Clustering and Load Balancing for Buffered Clock Tree Synthesis[*]

Ashish D. Mehta, Yao-Ping Chen[†], Noel Menezes, D.F. Wong[†], and Lawrence T. Pileggi
Department of Electrical and Computer Engineering
[†]Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1084

Abstract

Buffers in clock trees introduce two additional sources of skew: The first source of skew is the effect of process variations on buffer delays. The second source of skew is the imbalance in buffer loading. We propose a buffered clock tree synthesis methodology whereby we first apply a clustering algorithm to obtain clusters of approximately equal capacitance loading. We drive each of these clusters with identical buffers. A sensitivity based approach is then used for equalizing the Elmore delay from the buffer output to all of the clock nodes. The skew due to load imbalance is minimized concurrently by matching a higher-order model of the load by wire sizing and wire lengthening. We demonstrate how this algorithm can be used recursively to generate low-skew buffered clock trees.

1 Introduction

Clocks constitute some of the most important signals on a chip. With increasing clock frequency it is crucial to maintain sharp clock edges and control skew within tolerable limits. Moreover, given the current thrust in low power design, it is important to distribute the clock signal with minimum power consumption since it is the most active signal on the chip [1]. In [2] it was shown that significant savings in power can be made by selectively widening wires and inserting buffers in clock trees. The overall power can further be reduced by clock gating – "disconnecting" the clock to sections of logic that are inactive – which would not be possible in unbuffered trees. Also, it is well known that maintaining sharp signal edges requires intermediate buffers for signal regeneration. While buffer insertion offers these advantages, it is also a source of clock skew due to load mismatch and process-induced skew due to parameter variations. The skew due to process parameter variations can be limited by using identical buffers at a level. This makes the design as process *insensitive* as possible, but further requires that buffers at the same level of the clock tree drive identical loads. Thus it is imperative to apply a clustering algorithm as a preprocessing step to the routing and synthesis steps so that buffers at the same level drive nearly identical loads.

In this paper we first describe a clustering algorithm to obtain clusters of nearly identical loading. We then drive each of these clusters with identical buffers. Since the Elmore delay [3, 4] has been shown to be a good measure of the signal delay for skew reduction purposes [5], we equalize the Elmore delay from the buffer output to all the clock nodes. More importantly, we concurrently design the interconnect so that drivers at the same level are loaded by the same n^{th}-order impedance, thus minimizing the skew due to load imbalance.

2 Clock Skew Due to Load Mismatch

Consider a typical clock tree structure, as shown in Figure 1. In order to reduce skew we would attempt to equalize the signal delay from the main driver A to all of the clock nodes driven by buffers B and C. Traditional design automation algorithms approach this problem by first equalizing the Elmore delay from A to node 1 and from A to node 3. In [6] and [7] it was suggested that at a

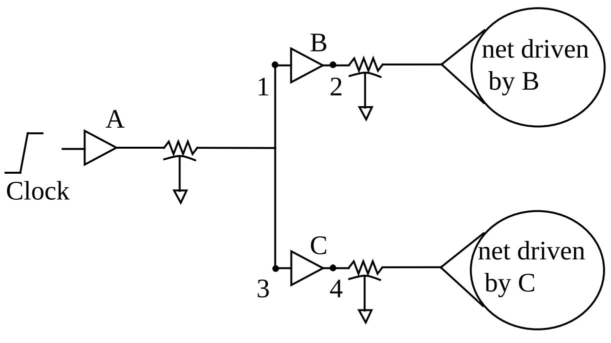


Figure 1: A typical buffered clock tree structure.

level of the tree buffers should be identical, hence buffers B and C are sized equally. The Elmore delay from nodes 2 and 4 to all the clock nodes in the nets driven by buffers B and C respectively are then equalized. The remaining part

* This work was supported in part by the Semiconductor Research Corporation under contract 95-DJ-343, the National Science Foundation under contract MIP-9157263, and IBM Corporation.

of the problem requires that the delays across buffers B and C be equalized. Since buffers B and C are identical, it is necessary that the load driven by them also be identical, as the delay through a gate is a function of its output loading. Current clock tree synthesis algorithms equalize the buffer delay through buffers B and C by either matching the total capacitive load at the driver output [8], or, even better, the effective capacitance seen by the driver [6].

