Wire Segmenting for Improved Buffer Insertion

Charles Alpert and Anirudh Devgan

IBM Austin Research Laboratory, Austin, TX 78758

Abstract

Buffer insertion seeks to place buffers on the wires of a signal net to minimize delay. Van Ginneken [14] proposed an optimal dynamic programming solution (with extensions proposed by [7] [8] [9] [12]) such that at most one buffer can be placed on a single wire. This constraint can hurt solution quality, but it may be circumvented by dividing each wire into multiple smaller *segments*. This work studies the problem of finding the correct number of segments for each wire in the routing tree. Too few segments yields sub-par solutions, but too many segments can lead to excessive run times and memory loads. We derive new theoretical results for computing the appropriate number of buffers (and hence wire segments) which motivate our new wire segmenting algorithm. We show that using wire segmenting as a precursor to buffer insertion produces solutions within a few percent of optimal, while using only seconds of CPU time.

1 Introduction

The scaling of process technology and device and interconnect size has led timing optimization techniques for VLSI circuits to become increasingly critical. Three techniques are commonly applied to reduce the delay of an existing topology: gate sizing, wire sizing, and buffer insertion, and this work focuses on the last technique. One can reduce net delays by inserting buffers that either decouple a large load that is off the critical path or directly reduce the ARC delay of a long wire. We assume that the tree topology is fixed and that the wire resistances and capacitances have been extracted.

Early works in buffer insertion [1] [13] sought solutions in which the tree topology was not necessarily fixed. Berman et al. [1] showed that simultaneously constructing a tree and placing buffers at the internal nodes of the tree is NP-Complete. The authors of [13] proposed a heuristic buffer insertion algorithm based on a linear delay model. The works [6] [10] perform buffer insertion by finding the best location for a *single* buffer and then recursively applying the algorithm. Hedenstierna and Jeppson [5] studied placing consecutive buffers to optimize their SPICE-based delay model. The algorithm of Lowe and Gulak [11] alternates buffer insertion with buffer sizing. Chen et al. [2] use Lagrangian relaxation to size pre-placed buffers. Finally, Dhar and Franklin [3] gave closed-form solutions for buffer insertion with multiple sizes on a single uniform line. Their work is similar in spirit to the theoretical portion of this work, but their formulation assumes that a series of resizable buffers drives the wire which leads to the conclusion that all buffers are equally spaced.

Design Automation Conference ®

0-89791-847-9/97/0006/$3.50 DAC 97 - 06/97 Anaheim, CA, USA

Our work is an extension of Van Ginneken's dynamic programming algorithm [14] which performs *optimal* buffer placement such that only one buffer may be placed on each wire, and the library contains only a single, non-inverting buffer. Van Ginneken [14] also outlined a non-polynomial extension for minimizing the total number of buffers. The optimality of this algorithm has inspired numerous variants. Lillis et al. [7] simultaneously perform wire sizing and buffer insertion using a library that contains both inverting and non-inverting buffers. The works [8] [9] extended these ideas further by integrating slew into the delay model and by minimizing a *power* function (e.g., the total number of buffers), while retaining optimality. Finally, Okamoto and Cong [12] integrated Van Ginneken's algorithm into a simultaneous Steiner tree and buffer insertion construction.

Works based on Van Ginneken's algorithm are optimal only under the condition that at most one buffer may be placed on each wire. This constraint will severely restrict the solution space when multiple buffers are required to effectively drive wires with large lumped RCs. However, this restriction can easily be circumvented by *wire segmenting*, the principle of which is shown for the 3-pin net in Figure 1(a). The net's *source* is represented by a black square, and the *sinks* are represented by white squares. The gray nodes show the possible locations where buffers may be inserted in Van Ginneken's algorithm, i.e., at the ends of any of the wires in the routing tree. Figure 1(b) shows the same tree after segmenting each wire into four smaller wires, thereby introducing 12 new possible buffer locations. Segmenting each wire into arbitrarily small wires completely eliminates the one buffer per wire restriction; however, the time complexity and memory requirements of Van Ginneken's algorithm becomes exorbitant. The question that this work seeks to answer is, "What is the ideal number of segments into which each wire should be divided?" We make the following contributions.

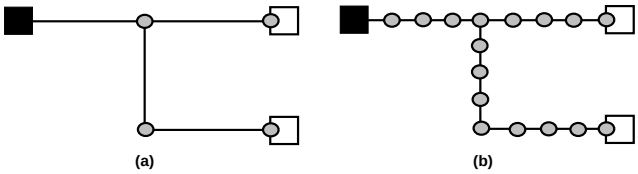


Figure 1: (a) A 3-pin net with 4 wires and (b) the same net after wire segmenting with 16 wires.

- We derive new theoretical results which illustrate the correct number of buffers that should be placed on a wire of fixed length for a given technology. In contrast to Dhar and Franklin [3], we do not allow resizing of the driver and sink, and this assumptions leads to the different conclusion that buffer spacing is non-uniform and depends on the driver size and sink capacitance.
- We utilize these results within a new wire segmenting algorithm which has been incorporated into the buffer insertion algorithm of [9].

