, which is usually referred to by the name *effective capacitance* [19]. According to [19], the value of this effective capacitance is give as:

$$C_{eff} = K * C \tag{1}$$

where K is the scaling factor defined as:

$$K = 1 - 2x(1 - e^{-\frac{1}{2x}}), \quad where \quad x = \frac{RC}{t_v} \tag{2}$$

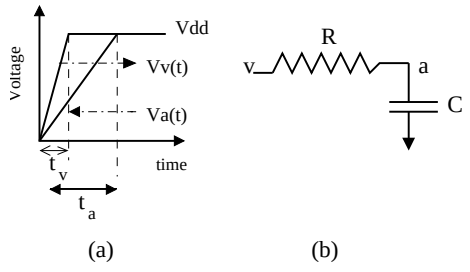


**Figure 2: (a) Definitions of transition times for nodes $v$ and $a$. (b) A simple example of RC network.**

It should noted that the value of the effective capacitance seen by node $v$ and the slew rate at $v$ are interdependent. The output slew rate of the CMOS buffer depends on both the input slew *and* the load capacitance [19]. From equations (1) and (2), the effective load capacitance seen by the buffer output depends on the slew at the buffer output. This factor introduces a chicken-egg problem which is addressed in [19] using an iterative delay evaluation technique.

## 3.2 Backward propagation of slew

In order to consider the node slews during the clock tree synthesis, we need to calculate the signal slew rate during the bottom-up topology generation phase of the DME [6, 17] algorithm. However, by definition, the slew rate at a child node can be calculated only when the slew rate at the parent node is known. For example, in figure 2, the slew rate at node $a$ can be obtained only when the slew rate at node $v$ is known. An efficient method for obtaining the transition times at the nodes of the clock tree for a given transition time at the source node has been proposed in [19]. Considering the figure 2, the transition time at node $v$ is given as $t_v$. Given $t_v$ and the R, C values, the transition time time at node $a$ can be obtained using the method of [19] as:

$$t_a = \frac{t_v}{1 - x(1 - e^{-\frac{1}{x}})}, \quad where \quad x = \frac{RC}{t_v} \tag{3}$$

In order to consider the slew *during* clock tree synthesis, we would like to get an inverse equation of 3. That is, we would like to get the value of $t_v$ for a given value of $t_a$. Such an inverse expression will enable us to consider slew during the bottom up phase of clock tree synthesis. Such an inverse expression can be obtained as follows: define a new quantity called $y$ and using 3, we have:

$$y = \frac{RC}{t_a} = \frac{RC(1 - x(1 - e^{-\frac{1}{x}}))}{t_v}$$

which can be simplified to

$$y = x(1 - x(1 - e^{-\frac{1}{x}})) \tag{4}$$

The plot of equation 4 is shown in figure 3. As it can be seen from the plot, the value of $y$ reaches a saturation point after the value of $x$ reaches a value of roughly 20. The saturation value of $y$ is 0.5, which can also be verified by applying the Taylor series approximation for the term $e^{-\frac{1}{x}}$ as $1 - \frac{1}{x} + \frac{1}{2x^2}$. Using this approximation in equation 4 will reduce the value of $y$ to 0.5. A key use of the above observation is that for a given value of $x$, there is an unique value of $y$ and vice versa. Thus, when we are given the *required* slew value at output node, we can obtain the value of $y$, which can be used to uniquely determine the value of $x$, which in turn can be used to obtain the *required* input slew. In other words, if we have a slew requirement at the child node $a$ in figure 2, using that we can uniquely obtain the *required* slew value at the parent node $v$. This technique can be used to build a buffered clock tree with simultaneous slew considerations in a bottom-up fashion.

# 4. VARIATION TOLERANT BALANCED CLOCK TREE SYNTHESIS

In this section, we propose our balanced buffered clock tree synthesis algorithm. First, we will consider the problem

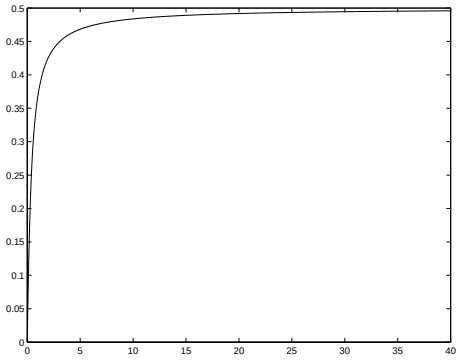

**Figure 3: Plot of x (ratio of RC and input slew) versus y (ratio of RC and output slew)**

of merging two subtrees using the backward slew propagation algorithm of the previous section. Then we introduce our novel merging scheme which guarantees the construction of a perfectly balanced buffered clock tree while simultaneously reducing the wirelength.