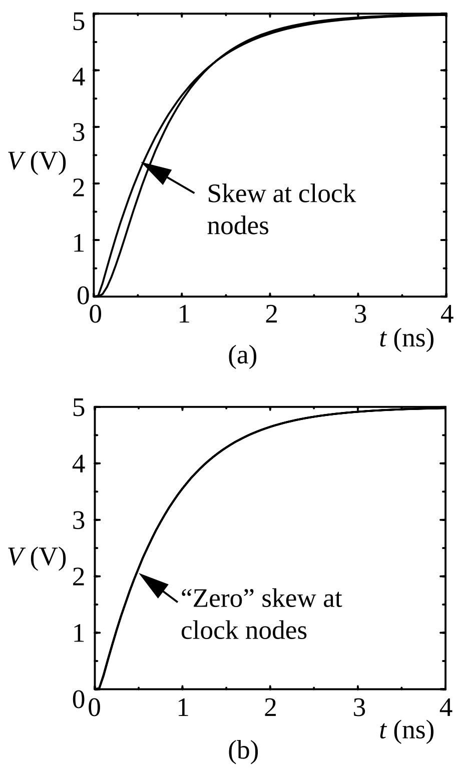


Figure 2: Response at two clock nodes after matching a) the total capacitance load, b) the π-model of the load.

There are two major problems associated with approaches that fail to consider the load imbalance, or consider it only in terms of a simplistic model. The first is that neither the total capacitance nor the effective capacitance are completely accurate models of the load. With scaling there is an increase in the interconnect resistance and a decrease in the gate output impedance. Also the interconnect lengths generally do not scale due to increased chip density. Thus, overall we see an increase in the interconnect resistance. This increased interconnect resistance shields some of the capacitance from the driver making the total capacitance a pessimistic approximation of the load [9]. In [10] it was shown that a π-model of the load, which can be synthesized from the first three moments of the driving-point admittance, accurately captures the resistance shielding effect of the interconnect. (We note that the total capacitance is the *first moment* of the driving-point admittance.) Higher-order models of the load are synthesized with similar ease using even more moments. In this paper we match the π-model of the load by matching the first three moments of the driving-point admittance to eliminate the skew due to load imbalance.

As an illustrative example, we powered two clock subnets in a configuration similar to that in Figure 1: one with 15 clock nodes forming a total capacitance of 4.69 pF was driven by buffer B, and another with 20 clock nodes with a total capacitance of 4.5 pF was driven by buffer C. We equalized the Elmore delay from the driver outputs of B and C to all of the associated clock nodes. We then obtained signal transition curves at two clock nodes, one of which was driven by B and the other by C. The curves in Figure 2a were obtained when we equalized the Elmore delays while simultaneously balancing the total capacitance of the two subnets. In Figure 2b, we show responses at the same clock nodes when we balanced the Elmore delays and the first three moments of the driving-point admittance at the driver outputs of B and C. It is apparent that substantial skew can result by using the total capacitance load model. *It is also clear that this load-imbalance skew can be practically eliminated by matching the first several moments of the driving-point admittance.*

As mentioned previously, the total net capacitance represents the first moment of the driving-point admittance. Hence, matching the total capacitances of the nets driven by identical buffers at the same level is an important, if not complete, step in matching the driving-point admittances. Referring to the clock tree structure of Figure 1, if the total capacitive load, that is, the first driving-point admittance moment, seen by driver B was much larger than the total capacitive load seen by driver C, then there would be no choice but to add extra capacitance to the net driven by driver B in order to match the first moments of the respective driving-point admittances. This would be at the expense of extra power dissipation and a degradation in the signal edge rate at the buffer output. This is the second major problem associated with current clock tree synthesis algorithms. The above example clearly shows the need to apply a clustering algorithm as a preprocessing step to the routing and the synthesis phases. The objective of the clustering is to partition the clock nodes into clusters of nearly equal capacitive loading with the interconnect capacitance of the cluster taken into account. Thus our clock tree synthesis algorithm is divided into two steps. The first is the clustering and routing phase which attempts to equalize the load driven by buffers at the same level, and generates an initial route for the clock nodes. The second step is the synthesis step, in which the we adjust the wire widths and the wire lengths to match the Elmore delays at the clock nodes, and the first three moments of the driving-point admittance at the buffer outputs for all buffers at the same level. In the following sections we describe our clustering and synthesis algorithms.