- Our experiments show that wire segmenting leads to significantly lower delays than no wire segmenting. Further, on average our algorithm produces solutions within 4% of optimal while using only a few seconds of CPU time, which is orders of magnitude more efficient than the time needed to compute an optimal solution.

The remainder of the paper is as follows. Section 2 defines the buffer insertion problem. Section 3 overviews the basic dynamic programming paradigm [9] [14]. Section 4 presents theoretical results that determine the appropriate number of buffers that should be inserted on a given wire. Section 5 describes our wire segmenting algorithm. Section 6 presents our experimental results, and Section 7 gives directions for future work.

2 Problem Formulation

We assume that the routing tree topology has been fixed or that an initial Steiner estimation is available for the given a net. A routing *tree* $T = (V, E)$ consists of a set of n *nodes* V and a set of $n-1$ *wires* E. We write V as $\{\{so\} \cup SI \cup IN\}$ where *so* is the unique *source* node, *SI* is the set of *sink* nodes and *IN* is the set of *internal* nodes. A wire $e \in E$ with length l_e is an ordered pair of nodes $e = (u, v)$ in which the signal propagates from u to v. Observe that for each node $v \in SI \cup IN$, there is a unique *parent wire* $(u, v) \in E$. The tree is assumed to be *binary*, i.e., each node can have at most two children. A non-binary tree can be converted into an equivalent binary tree by inserting wires with zero resistance and capacitance where appropriate. Let the left and right children of v be denoted by $left(v)$ and $right(v)$ respectively. If v has no left (right) child, we write $left(v) = \emptyset$ ($right(v) = \emptyset$). We are also given a buffer library B of size m which consists of inverting and non-inverting buffers $b_1, \ldots, b_m$.

Following [9] [12] [14] , we adopt the Elmore delay model [4] for interconnect delays and a linear model for gate delays. For each gate v, let C_v denote the input capacitance, R_v the intrinsic resistance and K_v the intrinsic delay of v. The lumped capacitance and resistance for each wire $e \in E$ are given by C_e and R_e respectively.

Let $T(v)$ denote the subtree rooted at v. The *load* at node v is given by the total lumped capacitance $C_{T(v)}$ of $T(v)$. If $T(v) = (\{v\} \cup SI' \cup IN', E')$ then

$$C_{T(v)} = \sum_{u \in SI'} C_u + \sum_{e \in E'} C_e.$$

The Elmore delay for the wire $e = (u, v)$ is given by

$$Delay(e) = R_e(\frac{C_e}{2} + C_{T(v)}).$$

The lumped RC model preserves the property that the Elmore delay is the same for a given wire no matter how the wire may be subdivided.

The delay through a gate $v \in \{so\} \cup B$ is a linear function of the load at v:

$$Delay(v) = K_v + R_v C_{T(v)}.$$

The delay from a node $v \in V$ to a sink si is

$$Delay(v\text{-}si) = \sum e = (u, w) \in path(v, si) Delay(e) + Delay(u),$$

where $path(v, si)$ is the set of edges on the path from v to si. If u is not a gate but simply an internal node, then $Delay(u) = 0$. Note that any off-path buffers decouple the load and effectively serve as "sinks" of $T(v)$, while buffers on the path from v to si serve as internal nodes with non-zero delays.

Each sink si has a required arrival time $RAT(si)$, and we assume that $RAT(so) = 0$. For the circuit to function properly, we must have $Delay(so\text{-}si) \leq RAT(si)$ for every $si \in SI$. The *slack* for every $v \in V$ is given by

$$q(v) = \max_{si \in T(v)} (RAT(si) - Delay(v\text{-}si)).$$

For the slack at v to be meaningful, buffer insertion must have already been performed on $T(v)$. Observe that this definition only looks at the subtree $T(v)$ and not at the whole routing tree. Once the slack for the source is computed, the "real slacks" for the sink nodes can then be propagated down the tree.

Buffer Insertion Problem: Given a tree $T = (\{\{so\} \cup SI \cup IN, E)$, a buffer library B, and $RAT(v)$ values for each $v \in SI$, place buffers from B on wires in E to maximize $q(so)$.

Observe that various formulations can be captured by manipulating the $RAT(si)$ values. For example, if si is the only critical sink then $RAT(w) = \infty$ for all $w \in SI - \{si\}$. Alternatively, setting all slacks to be equal captures minimizing $\max_{si \in SI} Delay(so\text{-}si)$.

3 Review of the Dynamic Programming Algorithm

The algorithm's main idea is to construct possible candidate solutions for each node in the tree. The candidates for node v only can be computed after the candidate solutions have been computed for all nodes in $T(v) - \{v\}$. A *candidate* is a 5-tuple (C, q, b, cnd_l, cnd_r) where C is the load seen at v, q is the slack at v, b is a potential buffer inserted at node v, and cnd_l and cnd_r are the candidates for the left and right children of v that were used to construct the current candidate. Given a candidate for a node v, the only information needed to compute the slack for the parent of v is the load at v, the slack at v, and the parent wire capacitance and resistance. Hence, C and q are used to percolate new candidates up the tree, and b, cnd_l, and cnd_r are only needed to recover the final solution when the algorithm terminates.