The high level framework of our algorithm is similar to the DME based algorithms like [7,17] in which the first step is the topology generation phase in which different subtrees are merged recursively based on a merging cost. After all the subtrees are merged into a single tree, a top down embedding is done to finalize the locations of the clock tree nodes.

## 4.1 Subtree merging with backward slew propagation

Consider figure 4 in which two subtrees $T_i$ and $T_j$ (rooted at nodes $i$ and $j$ respectively) are to be merged to form a new subtree $T_p$ with node $p$ as the root. In the traditional merging, the lengths of segments $l_{p,i}$ and $l_{p,j}$ are determined in such a way that the Elmore delay from $v$ to the sinks of both $T_i$ and $T_j$ are identical. During this step, the entire downstream capacitance at nodes $i$ and $j$ are considered. However, the delay evaluation method of [19] considers only the *effective* capacitance at the subtrees $T_i$ and $T_j$ while determining the edge lengths. The delay from node $p$ to nodes $i$ and $j$ are given as [19]:

$$D(p,i) = \frac{1}{2}rcl_{p,i}^2 + rl_{p,i}C_{eff1} \quad (5)$$

$$D(p,j) = \frac{1}{2}rcl_{p,j}^2 + rl_{p,j}C_{eff2}$$

where, $r$ and $c$ are the unit length resistance and capacitance, respectively. $C_{eff1}$ and $C_{eff2}$ are the effective downstream capacitances of nodes $i$ and $j$ respectively. It may be noted that for clock sinks, the value of effective capacitance is equal to the load capacitance.

In order to balance the *effective delays* of the two subtrees, the following equation must be satisfied:

$$D_i + D(p,i) = D_j + D(p,j) \quad (6)$$

where $D_i$ and $D_j$ are the delays from nodes $i$ and $j$ to their respective sink nodes. The edge lengths $l_{p,i}$ and $l_{p,j}$ can be obtained by solving equation 6 with the condition that $l_{p,i} + l_{p,i} = L$, where $L$ is the Manhatten distance between the nodes (or the merging segment of the nodes) $i$ and $j$. Wire snaking can be used to match the delays if wirelengths greater than $L$ is required [17]. Once the appropriate segment lengths have determined, the *required slew* at the parent node $p$ can be calculated using equations (3) to (4).

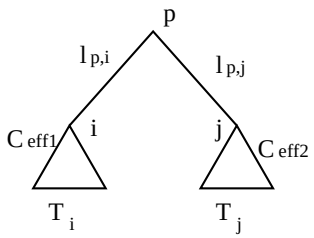


**Figure 4: An example of subtree merger using effective downstream capacitance**

| **Procedure:** $FindSlew(T_p)$ |
|---|
| **Input:** A subtree rooted at node $p$<br>**Output:** The signal transition time limit at $p$. |
| 1. If $p$ is a sink<br>$t_p$ = Transition time limit set by user.<br>return.<br>2. $i = LeftChild(p); j = RightChild(p)$.<br>3. $t_i, t_j \leftarrow$ Transition time limit at nodes $i$ and $j$.<br>4. $R1 = rl_{p,i}; R2 = rl_{p,j}$.<br>5. $C1 = C_{eff1} + 0.5cl_{p,i}; C2 = C_{eff2} + 0.5cl_{p,j}$.<br>6. $y1 = \frac{R1C1}{t_i}; y2 = \frac{R2C2}{t_j}$ (Similar to eqn.2)<br>7. For $y1$ and $y2$, obtain the corresponding unique values of $x1$ and $x2$ using eqn.(4).<br>8. Using $x1$ and $x2$, obtain transition time limits at node $p$ w.r.t nodes $i$ and $j$ as:<br>$t_p^i = \frac{R1C1}{x1}; t_p^j = \frac{R2C2}{x2}$<br>9. return $min(t_p^i, t_p^j)$ |

**Figure 5: Procedure to evaluate the signal transition values of a node given the transition values of the child nodes.**

Figure 5 explains this step in detail. A point to be noted regarding bottom-up transition time limit propagation is that, during merging of two subtrees with different transition time limits, two independent transition limits (one for each child node) can be obtained for the new root $p$, denoted by $t_p^i$ and $t_p^j$ in the figure 5. Since the transition time limits are defined as the maximum signal rise time acceptable at a particular node, we pick only the tighter requirement of the two. Also, slecting the lesser transition time might impact the zero skew property within the subtree that has bigger transition time. However the effect of this is minimal based on our experimental experience.