3 Clustering and Routing of Clock Trees for Load Balancing

We have developed a clustering algorithm that can be used at different levels of a clock tree. At the bottom level, the algorithm clusters clock pins, while at the middle levels of the clock tree, it clusters clock buffers. The input of the algorithm is a set of clock pins or clock buffers with associated locations and input capacitances and the number of required clusters, K. The task is to partition the input set into K clusters such that all of them have approximately the same load, and then route the K sets of clock pins or buffers independently. To simplify the explanation, we will discuss the algorithm only in terms of clock pins. However, nothing precludes these pins as representing a set of clock buffers.

We route each cluster using one of the best available clock algorithms [11] which generates a zero-skew clock tree under the Elmore delay model while minimizing the total wire length. In [11], it was shown that the clock routing algorithm generates a clock tree with wire length of the order of $O(\sqrt{N}D)$, where D is the *diameter* of the input set which is defined as the Manhattan distance between the two farthest points in the set. This order is the best possible, since the minimum Steiner tree has the same order of wire length. Suppose a cluster consists of N clock pins. Let c_i denote the load of clock pin i. The total load of each cluster, P_j $(1 \le j \le K)$, can be measured by a cost function

$$C(P_j) = \alpha \sum_{i=1}^{N} c_i + \beta \sqrt{N}D \qquad (1)$$

where α and β are the weights of the two terms, and D is as defined above. The *clock pin clustering problem* can now be described as follows:

> Given a set of clock pins (buffers), partition them into K clusters, such that the sum of the cost functions of all clusters is minimized, and the ratio of the largest cost function to the smallest cost function is less than a given real bound B, where $B \ge 1.0$.

This problem is difficult as shown by the following lemma. (*All proofs have been omitted for brevity.*)

Lemma 1. The clock pin clustering problem is NP hard.

3.1 Clock pin clustering by slicing structures

The clustering problem as formulated above is unnecessarily difficult for our application. For our problem, it is essential that the routing trees formed by each cluster do not overlap with one another, so that we have some control of routing congestion. Therefore, we consider solving the clustering problem by partitioning the chip area. One way of partitioning is to cover the chip area by mutually disjoint rectangles as shown in Figure 3a. In

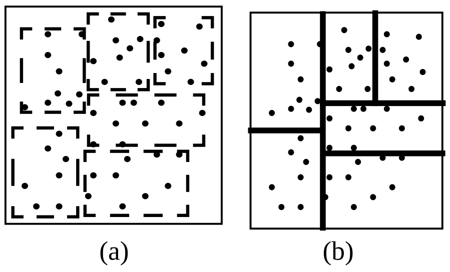


Figure 3: Clock pin clustering by a a) non-slicing structure, b) slicing structure.

many situations, a partition can be represented by a *slicing structure* [12]. For example, the partition in Figure 3a is represented by the slicing structure shown in Figure 3b, which is similar to the slicing floorplan of [12]. Starting with the rectangle corresponding to the whole chip, we recursively cut a rectangle into smaller rectangles by vertical or horizontal cut lines. The slicing structure enables us to apply the dynamic programming technique [13]. The algorithm for partitioning clock pins subject to the slicing structure constraint is presented in Fig. 4.