Buffer Optimization (T, B) Algorithm	
Input:	$T = (\{so\} \cup SI \cup IN, E) \equiv$ Routing tree $B \equiv$ Buffer library
Output:	$cnd \equiv$ Best candidate solution for node v
Variables:	$S \equiv$ List of candidates for the source
1. $T = Segment\_Wires(T, B)$.	
2. $S = Find\_Cnds(T, B, so)$.	
3. **for** each $cnd = (C, q, b, cnd_l, cnd_r) \in S$ **do** Set $q = q - K_{so} - R_{so}C$	
4. **return** $cnd \in S$ with maximum slack	

Figure 2: Buffer Optimization Algorithm

Figure 2 shows the Buffer Optimization algorithm which takes a routing tree and buffer library and returns a candidate solution for the source. The entire solution is revealed by recursively examining the left and right candidate solutions of the returned candidate. Step 1 performs wire segmenting, which is described in Section 5. Step 2 calls Find_Cnds (i.e., Find_Candidates) which is presented in Figure 3 (see [9] [14] for a detailed explanation).[1] It returns a set of possible candidates for the source, but without accounting for driver delay. Hence, Step 3 updates each candidate to include the driver delay, and Step 4 returns the candidate with largest slack. As noted by [9] the complexity of the buffer optimization algorithm is $O(n^2|B|^2)$.

[1] Several enhancements [9] can be made to the Find_Cnds procedure: (i) inverters can be incorporated by maintaining two candidate lists, instead of one; (ii) one can optimize some "power" function of the buffers in the solution, e.g., the total number of buffers; (iii) signal slew can be integrated into the gate delay model, although time complexity increases; and (iv) wire sizing can be integrated by viewing each possible wire size as a choice from a "wire library".

Find_Cnds (T, B, v) Procedure

Input: $T = (\{so\} \cup SI \cup IN, E) \equiv$ Routing tree
$B \equiv$ Buffer library
$v \equiv$ Current node to be processed
Output: $S \equiv$ List of candidate solutions for node v
Variables: $S_b, S_l, S_r \equiv$ Temporary lists for candidates
$cnd, cnd_l, cnd_r \equiv$ Candidate solutions
$e \equiv$ Parent wire of v

$S = S_b = \emptyset$.
1. **if** $v \in SI$ **then**
   $S = \{(C_v, RAT(v), 0, 0, 0)\}$.
2. **else if** $left(v) = \emptyset$ **then**
   for each $cnd = (C, q, b, cnd_l, cnd_r) \in Find\_Cnds(T, B, right(v))$ **do**
   $S = S \cup (C, q, 0, 0, cnd)$
3. **else if** $right(v) = \emptyset$ **then**
   for each $cnd = (C, q, b, cnd_l, cnd_r) \in Find\_Cnds(T, B, left(v))$ **do**
   $S = S \cup (C, q, 0, cnd, 0)$
4. **else**
5. $S_l = Find\_Cnds(T, B, left(v))$. $S_r = Find\_Cnds(T, B, right(v))$.
6. Set $i = 1$ and $j = 1$.
7. **while** $i \leq |S_l|$ and $j \leq |S_r|$ **do**
8. Let $cnd_l = (C_l, q_l, \ldots)$ be the i^{th} candidate in list S_l.
   Let $cnd_r = (C_r, q_r, \ldots)$ be the j^{th} candidate in list S_r.
9. $S = S \cup \{(C_l + C_r, min(q_l, q_r), 0, cnd_l, cnd_r)\}$.
10. **if** $q_l \leq q_r$ **then** $i = i + 1$.
    if $q_r \leq q_l$ **then** $j = j + 1$.
11. **if** v is a feasible buffer location **then**
12. **for** each buffer $b \in B$ **do**
    Find $(C, q, b, cnd_l, cnd_r) \in S$ that maximizes $q - K_b - R_b C_b$.
    $S_b = S_b \cup \{(C_b, q - K_b - R_b C_b, b, cnd_l, cnd_r)\}$.
13. $S = S \cup S_b$.
14. **for** each $cnd = (C, q, b, cnd_l, cnd_r) \in S$ **do**
    $S = S \cup \{(C + C_e, q - R_e(\frac{C_e}{2} + C), cnd_l, cnd_r)\} - cnd$.
15. Prune S of inferior solutions.
16. Store S as candidate list for v and **return S**

Figure 3: Find_Candidates Procedure

4 Theoretical Results

The above algorithm is optimal under the condition that a buffer can only be placed at the end of a wire. We can avoid this by dividing each wire into arbitrarily small segments, but this may significantly increase CPU times. Our goal is to enable the designer to automate buffer insertion for several thousand nets. Even if the CPU time resulting from arbitrarily small segmenting of wires is reasonable for a single net, it will not be for a task of this magnitude.

We now analytically explore the problem of finding the optimal number of buffers to insert on a given wire. Consider inserting a single buffer on a net with two pins connected by a single wire (see Figure 4(a)). In this case $T = (\{so\} \cup \{si\} \cup \emptyset, \{e\})$ where si is the single sink node and $e = (so, si)$. Let $R = \frac{R_e}{l_e}$ and $C = \frac{C_e}{l_e}$ be the unit resistance and capacitance for wire e respectively, and let $D_k = delay(so\text{-}si)$ with k buffers placed optimally on e.