Once the *required slew* information at the root node $p$ is available, the effective downstream capacitance at node $p$ can also be calculated as demonstrated in figure 6.

Thus, using the algorithms of figures 5 and 6, we can merge a give pair of subtrees and obtain the values of slew and effective downstream capacitance of the new subtree. In order for this method to be applied in a recursive fashion, the slew requirements at the clock sink nodes must be predefined by the user. This can be used during the bottom-up clock tree construction as shown in the next section.

## 4.2 Balanced CTS algorithm

As discussed in section 2.2, one of the key disadvantages with some existing algorithms is the difficulty in getting a balanced clock tree without a wirelength penalty. We propose to address this key problem using a novel merging

| **Procedure:** $FindEffectiveCapacitance(T_p)$ |
|---|
| **Input:** A subtree rooted at node $p$<br>**Output:** The effective downstream capacitance at node $p$. |
| 1. If $p$ is a sink<br>$C_{eff}$ = Sink load capacitance<br>return.<br>2. $i = LeftChild(p); j = RightChild(p)$<br>3. $C_{eff1}, C_{eff2} \leftarrow$ Effective downstream capacitance of $i$, $j$<br>4. $R1 = rl_{p,i}; R2 = rl_{p,j}$.<br>5. $C1 = C_{eff1} + 0.5cl_{p,i}; C2 = C_{eff2} + 0.5cl_{p,j}$.<br>6. $t_p$ = transition time limit of node $p$<br>7. $K1 = \frac{R1C1}{t_p}; K2 = \frac{R2C2}{t_p}$;<br>8. $C_{eff} = K1C1 + K2C2 + 0.5c(l_{p,i} + l_{p,j})$; return. |

**Figure 6: The procedure to evaluate the effective downstream capacitance recursively.**

scheme, which is explained below.

In any merging scheme, node pairs to be merged are selected as per a cost function. In most of the traditional merging schemes like [6], node pairs that are physically closest are merged together with the intention of reducing the total wire length. But, as noted in [17], this might result in excessive wire snaking when the nodes to be merged do not have similar delays. The algorithm in [7] selects the node pairs that result in the smallest delay after the merger. This generally results in a more balanced tree. However, the wirelength consumed is generally more. In [17], the pair that results in the minimal merging wirelength are merged. Since this is in some ways similar to the minimum spanning tree algorithm (which at each step selects the new edge with minimal cost), it results in much a lower wirelength when compared to the approaches of [6] and [7]. However, as noted in [17], it might result in an highly unbalanced clock tree. In our work, we use a modified form of the cost function of [17] such that a balanced structure is obtained and wirelength is also reduced.

**Top level algorithm**: The top-level steps involved in our buffer insertion flow are given below:

1. Initialize a list $F$ as an empty list. This list will contain all the *flagged, unmerged* nodes. A *flagged* node is one that cannot be merged with any of the other unmerged nodes without violating the limit on effective downstream capacitance (which is the maximum driving capability of the buffer used).
2. Initialize a list $U$ with the set of all the sink nodes. This list will store all the *unmerged, unflagged* nodes.
3. While $(Sizeof(U) + Sizeof(F) > 1)$ Do
   (a) $(T_i, T_j)$ = GetSubTreesToBeMerged(U) using steps in figure 7.
   (b) If $(T_i, T_j) \neq$ NULL
      i. Merge the subtrees to get a new subtree $T_k$. Obtain the values of *required slew* and $C_{effk}$ for node $k$ using figures 5 and 6.
      ii. Remove $T_i, T_j$ from $U$.
      iii. Add $T_k$ to list $U$.
   (c) else if ( $(T_i, T_j)$ = NULL) AND $(Sizeof(U) + Sizeof(F) > 1)$
      i. Insert buffers at all the nodes of $F$.
      ii. Update the values of delay, slew and effective downstream capacitance for all nodes $\in F$ using the delay characteristics of the buffer.
      iii. Move all the nodes in list $F$ to list $U$ and empty list $F$.
4. Perform top down embedding.