3.2 Algorithm description

Algorithm 1 in Figure 4 is a recursive procedure that does clustering at any level of the slicing structure in terms of four input parameters. Line 1 checks whether the given clustering problem has been solved. A problem can be identified by R and K, which means that the pins contained by rectangle R are to be partitioned into K clusters. We store the solved problems in a hash table with hash key K and the bottom left and top right corners of R. If the problem has been solved, it simply returns the list of possible clustering solutions associated with it (Line 2). On the other hand, if the problem has not been solved, it checks whether it is the base case ($K = 1$). If so, it calculates the cost of the cluster using equation (1). Otherwise, it considers all possible bipartitions generated by either vertical cuts or horizontal cuts under all k_1 and $K - k_1$ combinations. For each bipartition, it recursively calls itself twice. After the two recursive calls return, it merges the two resultant lists and appends to its own list L. It discards all unbalanced solutions and keeps the balanced ones. An element of the list corresponds to a set of K clusters. It has three fields, which are the average cost of each cluster in the cluster set, the largest cost of all clusters in the set, and the smallest cost of all clusters in the set. The second and third fields are used for determining whether a combined cluster set at the next higher level is balanced (Line 16). In addition, for each element of L, we store the necessary

Algorithm 1: Clustering-By-Slicing (*S, R, K, B*).

```
Input:  1. A set $S$ of $N$ clock pins (buffers),
        2. The rectangle $R$ containing $S$,
        3. The number of desired clusters $K$,
        4. The balance bound $B\ (B \geq 1.0)$ .
Output: A list $L$ of balanced clustering solutions such that
        $L = \{(Average, Largest, Smallest) \mid Largest/Smallest \leq B\}$.
Begin.
1.  if $(R, K)$ is already processed
2.      return the list associated with $(R, K)$;
3.  else
4.      $L = \phi$;
5.  if $K = 1$ return $L = \{(\mathrm{cost}(S), \mathrm{cost}(S), \mathrm{cost}(S))\}$;
6.  else for cut-line = {"vertical", "horizontal"} do
7.      for $k_1 = 1$ to $K - 1$ do
8.      Compute all bipartitions $P = \{((R_1, k_1), (R_2, K - k_1)) \mid$
9.                          $(R_1 \cap R_2 = \emptyset)$ and $(R_1 \cup R_2 = R)$ ;
10.     for each $((R_1, k_1), (R_2, K - k_1))$ in $P$ do
11.         Compute clock pin set $S_1(S_2)$ contained by $R_1(R_2)$;
12.         $L_1$ = Clustering-By-Slicing $(S_1, R_1, k_1, B)$;
13.         $L_2$ = Clustering-By-Slicing$(S_2, R_2, K - k_1, B)$;
14.         for each $(t_1, l_1, s_1)$ in $L_1$ do
15.             for each $(t_2, l_2, s_2)$ in $L_2$ do
16.                 if $max(l_1, l_2)/min(s_1, s_2) \leq B$
17.                     $Ave = (t_1 \times k_1 + t_2 \times (K - k_1))/K$;
18.                     Append $(Ave, max(l_1, l_2), min(s_1, s_2))$ to $L$;
19. return $L$;
End.
```

Figure 4: Algorithm for clock pin clustering subject to slicing structure constraints.

information for determining its two children, though it is not shown in Algorithm 1.

The first field, which is the average cost of each cluster in a cluster set, is computed by the sum of the costs of all clusters divided by the number of clusters. We use it to denote the *clustering cost* of a clustering solution. When Algorithm 1 is done, we choose the element in L whose clustering cost is the smallest. This element and all of its descendents form a binary tree representing the slicing structure. By a depth-first traversal, we can find all leaf elements. The set of all leaf elements is the best clustering solution of the clock pin set S.

Lemma 2. Algorithm 1 exactly computes the list of all balanced clustering solutions, each of which corresponds to a slicing structure.

Since the size of the list L grows exponentially, the time and space complexity of Algorithm 1 is not polynomial. We discretize the three fields of each list element. Let $\Delta_c = [c_{min}, c_{max}]$ be the range of the values for the three fields, where c_{min} and c_{max} are the minimum and maximum possible cost of a single cluster in the given problem. We partition Δ_c into M intervals, $[i \cdot \delta + c_{min}, (i+1) \cdot \delta + c_{min}]$, $0 \leq i \leq M - 1$, where $\delta = (c_{max} - c_{min})/M$. Given any two elements (t, l, s) and (t', l', s') whose corresponding fields all fall into the same intervals, only the one with the smaller first field value (clustering cost) is stored.