Theorem 1 Given $T = (\{so\} \cup \{si\} \cup \emptyset, \{e\})$ and a buffer b, $D_1 < D_0$ if and only if

$$l_e > \frac{R_b - R_{so}}{R} + \frac{C_b - C_{si}}{C} + 2\sqrt{\frac{R_b C_b + K_b}{RC}}, \qquad (1)$$

i.e., it is worthwhile to insert at least one buffer on e.

Proof: The delay D_0 is the sum of the wire delay $Rl_e(\frac{Cl_e}{2} + C_{si})$ plus the delay of the source $R_{so}(Cl_e + C_{si}) + K_{so}$.

$$D_0 = R_{so}(Cl_e + C_{si}) + K_{so} + Rl_e(\frac{Cl_e}{2} + C_{si})$$

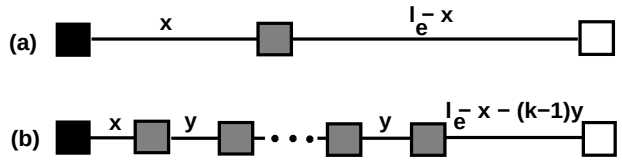


Figure 4: (a) The placement of a single buffer at distance x from the source and (b) the placement of k buffers each separated by distance y.

Let x be the distance that buffer b is placed from the source so that D_1 is minimized. The delay D_1 is the sum of the delays of the two wire segments plus the source and buffer delays:

$$\begin{aligned} D_1 &= R_{so}(Cx + C_b) + K_{so} + Rx(\frac{Cx}{2} + C_b) + K_b \qquad (2) \\ &\quad + R_b[C(l_e - x) + C_{si}] + R(l_e - x)(\frac{C(l_e - x)}{2} + C_{si}) \end{aligned}$$

Setting the derivative of D_1 with respect to x to 0 and solving for x yields

$$\frac{dD_1}{dx} = R_{so}C + RCx + RC_b - R_bC - RC(l - x) - RC_{si} = 0$$

$$x = \frac{l_e}{2} + \frac{R_b - R_{so}}{2R} + \frac{C_{si} - C_b}{2C} \qquad (3)$$

Thus, if only a single buffer may be placed on a given wire, its optimal location is at distance $\frac{l_e}{2} + \frac{R_b - R_{so}}{2R} + \frac{C_{si} - C_b}{2C}$ from the source. Substituting this value for x into Equation (2) gives us:

$$\begin{aligned} D_1 &= \frac{RCl_e^2}{4} + K_{so} + K_b + \frac{l_e}{2}[C(R_{so} + R_b) + R(C_b + C_{si})] \\ &\quad + \frac{(R_{so} + R_b)(C_b + C_{si})}{2} - \frac{(R_{so} - R_b)^2 C}{4R} - \frac{R(C_b - C_{si})^2}{4C} \end{aligned}$$

Clearly, it is worthwhile to insert a buffer only if the delay is reduced by the buffer's insertion, i.e., if $D_0 - D_1 > 0$.

$$\begin{aligned} 0 &< D_0 - D_1 \\ &= l_e^2 + 2l_e(\frac{R_{so} - R_b}{R} + \frac{C_{si} - C_b}{C}) - \frac{4K_b}{RC} - \frac{(R_{so} - R_b)^2}{R^2} + \\ &\quad \frac{2R_{so}(C_{si} - C_b) - 2R_b(C_b + C_{si})}{RC} + \frac{(C_b - C_{si})^2}{C^2} \end{aligned}$$

which is a quadratic in l_e. Solving for l_e using the quadratic formula gives us Equation (1). □

Theorem 1 gives a threshold value for the wire length for inserting one buffer. We now extend this result to the insertion of k buffers. First, we need the following corollaries:

Corollary 1 Given $T = (\{b\} \cup \{b\} \cup \emptyset, \{e\})$, i.e., a 2-pin net with source b and sink b connected by a single wire, the optimal location for the placement of a buffer b on the wire is half-way between the source and sink (apply Equation (3) with $so = b$ and $si = b$).

Corollary 2 Given $T = (\{b\} \cup \{b\} \cup \emptyset, \{e\})$, the optimal placement of k buffers of type b is to space them at equal increments of $\frac{l_e}{k+1}$ on the wire.

Consider a non-equally spaced solution. Then there must exist a buffer b that lies closer to either its predecessor b_1 or its successor b_2. But then b_1 can be viewed as the driver for a net with sink b_2,

so the optimal location for b must be equidistant between b_1 and b_2 by Corollary 1.[2]

Theorem 2 Given $T = (\{so\} \cup \{si\} \cup \emptyset, \{e\})$ and a buffer b, $D_k < D_{k-1}$ if and only if

$$l_e > \frac{C_b - C_{si}}{C} + \frac{R_b - R_{so}}{R} + \sqrt{\frac{2k(k+1)(K_b + R_b C_b)}{RC}} \quad (4)$$

i.e., it is worthwhile to place at least k buffers on e.