The key step in the above procedure is the step 3(a) which selects the node pairs to be merged. This step is detailed in figure 7. For node-pair selection, we use similar cost function as in [17] with an important change. In [17], a buffer will be inserted in a node *as and when* the node downstream capacitance exceeds a certain limit. But such an approach will result in an highly unbalanced clock tree.

In our algorithm, we insert buffers only when there is no node pair that can be merged without violating the *effective downstream capacitance limit*. To enforce this requirement, we maintain two separate lists - one called $F$ which will have a list of $flagged$ nodes and another list called $U$ in which we will store the list of $unflagged$ nodes. For node pair selection, we consider only the list $U$. If, for a particular node $i \in U$, we are not able to identify a suitable node pair for merger without exceeding the capacitance limit, we add that node to the list of *flagged* nodes $F$ and remove $i$ from $U$. We repeat the node-pair selection process until the list $U$ becomes empty or contains a single element that cannot be merged with any other node. At that stage, we add buffers to *all the unmerged nodes* of $F$, update their delays, slews and effective downstream capacitances and transfer all the nodes to the list $U$. This cycle continues till there is only a single clock tree.

| **Procedure:** $GetSubTreesToBeMerged(U)$ |
|---|
| **Input:** Set of all unmerged subtrees<br>**Output:** The two subtrees to be merged |
| 1. $PairsFound = 0$<br>2. While (PairsFound $\neq$ 1) AND $(Sizeof(U) > 1)$ Do<br>(a) $T_i$ = subtree with min root-sink delay in $U$<br>(b) $MergingCost = \infty$<br>(c) For each subtree $T_k \in U$ and $T_k \neq T_i$<br>i. $cost = MergingCost(T_i, T_k)$ defined in Fig. 8<br>ii. if $cost < MergingCost$<br>$MergingCost = cost; T_j = T_k$.<br>(d) if $MergingCost \neq \infty$<br>PairsFound = 1<br>else<br>Remove $T_i$ from $U$; Add $T_i$ to $F$.<br>3. if $MergingCost \neq \infty$<br>return ($T_i$, $T_j$)<br>else<br>Transfer the possible single node $\in U$ to list $F$.<br>return $NULL$. |

**Figure 7: The algorithm for selecting the subtrees to be merged.**

A minor point that may be noted here is that the $MergingCost$ algorithm of Figure 8 that is used in the $GetSubTreesToBeMerged$ returns a value of $\infty$ when a possible merger of two node pairs $i$ and $j$ causes the effective capacitance limit to be violated. Thus, only node pairs that result in a node with lesser effective capacitance than the preset limit are merged.

**Merits of balanced CTS algorithm**: An obvious advantage of the above procedure is that it will, by construction, result in a perfectly balanced clock tree. This is because buffers are added only in the step 3(c) of the top-leve

| **Procedure:** $MergingCost(T_i, T_j)$ |
|---|
| **Input:** A pair of subtrees<br>**Output:** The merging cost of the subtree pair |
| 1. Cost = Total wire length required to merge $T_i$ and $T_j$<br>2. EDSC = Effective downstream capacitance of the parent node *assuming* the merging of subtrees $T_i$ and $T_j$ using steps of figure 6<br>3. If $EDSC <$ Capacitance Limit<br>return Cost<br>else<br>return $\infty$ |

**Figure 8: The Merging cost for two subtrees.**

algorithm in which *all* the unmerged nodes are buffered. As a result, the number of buffers from the clock source to *every* sink will be the same, thus satisfying one of the important objectives of our work.

A less obvious advantage of the proposed merging scheme is that, on the average, all the nodes to be *flagged* are mostly in the same ballpark as 1/2 times the effective calacitance limit used in the figure 8. This results in similar equivalent capacitance loads for all the buffers at a given level. This helps to a great extent in reducing the actual SPICE skew. It may be noted here that works of [7, 12] also target the objective of balancing the loads for buffers at a given level. However, they obtain the balancing by adding excessive wire capacitance, which results in a big increase in total wirelength. In our scheme, since we merge nodes considering the wire length cost, our algorithm generally results in considerably lesser wirelength that [7, 12]. Our top-down embedding after obtaining the topology is identical to the DME algorithm [6].