Theorem 1. Given $\varepsilon > 0$, there exists a discretization such that Algorithm 1 runs in polynomial time using polynomial space and $|C_A - C_{OPT}| \leq \varepsilon$, where C_A is the best clustering cost obtained by the discretized version of Algorithm 1, and C_{OPT} is the optimal clustering cost.

Theorem 1 states that Algorithm 1 after discretization runs in polynomial time using polynomial space. Though the worst case complexity of the discretized version of Algorithm 1 is a high-degree polynomial, the average running time and required space should be far less than those of the worst case. To further speedup the algorithm, we also divide the chip area into small cells such that the pins in the same cell are always in the same cluster.

4 Clock Subtree Synthesis for Buffered Clock Trees

The clustering algorithm described above is an essential first step in equalizing the load seen by the buffers at the same level. Skew reduction and matching of the input admittances of each cluster at the same level is done by varying the widths and lengths of the interconnect branches of each cluster to achieve a uniform target Elmore delay at all clock nodes and to concurrently equalize the first three moments of the driving-point admittance.

4.1 Problem formulation

Each branch of the clock tree represents a uniformly distributed RC line (URC). In our implementation, we section each URC into equally-sized segments using the lumping technique described in [14]. For clarity in exposition, however, in this paper we model each URC segment by an equivalent lumped L-section. We assume that branch i connects node i to its parent node in the tree. The lumped resistance R_i of each branch i with length l_i and width w_i is given by $r_i l_i / w_i$, where r_i is the sheet resistance. The lumped capacitance C_i of branch is given by $c_i l_i w_i + 2 f_i l_i$, where c_i is the capacitance per unit length per unit width and f_i is the fringe capacitance per unit perimeter. The leaf nodes have additional load capacitance C_{Li}. The set of all branches which lie on the path from a node n to the root is denoted by $P(n)$ while the set of all descendant nodes of node n and node n itself is denoted by $D(n)$. Let T denote the set of all branches and L denote the set of all leaf nodes. The j^{th} moment of the voltage response at node i is denoted by m_j^i [18].

With the above definitions the Elmore delay [3] at a node n in an RC tree is expressed by the following equation [4]:

$$m_1^n = \sum_{i \in P(n)} r_i \frac{l_i}{w_i} \sum_{j \in D(i)} (c_j l_j w_j + 2 f_j l_j + C_{Lj}). \qquad (2)$$

If $V_i(s)$ is the voltage response at the i^{th} node to an impulse function at the root of the clock subtree, then the current flowing out of the root of the clock subtree is given by:

$$I(s) = \sum_{i \in T} s(c_i l_i w_i + 2 f_i l_i + C_{Li}) \cdot V_i(s). \qquad (3)$$

The driving-point admittance, $Y(s) = I(s)/V_{in}(s) = I(s)$ since the input voltage at the root of the tree, $V_{in}(t)$, is an impulse function. Therefore, if m_j^Y denotes the j^{th} moment of the driving-point admittance [10], we have the following relationship:

$$m_0^Y + m_1^Y s + \ldots = \sum_{i \in T} s(c_i l_i w_i + 2 f_i l_i + C_{Li}) \left(m_0^i + \ldots \right) \quad (4)$$

By matching the coefficients of the orders of s in (4), the first three moments of the driving-point admittance are expressed as:

$$\begin{aligned} m_1^Y &= \sum_{i \in T} (c_i l_i w_i + 2 f_i l_i + C_{Li}) m_0^i, \\ m_2^Y &= \sum_{i \in T} (c_i l_i w_i + 2 f_i l_i + C_{Li}) m_1^i, \\ m_3^Y &= \sum_{i \in T} (c_i l_i w_i + 2 f_i l_i + C_{Li}) m_2^i. \end{aligned} \qquad (5)$$