Proof: See Figure 4(b). Let x be the distance between the source and the first buffer and let y be the distance between two consecutive buffers. From Corollary 2, one concludes that the buffers are equally spaced from each other (but not necessarily from the source and sink). The optimal delay for the placement of k homogeneous buffers on e is given by

$$\begin{aligned} D_k &= R_{so}(Cx + C_b) + K_{so} + \frac{1}{2}RCx^2 + RxC_b + \quad (5) \\ &\quad (k-1)[R_b(Cy + C_b) + K_b + \frac{1}{2}RCy^2 + RyC_b] + \\ &\quad R_b(C(l_e - x - (k-1)y) + C_{si}) + K_b + \\ &\quad \frac{1}{2}RC(l_e - x - (k-1)y)^2 + R(l_e - x - (k-1)y)C_{si} \end{aligned}$$

Equation (5) is a quadratic in x and y and is clearly convex. Hence, D_k has a unique global minimum which can be found by setting $\frac{\partial D_k}{\partial x} = 0$ and $\frac{\partial D_k}{\partial y} = 0$ and solving for x and y.

$$x = \frac{l_e - (k-1)y}{2} + \frac{R_b - R_{so}}{2R} + \frac{C_{si} - C_b}{2C}, \quad y = \frac{l_e - x}{k} + \frac{C_{si} - C_b}{kC}$$

Combining these equations yields:

$$\begin{aligned} x &= \frac{1}{k+1}(l_e + \frac{k(R_b - R_{so})}{R} + \frac{C_{si} - C_b}{C}) \\ y &= \frac{1}{k+1}(l_e - \frac{R_b - R_{so}}{R} + \frac{C_{si} - C_b}{C}) \end{aligned}$$

Now that we have the optimal buffer locations, we can substitute these x and y values into Equation (5) to obtain the optimum delay explicitly in terms of known quantities:

$$\begin{aligned} D_k &= \frac{Rl_e(kC_b + C_{si}) + Cl_e(R_{so} + kC_b) + (kC_b + C_{si})(kR_b + R_{so})}{k+1} \\ &\quad + K_{so} + kK_b + \frac{(RCl_e^2 + \frac{kR(C_b - C_{si})^2}{C} - \frac{kC(R_b - R_{so})^2}{R})}{2(k+1)}. \quad (6) \end{aligned}$$

In the proof of Theorem 1, a buffer was worthwhile inserting if $D_0 > D_1$. Similarly, k buffers are preferred to $k-1$ buffers if $D_{k-1} > D_k$. D_{k-1} can be derived by substituting $k-1$ in for k in Equation (6). The result of simplifying $D_{k-1} > D_k$ is:

$$(RCl_e + R(C_{si} - C_b) + C(R_{so} - R_b))^2 > 2k(k+1)RC(K_b + R_b C_b)$$

Solving for l_e gives the desired result. □

Despite using a two-variable analysis for the proof of Theorem 2, we observe that Theorems 1 and 2 are consistent, which makes Theorem 2 supersede Theorem 1. We can now compute the optimum number of buffers k for a given wire:

[2]Note that our results are similar to [3] in that they conclude that buffers should be equally spaced. However, their formulation assumed that the driving and sink buffers were resizable, while we assume a fixed driver and sink in the next theorem. This yields a solution in which all nodes are *not* equally spaced.

Theorem 3 Given $T = (\{so\} \cup \{si\} \cup \emptyset, \{e\})$ and a buffer b, the number of buffers k which minimizes the $Delay(so\text{-}si)$ is

$$k = \lfloor -\frac{1}{2} + \sqrt{1 + \frac{2(RCl_e + R(C_b - C_{si}) - C(R_b - R_{so}))^2}{RC(R_b C_b + K_b)}} \rfloor \quad (7)$$

Proof: The optimum number of buffers can be determined by finding k such that Theorem 2 holds for k but not for $k+1$. We can rewrite Equation (4) as the following quadratic in k

$$2k^2 + 2k - \frac{RC(l_e - \frac{C_b - C_{si}}{C} - \frac{R_b - R_{so}}{R})^2}{K_b + R_b C_b} < 0.$$

Solving for k yields

$$k < -\frac{1}{2} + \sqrt{1 + \frac{2(RCl_e + R(C_b - C_{si}) - C(R_b - R_{so}))^2}{RC(R_b C_b + K_b)}}$$

and the optimal value is obtained by finding the maximum k such that the above equation holds. □

5 The Wire Segmenting Algorithm

The above analysis assumes that the tree has only one wire and that the buffer library has only one buffer. Varying tree topologies and multiple buffer libraries make it more difficult to find closed form solution for the exact number of buffers, so we attempt to estimate it while preferring to err on the side of too many buffers. The cost of underestimation could be an inferior solution while the cost of overestimation is additional CPU time.

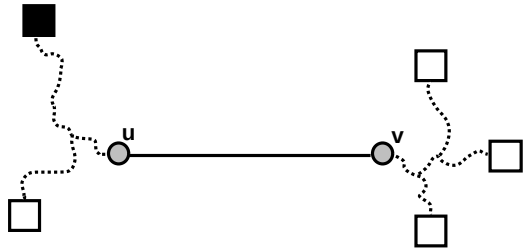


Figure 5: Isolation of a single wire (u,v) in a tree. The upper bound on the load at v is $C_{T(v)}$.