# 5. LINK INSERTION FOR BUFFERED CLOCK TREES

Recently, a link based non-tree was proposed in [1], in which cross links are added to an existing clock tree so as to obtain a non-tree CND. Such a link based non-tree was shown to be highly tolerant to skew variations with minimal increase in wire length. An example is shown in figure 9. Consider the figure 9 (a), in which an unbuffered clock tree with a single clock driver is shown. In this tree, the sinks 4 and 5 are located physically very close to each other but still they share no common path other than the clock source S. As demonstrated in [1], adding a link (shown in dotted lines) between nodes such as 4 and 5 result in reducing the effect of variation factors on the clock skew. This is mainly because the addition of link introduces a redundant clock to sink path for the sinks. So the variations in these paths will tend to cancel each other resulting in low skew variability. In a practical clock tree, many such links needs to be added so as to sufficiently reduce the skew variability. In [1], the clock network is divided into a bipartite graph and Minimum Spanning Tree (MST) algorithm is used selecting the node-pair selection for link insertion. The cost function for MST link insertion is the Manhattan distance between the clock sinks. This helps in reducing the wirelength used for link insertion.

**Limitations of link insertion algorithm of [1]** : The algorithm used in [1] is applicable only to unbuffered clock networks and cannot be applied to buffered case for the following reasons:

- It considers only the Elmore delay while inserting the links. While Elmore delay has been shown to have good fidelity in [1] for the unbuffered clock trees w.r.t SPICE, the fidelity is poor for a buffered clock tree as demonstrated in [18].
- Adding links between two sinks driven by different buffers introduces the problem of multi-driver nets. For example, in figure 9 (b), the link between nodes 4 and 5 has two drivers, namely buffers A and B. If the links are not selected considering accurate delays, then it is possible to insert links between nodes whose delay values are quite different. As explained in [1], adding links between node-pairs with vastly different delays might actually harm the skew between the other pairs.
- Another concern with multi-driver links is that it might increase the short circuit power of the clock network. This is because of the virtual shorting of Vdd and Vss (Source and Ground) that might occur when one of the drivers gets turned much ahead of others or vice versa. For example, in figure 9(b), if the nominal delays of nodes 4 and 5 are vastly different, it means that buffers A and B will get switched on and off at quite different times. Because of this, nodes 4 and 5 might be on opposite voltage extremes (Vdd and Vss) for a considerable amount of time. This might increase the short circuit power drastically. By making sure that the nodes have very similar accurate delay values, we not only make the skew better, but also reduce the duration of the possible short-circuit power consumption.

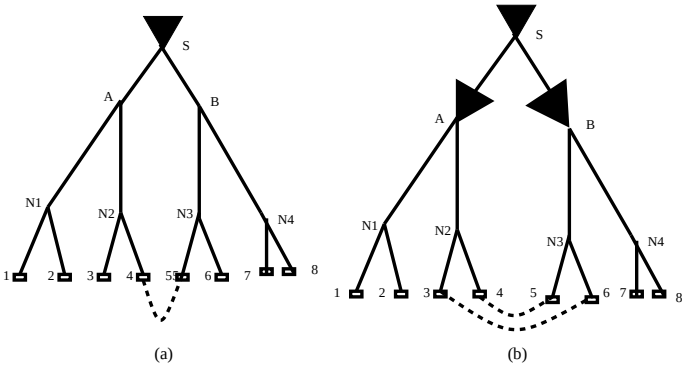


**Figure 9: An example of link-based non-tree. (a) Unbuffered case. (b) Buffered case.**

Thus, it is clear from the above points that link insertion for buffered clock tree is a non-trivial problem. In order to insert links in a buffered clock tree, we required the buffered clock tree to have a good nominal skew in SPICE. Otherwise, adding the links might either harm the overall skew between other node pairs or result in too much short circuit power. It may be noted here that our buffer insertion algorithm is especially suited for link insertion because of the balanced nature of the clock tree and consideration of more accurate delay models than most existing algorithms.

## 5.1 Link insertion flow

The approach taken by us to address the concerns regarding the algorithm of [1] for link insertion for buffered clock trees is detailed below:

- We overcome the problem of inaccuracies in Elmore delay by using the same bottom-up clock tree synthesis flow as in section 4.2. Since we consider both resistive shielding and signal slew, we are able to address the most important drawback of using Elmore delay, namely accuracy.