We synthesize the clock tree interconnect to optimize for area by minimizing $\Sigma l_i w_i$ while trying to equalize the Elmore delay at the clock nodes and match the first three driving-point admittance moments at the buffer outputs. Thus, for a buffer and its associated net, our problem can be stated formally as follows:

$$\begin{aligned} &\text{minimize } area = \sum_{i \in T} l_i w_i \\ &\text{subject to } m_1^i = T_d, \qquad i \in L, \\ &\qquad m_1^Y = \hat{m}_1^Y,\ m_2^Y = \hat{m}_2^Y,\ m_3^Y = \hat{m}_3^Y, \\ &\qquad \underline{w}_i \le w_i \le \bar{w}_i, \quad i \in T, \\ &\qquad \underline{l}_i \le l_i \le \bar{l}_i, \qquad i \in T. \end{aligned} \qquad (6)$$

In the above equations, T_d, $\hat{m}_0^Y$, $\hat{m}_1^Y$, $\hat{m}_2^Y$ are the target Elmore delay at the clock nodes and the target first three moments of the driving-point admittance respectively. We also place minimum and maximum wire width and wire length constraints on all wires. Alternatively, we may also want to minimize the clock tree interconnect (CV^2) power dissipation subject to the same constraints.

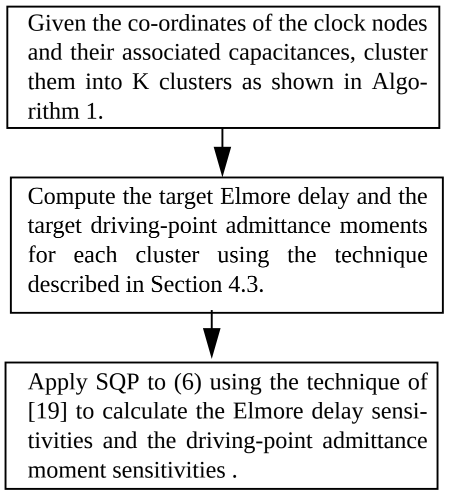
4.2 Wire sizing using sequential quadratic programming (SQP)

We apply sequential quadratic programming (SQP) [15], one of the most common approaches to non-linear optimization, to the optimization problem defined in (6). At each iteration, a QP subproblem is constructed from a quadratic approximation of the non-linear objective function and the linearization of the constraints about the solution from the previous iteration. The solution of the QP subproblem which is determined by any general-purpose QP-solver is then used as the initial solution for the next iteration. Furthermore, due to the success of SQP, several commercial as well as academic SQP implementations, for example, [17], are readily available.

SQP, like most gradient-based optimization techniques, can be speeded up significantly by efficient computation of the sensitivities of the objective function as well as the constraints. Since the objective function in our case can be expressed analytically the main bottleneck is the computation of the sensitivities of the constraints of (6) which are related to the moments of the RC tree. We apply the moment sensitivity computation algorithm described in [19], which applies path-traversing techniques to RC trees, for efficient sensitivity computation.

4.3 Selecting target driving-point admittance moments

For selecting appropriate target moments for the driving-point admittance, we first minimize the area of each cluster subject to Elmore delay constraints only. Hence, we equalize the Elmore delay at all the clock nodes for all the clusters while simultaneously ensuring that minimum wiring capacitance is added to each of the clusters. Since the Elmore delay at all the clock nodes in each of the clusters have been equalized with minimum wiring capacitance for each of the clusters, it is necessary to choose the target first moment of the driving-point admittance to be the largest total capacitance of all clusters. The same argument holds for the second and the third moment of the driving-point admittance. In the various examples that we have run, we have found that generally the cluster with the largest first moment of driving-point admittance is also the cluster which has the largest second and third moment of the driving-point admittance. A flow for the wire sizing and clustering steps is presented in Figure 5.



Given the co-ordinates of the clock nodes and their associated capacitances, cluster them into K clusters as shown in Algorithm 1.

Compute the target Elmore delay and the target driving-point admittance moments for each cluster using the technique described in Section 4.3.

Apply SQP to (6) using the technique of [19] to calculate the Elmore delay sensitivities and the driving-point admittance moment sensitivities .

Figure 5: Algorithm 2: Wire sizing and lengthening to match the Elmore delays and the π-model of the load.