Consider each wire $e = (u,v)$ individually as part of a routing tree (see Figure 5). If buffer insertion has already been performed for the other wires, then the previous analysis can be applied to wire e. The subtree at v can be replaced by a single sink with capacitance $C_{T(v)}$ and all off-path subtrees can be replaced by their loads. Since the load may become decoupled by buffer insertion, we compute Equation (7) for the extreme cases $C_{si} = T(v)$ and $C_{si} = 0$ and pick the one which yields the higher k value. We tried computing Equation (7) for drivers from the buffer library and found that the source generally yields the highest value for k; hence, the original source is assumed as the driver for every wire. We compute Equation (7) for each $b \in B$ and set k to be the highest resulting value.

Figure 6 presents our wire segmenting algorithm. It accepts an unbuffered tree T and a buffer library B as inputs and returns a modified tree with segmented wires. Step 1 computes the load capacitance for each subtree by recursively propagating sink capacitances up the tree. Step 2 iterates through all the wires $(u,v) \in E$. Steps 4-6 compute Equation (7) for each buffer in B and for the cases $C_{si} = 0$ and $C_{si} = C_{T_v}$ and stores the largest value for k. The value of k seen in Step 7 is our estimate for the number of buffers

that might be needed for wire e. Steps 7-9 then create $k+1$ new nodes and $k+1$ wire segments of length $\frac{l_e}{k+1}$. We use $k+1$ instead of k to allow buffer insertion just prior to node v (in order to decouple a very large load); a zero length wire is added between v and v_0.[3] The complexity of the procedure is $O(n|B|)$ assuming that the maximum value for k is a constant (which must hold for any given technology).

Segment_Wires (T, B) Procedure

Input: $T = (\{so\} \cup SI \cup IN, E) \equiv$ Routing tree
$B \equiv$ Buffer library
Output: $T \equiv$ Modified routing tree with segmented wires.

1. **for** each $v \in V$ **do** compute $C_{T(v)}$.
2. **for** each wire $e = (u,v) \in E$ **do**
3. Let $R = \frac{R_e}{l_e}$ and $C = \frac{C_e}{l_e}$. Set $k = 0$.
4. **for** each $b \in B$ **do**
5. Set $k_1 = -\frac{1}{2} + \sqrt{1 + \frac{2(RCl_e + R(C_b - C_{T(v)}) - C(R_b - R_{so}))^2}{RC(R_bC_b + K_b)}}$
Set $k_2 = -\frac{1}{2} + \sqrt{1 + \frac{2(RCl_e + RC_b - C(R_b - R_{so}))^2}{RC(R_bC_b + K_b)}}$
6. $k = \lfloor \max(k, k_1, k_2) \rfloor$
7. Create $k+1$ new nodes $v_0, v_1, \ldots, v_k$ and add them to IN. Let $v_{k+1} = u$.
8. Add wire (v_0, v) with capacitance and resistance 0 to E.
9. **for** $i = 0$ **to** k **do**
Create wire $e' = (v_{i+1}, v_i)$ with $l_{e'} = \frac{l_e}{k+1}$, $R_{e'} = \frac{R_e}{k+1}$, and $C_{e'} = \frac{C_e}{k+1}$. Add e' to E.
10. **return** T.

Figure 6: Segment_Wires Procedure

6 Experimental Results

Our algorithm was implemented in C++ on an IBM RS6000/390. We changed Step 6 in Figure 6 to $k = \lfloor M \cdot \max(k, k_1, k_2) \rfloor$, where $M \geq 0$ is a parameter used to test various degrees of wire segmenting. Thus, we tested the following cases by simply changing the value of M: (i) $M = 0$ corresponds to no wire segmenting; (ii) $M = 1$ corresponds to our proposed wire segmenting scheme; (iii) a very large value of M corresponds to the optimal solution. We have observed that $M = 30$ is sufficiently large. Our experiments show that using $M = 1$ yields solutions that are within a few percent of optimal, while using significantly less CPU time than $M = 30$.

All of our test cases were routed on two layers using parasitics from a current technology. We set all sink slacks to be equal to minimize the maximum delay. Our buffer library consisted of 9 inverting and 3 non-inverting buffers. The intrinsic delays, resistances, and input capacitances for the buffers ranged from 6.83 - 120 ps, 0.044 - 0.752 Ω, and 31.8 - 488 fF respectively. Our test cases had from 1 to 15 sinks with input capacitances ranging from 10-500 fF. The intrinsic delays and resistances of the drivers ranged from 32-37 ps and 0.077 - 1.26 Ω respectively. We studied 13 test cases based on three topologies:

1. Each of the *wc* test cases consist of two long wires that connect a single sink to the source; the lengths of the wires vary between 7 and 14 mm.

2. Each of the *cone* test cases contains between 3 and 5 sinks. These cases represent the typical instance in terms of the number of sinks and total wirelength for which buffer insertion will be applied.