- During node pair selection for link-insertion, we consider both the spatial proximity of the nodes *and* the proximity in terms of delays. This enables us to address the concerns about excessive short-circuit power. This approach also makes sure that link addition does not affect the skew between other sink pairs adversely. More specifically, we use a modified cost function for the MST algorithm of [1] by making the link insertion cost as a weighted function of *both* link insertion and accurate delays (obtained using the algorithm of [19]) of the clock tree end points before link insertion. For example, in figure 9(b), assume that the nodes 4 and 5 have delays that differ considerably and that nodes 3 and 6 have accurate delays very close to each other. The original algorithm will insert the link between nodes 4 and 5 because they are the physically closest pairs. However, as discussed above, if the delays of nodes 4 and 5 differ considerably, then choosing the link between 3 and 6 will result in a better in terms of both skew variability reduction and short circuit power reduction eventhough the link between nodes 3 and 6 is slightly longer than the link between nodes 4 and 5.

The major steps in constructing a linked buffered clock tree are:

1. Construct a buffered clock tree using the flow described in Section 4.

2. Obtain the node pairs to be linked using a modified form of algorithm in [1] with the cost function as weighted function of link length and proximity of accurate delays for the node pairs. The accurate endpoint delays are obtained using the algorithm of [19] for delay evaluation.

3. Obtain the values of extra capacitances to be added at the sinks due to link insertion. Since the link positions are fixed, the amount of extra capacitance to be added can be calculated once the node pairs have been selected.

4. Using the link capacitance values as extra load capacitance at the selected sinks, construct another buffered clock tree with the same topology as the one constructed in step(1). This new buffered clock tree will be equivalent to the first buffered clock tree *plus* the link capacitance.

5. Add the link resistances to the new clock tree built in step (3). As described in [1], adding link resistances does not change the nominal values of skew once the link capacitances are accounted for. The final result will be equivalent to the buffered clock tree constructed in the first step *plus* the link capacitance *and* link resistances. This is our final buffered, linked clock network.

# 6. EXPERIMENTAL RESULTS

In order to verify the variation tolerance of our new buffered clock tree and the linked buffered clock tree approaches, we run SPICE based Monte Carlo simulations (500 trials) considering both interconnect and device variations. We assume that interconnect width, loac capacitances, device channel lenght and oxide thickness vary with a Gaussian distribution with $\sigma = 5\%$. We implemented our algorthms in C++ and experiments were run with a 3.25GHz, 2Gb memory Linux system. We use the same benckmarks as in [5] scaled for the paramaters for $90nm$ technology. We scale both the load capacitance and the chip area so that the bench marks are applicable for $90nm$ technology.

We compare our results with the algorithms in [7] and [17]. We chose these two algorithms for comparison because the algorithm in [17] will result in a clock tree with greatly reduced wire length consumption. So it can be a good benchmark to do the wirelength comparisons. The algorithm in [7], due to its balanced nature, is likely to yield a good and balanced clock tree with reduced skew variability. Thus, comparing our results with these two algorithms will give us apt benchmarks for both wirelength and skew. It may be noted that, for the major part, the code for our algorithms and our implementation for [7, 17] are identical *except* the merging schemes used. So the difference in runtime and results can be directly attributed to the different merging schemes.

Since the results of [17] are expected to yield the minimum wire length and worst skew (because of its unbalanced clock trees), we use [17] as the baseline for comparing our results. The skew variation and resource consumption for [17] are shown in table 1. While selecting the clock trees for different algorithms, we made sure that all of them meet the slew requirement of 100ps that we enforced on the clock tree points. We also made sure that the clock tree with minimal resources that met the slew criterion was selected for each algorthm so as to ensure a fair comparison.

Table 2 shows the results of our new algorithms and the algorithm in [7] *scaled in terms of* the results of [17] (All the columns expect the # Buf and CPU have been scaled). The Method column specifies the method for which results have been give. We have identified the algorithm of [7] and [17] as CTS of [7] and CTS of [17] respectively. We identify our algorithms as CTS and Link+CTS. The wire length consumtion is shown under the column titles WL. The '# Buf' column gives the number of buffers for the particular clock tree. The NS, WCS and AS denote the 'Nominal Skew', 'Worst Case Skew' and 'Average Skew' in SPICE, the last two values obtained for 500 trials of Monte Carlo simulations in SPICE. Finally, the CPU time for completing the clock tree synthesis (in seconds) is shown in the column CPU. The important observations from table (2) are as follows:

- From column 2 of table 2, it can be observed that our buffered clock tree results in comparable wirelength to that of [17] and much reduced wirelength than [7] always.
- As expected, the skew values for [17] is the worst among all the algorithms. Also, it can be observed that our buffer insertion algorithm produces consistently better results than [7] in terms of skew variability reduction.