4.4 Recursive clock tree generation

The synthesis approach described above can be applied recursively to generate a zero skew clock tree. Consider Figure 6 where the clock pins are indicated by crosses. In order to generate a zero skew clock tree, we first cluster the clock nodes into clusters of nearly equal capacitive loading using Algorithm 1. Each of these clusters is then driven by identical buffers. We then equalize the Elmore delay from the buffer outputs to all the clock nodes associated with it, and concurrently match the π-model of the load at the buffer output using the approach described in Figure 5. For the next level of clustering, the coordinates of the buffers become the coordinates of the clock nodes, and the loading at these nodes is equal to the buffer input capacitance. Thus we generate another level of the clock tree.

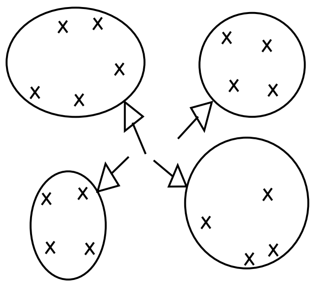


Figure 6: Recursive clock tree generation: buffer input capacitances and locations used as input for next level of clustering.

5 Results

We have applied the above clustering and synthesis algorithm to the benchmark examples of [5]. We first applied our clustering algorithm, then drove the clusters with identical buffers. We equalized the Elmore delays and driving-point admittances within clusters. The number of clusters was based on the maximum allowable clock pin load in a cluster. Process parameters for our experiments are based on the 0.8 μm MOSIS process.

Results are shown in Table 1 for the benchmark circuits of [5]. The clustered circuits were simulated with HSPICE to measure the skew. The third column in Table 1 shows the number of clusters for each of the examples while the last column displays the values for the maximum skew at the 50% time-point between the clock nodes of each clusters.

Example	No. of pins	No. of clusters	Total int. cap. (pF)	Max. skew (ps)
r1	267	8	11.38	4.28
r2	598	30	46.28	6.04
r3	862	30	70.72	18.18
r4	1903	50	143.85	9.58
r5	3101	90	182.67	19.4

Table 1. Total interconnect capacitance and maximum skew between clock nodes within clusters calculated using HSPICE.

We also applied our clustering algorithm to recursively generate low-skew clock trees as described in Section 4.4. The resultant trees were simulated using HSPICE with active elements used for the clock tree driver and buffers. The results are shown in Table 2.

Example	No. of pins	No. of buffers	Total int. cap. (pF)	Max. skew (ps)
r1	267	8	13.03	6.6
r2	598	30	51.33	20.0
r3	862	30	75.82	33.0
r4	1903	50	153.68	39.8
r5	3101	90	198.23	47.4

Table 2. Total interconnect capacitance and the maximum skew between clock nodes of the complete buffered clock tree calculated using HSPICE.

In Figure 7 the waveforms for the two clock nodes of example r4 of Table 2 which cause the maximum skew are shown. It is apparent that the waveforms at these two clock nodes are well matched throughout the signal transition.

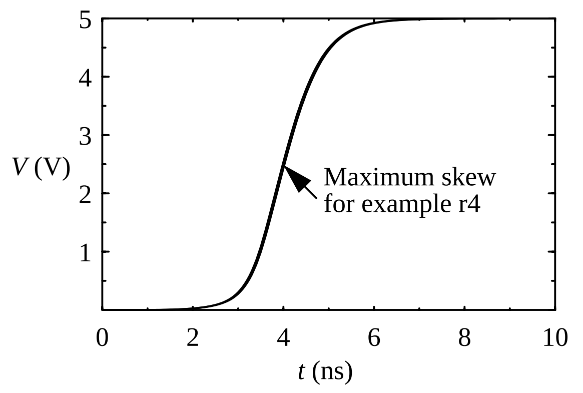


Figure 7: Waveforms at clock nodes which cause maximum skew for the complete buffered clock tree of example r4 [5] from HSPICE with active driver and buffer elements.

6 Conclusion

Buffered clock trees are often desirable, but added at the expense of complicating the clock design. In this paper, skew due to buffer mismatch is minimized by first clustering the clock nodes so that identical buffers can be used at a level, and balancing the higher-order loads of the clusters so that load dependent buffer delays are matched. Interconnect delays within clusters are concurrently balanced too, thereby generating a low-skew buffered clock tree design.