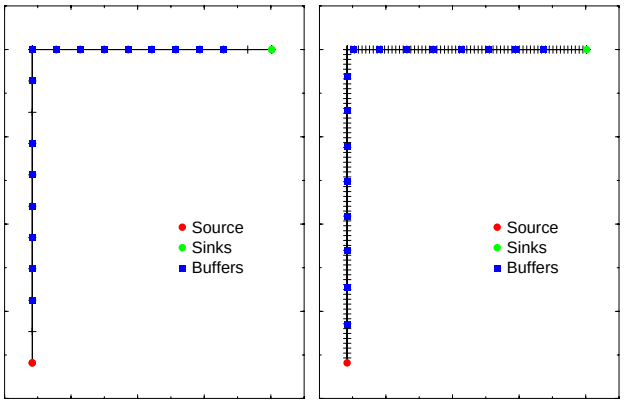
[3]The length of the "zero" wire can be set to some nominal length to satisfy minimum gate spacing requirements.

3. Each of the *star* test cases contain 15 sinks that span a large portion of the chip. These cases comprise some of the more complex instances for which our algorithm will be applied.

Test Case	Elmore Delay (ps)			# Buffers			CPU(s)	
	0	1	30	0	1	30	1	30
wc1.3	3388	1321	1217	1	8	7	0.5	302
wc2.3	1651	1119	1099	2	8	8	0.4	48
wc3.3	1696	1133	1116	2	8	8	0.3	49
wc1.4	5733	1889	1724	1	14	12	0.9	486
wc2.4	3078	1665	1622	2	10	12	0.4	107
wc1.5	8587	2434	2265	1	17	17	1.5	861
wc2.5	5013	2189	2147	2	16	16	0.7	201
wc3.5	4819	2082	2054	1	16	16	0.7	167
cone1	278	276	275	2	2	2	0.5	687
cone2	487	411	409	1	6	5	0.3	227
cone3	608	541	540	2	6	6	1.2	507
star1	2349	1112	1024	7	13	12	5.3	1939
star2	1255	662	610	5	8	9	3.3	10496

Table 1: Maximum source-sink delays, total number of inserted buffers and CPU times obtained by the Buffer Optimization algorithm for $M = 0, 1$ and 30. We used $M = 15$ instead of 30 for the star1 test case since $M = 30$ exceeded our system's memory capacity. CPU times for $M = 0$ were all less than 0.5 seconds.

We ran Buffer Optimization for each test case and for $M = 0, 1$ and 30. The Elmore delay, the total number of inserted buffers, and the required CPU time obtained for each run are reported in Table 1. We observe that the $M = 1$ delays are significantly lower than the $M = 0$ delays which shows that wire segmenting is certainly necessary. The $M = 1$ delays are 3.8% worse on average than the $M = 30$ delays, which are virtually optimal solutions. The $M = 1$ solutions can typically be computed in less five seconds while the $M = 30$ solutions can require several minutes, and require much more memory. This time savings is an enormous advantage for automating buffer insertion for thousands of nets. We successfully ran buffer insertion with $M = 1$ on a suite of 11000 nets for in under 22 minutes. For $M = 30$, the job did not terminate after working for several hours on one of the more complex nets in the suite.



(a) (b)

Figure 7: Solutions for the wc3.5 test case (a) generated using our wire segmenting scheme ($M = 1$) and (b) corresponding to the optimal solution ($M = 30$).

Note that additional buffers commonly offer only marginal delay improvements. Our implementation computes the solution with lowest delay for each possible number of buffers. For example, the $M = 1$ solutions obtained for the wc3.5 test case with 1, 2, 4, 8 and 12 buffers have delays of 4819, 3626, 2801, 2272, and 2125

picoseconds respectively. Note that even if only two buffers can be used, the delay obtained by $M = 1$ (3626) is still significantly better than the $M = 0$ solution (4819), despite $M = 0$ having two possible locations to insert buffers. The $M = 1$ delay is virtually identical to the optimal 2-buffer delay (3619) which shows that even if only a few buffers are desired, wire segmenting is still necessary, but too much wire segmenting may be wasteful.

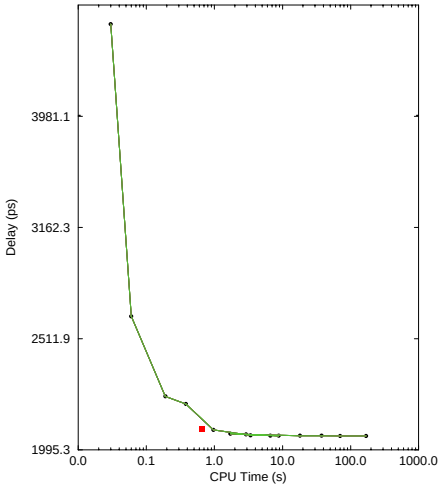


Figure 8: wc3.5 CPU/delay tradeoff.

Figures 7(a) and (b) show the locations of the inserted buffers for the wc3.5 test case. The $M = 1$ solution in (a) contains 16 buffers and was segmented to allow up to 20 buffers to be inserted.[4] The $M = 30$ solution in (b) also yields 16 buffers, but has 542 possible locations for buffers (only about one-fourth of which are shown).