- The linked, buffered clock networh has the best skew variability reduction among all the algorithms. Also,

the percentage of extra wirelength consumed for link insertion is small and drops heavily as the size of the clock tree increases. This proves the effectiveness of link insertion for buffered clock trees.

- The CPU time consumed for the algorithm [17] is the lowerst while our algorithms yields comparable CPU times. When compared to the run times of [7], the run times of our buffer insertion algorithm and the linked buffered clock network algorithm are much faster.

  Thus, it can be said that our algorithms are a fast and efficient approach to get variation tolerant buffered clock trees and linked buffered clock tree.

| TC | WL | # Buf | NS | WCS | AS | CPU |
|---|---|---|---|---|---|---|
| r1 | 25937 | 16 | 100 | 190 | 76 | 0.06 |
| r2 | 34110 | 28 | 96 | 222 | 60 | 0.36 |
| r3 | 34353 | 36 | 101 | 196 | 52 | 0.71 |
| r4 | 55115 | 78 | 176 | 362 | 76 | 3.46 |
| r5 | 109722 | 163 | 110 | 226 | 56 | 9.4 |

**Table 1: Skew variation and resource consumption results for the algorithm in [17]**

| TC | Method | WL | # Buf | NS | WCS | AS | CPU |
|---|---|---|---|---|---|---|---|
| r1 | CTS of [17] | 1.0 | 16 | 1.0 | 1.0 | 1.0 | 0.06 |
| | CTS of [7] | 5.219 | 18 | 0.57 | 0.72 | 0.46 | 1.1 |
| | Our CTS | 0.829 | 18 | 0.37 | 0.49 | 0.23 | 0.08 |
| | Link+CTS | 1.136 | 18 | 0.41 | 0.45 | 0.16 | 0.18 |
| r2 | CTS of [17] | 1.0 | 28 | 1.0 | 1.0 | 1.0 | 0.36 |
| | CTS of [7] | 7.588 | 36 | 0.91 | 0.95 | 0.91 | 14 |
| | Our CTS | 1.836 | 40 | 0.62 | 0.60 | 0.59 | 0.42 |
| | Link+CTS | 1.852 | 40 | 0.65 | 0.39 | 0.37 | 0.52 |
| r3 | CTS of [17] | 1.0 | 36 | 1.0 | 1.0 | 1.0 | 0.71 |
| | CTS of [7] | 9.627 | 41 | 0.59 | 0.57 | 0.61 | 44 |
| | Our CTS | 1.192 | 45 | 0.49 | 0.54 | 0.51 | 0.78 |
| | Link+CTS | 1.395 | 45 | 0.51 | 0.40 | 0.32 | 0.88 |
| r4 | CTS of [17] | 1.0 | 78 | 1.0 | 1.0 | 1.0 | 3.46 |
| | CTS of [7] | 12.46 | 85 | 0.56 | 0.55 | 0.47 | 509 |
| | Our CTS | 2.164 | 83 | 0.34 | 0.42 | 0.36 | 3.94 |
| | Link+CTS | 2.173 | 83 | 0.41 | 0.33 | 0.25 | 4.41 |
| r5 | CTS of [17] | 1.0 | 163 | 1.0 | 1.0 | 1.0 | 9.4 |
| | CTS of [7] | 9.1894 | 174 | 0.79 | 0.55 | 0.49 | 2009 |
| | Our CTS | 1.5236 | 183 | 0.46 | 0.38 | 0.35 | 10.12 |
| | Link+CTS | 1.588 | 183 | 0.48 | 0.30 | 0.28 | 11.62 |

**Table 2: Skew variation and resource consumption results for our new algorithms and algorithms in [7] in terms of resuts of [17] in Table 1**

# 7. CONCLUSIONS

We have proposed an effective clock tree synthesis methodology for constructing a perfectly balanced clock network with link addition. The clock network is robust and tolerant to varation effects. When compared to existing algorithms, there is as much as 50% reduction in the average skew in the clock sinks due to variation effects. Also, the buffer/wire length cost and CPU time is also significantly less than most of the previous algorithms.

# 8. REFERENCES


[1] A. Rajaram, D. Z. Pan, and J. Hu, "Improved Algorithms for Link Based Nontree Clock Networks for Skew Variability Reduction," in *Proceedings of the ISPD 2005*, San Fransisco, CA, April 2005.