While the two techniques we have presented are most effective when used concurrently, they are completely independent of each other. The clustering technique can be used to generate clusters of equal capacitive loading for any clock tree synthesis methodology. Similarly, the delay- and admittance-matching wire sizing technique can be used for constructing any buffered clock tree that uses equally-sized buffers at the same level.

References

[1] D. W. Dobberpuhl et al., “A 200 MHz dual issue CMOS microprocessor,” *IEEE Journal of Solid State Circuits,* vol. 27, pp. 1555-1567, Nov. 1992.

[2] S. Pullela, N. Menezes and L. T. Pillage, “Low power IC clock tree design,” *Proc. Custom Integrated Circuits Conf.,* pp. 263-266, May 1995.

[3] W. C. Elmore, “The transient response of damped linear networks with particular regard to wideband amplifiers,” Journal of Applied Physics, vol. 19, no. 1, 1948.

[4] P. Penfield and J. Rubinstein, “Signal delay in RC tree networks,” *IEEE Trans. Computer-Aided Design,* vol. CAD-2, pp. 202-211, July 1983.

[5] R.-S. Tsay, “An exact zero-skew algorithm,” *IEEE Trans. Computer-Aided Design.*, vol. 12, pp. 242-249, Feb. 1993.

[6] S. Pullela, N. Menezes, J. Omar and L. T. Pillage, “Skew and delay optimization for reliable buffered clock trees,” *Proc. IEEE Intl. Conf. on Computer-Aided Design,* pp. 556-562, Nov. 1993.

[7] J. Chung and C. K. Cheng, “Skew sensitivity minimization of buffered clock trees,” *Proc. IEEE Intl. Conf. on Computer-Aided Design,* pp. 280-283, Nov. 1994.

[8] F. Minami and M. Takano, “Clock tree synthesis based on RC delay balancing,” *Proc. IEEE Custom Integrated Circuits Conf.,* May 1992.

[9] J. Qian, S. Pullela and L. T. Pillage, “Modeling the *effective capacitance* for the RC interconnect of CMOS gates,” *IEEE Trans. Computer-Aided Design.*, vol. 13, no. 12, pp. 1526-1535, Dec. 1994.

[10] P. R. O’Brien. and T. L. Savarino, “Modeling the driving-point characteristic of resistive interconnect for accurate delay estimation,” *Proc. IEEE Intl. Conf. on Computer-Aided Design,* Nov. 1989.

[11] M. Edahiro, “Clustering-based optimization algorithm in zero-skew routings,’’ *Proc. ACM/ IEEE Design Automation Conference*, pp. 612-616, June 1993.

[12] D.F. Wong and C.L. Liu, “A new algorithm for floorplan design,’’ *Proc. 23rd Design Automation Conference,* pp. 101-107, June 1986.

[13] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, The MIT Press, 1991.

[14] N. Gopal, D. P. Neikirk, and L. T. Pillage, “Evaluating RC interconnect using moment-matching approximations,” *Proc. IEEE Intl. Conf. on Computer-Aided Design,* pp. 74-77, Nov. 1991.

[15] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization,* Academic Press, 1981.

[16] Q. Zhu, W. W.-M. Dai, and J. G. Xi, “Optimal sizing of high-speed clock networks based on distributed RC and lossy transmission line models,” pp. 628-633, *Proc. International Conference on Computer Aided Design,* Nov. 1993.

[17] K. Schittkowski, “NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems,” *Annals of Operation Research,* vol. 5, pp. 485-500, 1985/86.

[18] L. T. Pillage and R. A. Rohrer, “Asymptotic waveform evaluation for timing analysis,” *IEEE Trans. Computer-Aided Design,* vol. 9, no. 4, pp. 352-366, April 1990.

[19] N. Menezes, S. Pullela, and L.T. Pileggi, “A sequential quadratic programming approach to gate and wire sizing,” *Proc. IEEE Intl. Conf. on Computer-Aided Design,* pp. 144-151, Nov. 1995.