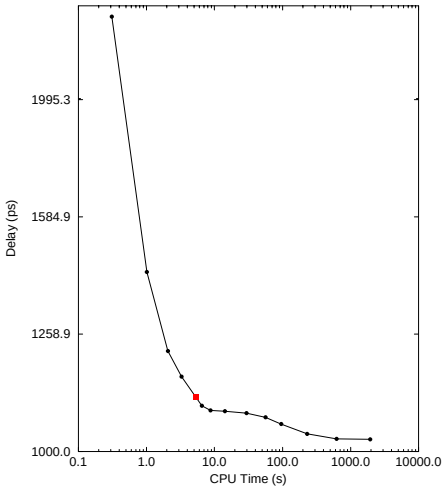


Figure 9: star1 CPU/delay tradeoff.

Finally, Figures 8 and 9 show the tradeoff between CPU time and delay that results from using different values for M. The log-log data plots shown are for the wc3.5 and star1 test cases. Each point corresponds to the solution obtained by the algorithm for a particular value of M, and the square data points correspond to $M = 1$ solutions. Note that each "square" solution is reasonably close to the "elbow" of its curve. As M increases, the wc3.5 delays decrease very slightly, but for star1, there is a fairly significant delay reduction between 100 and 500 CPU seconds.

Our wire segmenting algorithm produces a reasonably good solution on the CPU-delay tradeoff curve. If CPU time is not much of a factor, we suggest using a higher value of M in order to derive a better solution.

[4] This implies that Steps 4-6 of Figure 6 predicted 18 buffers in the final solution. Our Segment_Wires procedure typically overestimated the actual number of buffers produced in the optimal solution by 2-4 for the wc test cases. For the star and cone examples, the overestimates were much greater since the loads at subtrees were not known during the Segment_Wires procedure.

7 Conclusions

We have proposed integrating wire segmenting into the buffer insertion algorithms of [9] [14]. Our approach is motivated by new theoretical results which show the tradeoff between wire length and the optimal number of buffers needed for a 2-pin net. Experiments show that our method yields Elmore delays within 4% of optimal while using much less CPU time. Future work seeks to handle slew thresholds on the gates and to find alternative wiring strategies when large portions of the net are routed over macros that cannot permit buffer insertion.

REFERENCES

[1] C. L. Berman, J. L. Carter, and K. F. Day, "The Fanout Problem: From Theory to Practice" *Advanced Research in VLSI: Proc. 1989 Decennial Caltech Conference*, C. L. Seitz Ed., MIT Press, March 1989, pp. 69-99.

[2] C.-P. Chen, Y.-W. Chang and D. F. Wong, "Fast Performance-Driven Optimization for Buffered Clock Trees Based on Lagrangian Relaxation", *33rd IEEE/ACM Design Automation Conference*, 1996, pp. 405-408.

[3] S. Dhar and M. A. Franklin, "Optimum Buffer Circuits for Driving Long Uniform Lines" *IEEE Journal of Solid-State Circuits* 26(1), 1991, pp. 32-40.

[4] W. C. Elmore, "The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifiers", *J. Applied Physics* 19, 1948, pp. 55-63.

[5] N. Hedenstierna and K. O. Jeppson, "CMOS Circuit Speed and Buffer Optimization", *IEEE Trans. on Computer-Aided Design* 6(2), 1987, pp. 270-281.

[6] L. N. Kannan, P. R. Suaris and H.-G. Fang, "A Methodology and Algorithms for Post-Placement Delay Optimization", *31st IEEE/ACM Design Automation Conference*, 1994, pp. 327-332.

[7] J. Lillis, C.-K. Cheng and T.-T. Y. Lin "Optimal and Efficient Buffer Insertion and Wire Sizing", *IEEE Custom Integrated Circuits Conference*, 1995, pp. 259-262

[8] J. Lillis, C.-K. Cheng and T.-T. Y. Lin "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model", *IEEE/ACM International Conference on Computer-Aided Design*, 1995, pp. 138-143.

[9] J. Lillis, C.-K. Cheng and T.-T. Y. Lin "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model", *IEEE Journal of Solid-State Circuits*, 31(3), 1996, pp. 437-447.

[10] S. Lin and M. Marek-Sadowska, "A Fast and Efficient Algorithm for Determining Fanout Trees in Large Networks", *Proc. of the European Conference on Design Automation*, 1991, pp. 539-544.

[11] K. S. Lowe and P. G. Gulak, "Gate Sizing and Buffer Insertion for Optimizing Performance in Power Constrained BiCMOS Circuits", *IEEE/ACM International Conference on Computer-Aided Design*, 1993, pp. 216-219.

[12] T. Okamoto and J. Cong, "Interconnect Layout Optimization by Simultaneous Steiner Tree Construction and Buffer Insertion", *Fifth ACM/SIGDA Physical Design Workshop*, 1996, pp. 1-6.

[13] K. J. Singh and A. Sangiovanni-Vincentelli, "A Heuristic Algorithm for the Fanout Problem", *27th ACM/IEEE Design Automation Conference*, 1990, pp. 357-360.

[14] L. P. P. P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay", *Proc. International Symposium on Circuits and Systems*, 1990, pp. 865-868.