[2] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," in *Proceedings of the IEEE*, vol. 89, no.5, pp.665–692, May 2001.

[3] R. Saleh, S. Z. Hussain, S. Rochel, and D. Overhauser, "Clock skew verification in the presence of IR-drop in the power distribution network," in *IEEE Transactions on CAD*, vol.19, no.6, pp.635–644, June 2000.

[4] W.-C. D. Lam, C.-K. Koh, and C.-W. A. Tsao, "Power supply noise suppression via clock skew scheduling," in *Proceedings of the IEEE ISQED*, San Jose, CA, March 2002, pp. 355–360.

[5] R.-S. Tsay, "Exact zero skew," in *Proceedings of the IEEE/ACM ICCAD*, Santa Clara, CA, November 1991, pp. 336–339.

[6] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese, and A. B. Kahng, "Zero skew clock routing with minimum wirelength," in *IEEE Transactions on CS-ADSP*, vol.39, no.11, pp.799–814, November 1992.

[7] Y. P. Chen, and D.F. Wong, "An algorithm for zero-skew clock tree routing with buffer insertion," in *Proceedings of the ED & TC*, Pairs, France, March 1996, pp. 230–236.

[8] S. Pullela, N. Menezes, and L. T. Pillage, "Low power IC clock tree design," in *Proceedings of the CICC*, May 1995, pp.263–266.

[9] J. Chung and C.K. Cheng, "Optimal Buffered Clock Tree Synthesis," in *IEEE ASIC conference*, Austin, TX, Sept. 1994, pp. 130–133.

[10] A. Vittal, and M. Marek-Sadowska, "Low-power buffered clock tree design," in *IEEE Transactions on CAD*, vol. 16, no. 9, pp. 965 - 975 , Sept. 1997.

[11] G. E. Tellez, and M. Sarrafzadeh, "Minimal buffer insertion in clock trees with skew and slew rate constraints" in *IEEE Transactions of CAD*, vol. 16, no.4, pp.333–342, April 1997.

[12] A. D. Mehta, Y. P. Chen, N. Menezes, D. F. Wong, and L. T. Pilegg, "Clustering and load balancing for buffered clock tree synthesis" in *Proceedings of the ICCD*, Austin, Tx, October 1997, pp. 217–223.

[13] Y. Liu, X. Hong, Y. Cai, and X. Wei, "Reliable buffered clock tree routing algorithm with process variation tolerance" in *Proceedings of the ASIC*, October 2003, pp. 344–347.

[14] B. Lu, J. Hu, G. Ellis, H. Su, "Process variation aware clock tree routing," in *Proceedings of the ISPD*, Monterey, CA, April 2003, pp. 174–181.

[15] J. Tai. Yan, C. W. Wu, K. P. Lin, Y. C. Lee, and T. Y. Wang, "Iterative convergence of optimal wire sizing and available buffer insertion for zero-skew clock tree optimization" in *Proceedings of Asia-Pacific Conference*, December 2004, pp.529–532.

[16] J. L. Tsai, T. H. Chen, and C. C. P. Chen, "Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing" in *IEEE Transactions of CAD*, vol. 23, no. 4, pp.565 - 572, April 2004.

[17] R. Chaturvedi, and J. Hu, "Buffered clock tree for high quality IC design" in *Proceedings of the ISQED*, March 2004. pp. 381–386.

[18] K. Wang, and M. Marek-Sadowska, "Clock network sizing via sequential linear programming with time-domain analysis" in *Proceedings of the ISPD*, Monterey, CA, April 2003, pp. 182–189.

[19] R. Puri , D. S. Kung, and A. D. Drumm, "Fast and accurate wire delay estimation for physical synthesis of large ASICs" in *Proceedings of the GLSVLSI*, New York, NY, April 2002, pp. 30-36.

[20] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *Proceedings of the IEEE CICC*, San Diego, CA, May 2001, pp. 223–228.

[21] D. Harris, and S. Naffziger, "Statistical clock skew modeling with data delay variations" in , *IEEE Transactions on VLSI Systems*, vol.9, no.6, pp.888-898, December 2001.

[22] Y. Liu , S. R. Nassif , L. T. Pileggi, and A. J. Strojwas, "Impact of interconnect variations on the clock skew of a gigahertz microprocessor" in *Proceedings of DAC*, Los Angeles, CA,, June 2000, pp.168-171.