Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model

John Lillis, Chung-Kuan Cheng
Dept. of Computer Sci. & Engr.

Ting-Ting Y. Lin
Dept. of Elect. & Computer Engr.

University of California, San Diego
La Jolla, CA 92093-0114

Abstract

We present efficient, optimal algorithms for timing optimization by discrete wire sizing and buffer insertion. Our algorithms are able to minimize dynamic power dissipation subject to given timing constraints. In addition, we compute the complete power-delay tradeoff curve for added flexibility. We extend our algorithm to take into account the effect of signal slew on buffer delay which can contribute substantially to overall delay. The effectiveness of these methods is demonstrated experimentally.

1 Introduction

Timing optimization techniques for VLSI circuits have received much attention in recent years due to increasingly aggressive designs and technology trends such as shrinking geometries. This work focuses on two such techniques: wire sizing and buffer insertion.

With the advent of sub-micron geometries, wire resistance has become a significant contributor to signal delay and cycle time. However, significant improvements in delay can be achieved by selectively widening some of the wires of a routing tree. Such wire sizing is useful for both on-chip and for inter-chip (e.g. MCM) interconnects.

Previous work on wire sizing includes the works of Cong, Leung, Zhou and Koh [3, 1, 2], the work of Sapatnekar [8] and the authors previous work [7]. Cong et. al. studied the problem of minimizing the weighted sum of source-to-sink Elmore delays. Determination of weighting coefficients is presumably done by the designer. They also proposed an additive weighted term for power. Sapatnekar [8] studied the more common metric of *maximum source-to-sink delay*. He noted that the *separability property* used by Cong and Leung in designing their algorithm did not hold in this case. He proposed a geometric programming approach to the *continuous* wire-sizing problem of minimizing power for given timing constraints followed by a mapping heuristic to discretize the solution. Recently [7] gave a dynamic programming algorithm which exploited the fact that wire segment lengths are discrete in nature leading to the observation that, over all width assignments to a sub tree, the number of distinct capacitive values at the root is polynomially bounded. This yielded a polynomial time minimum delay wire sizing algorithm. However, power was not explicitly taken into account.

Buffer insertion is a powerful technique largely because of the decoupling property of buffers. Much previous research on buffer insertion (or fanout optimization) focused on construction of buffer trees before layout (for example, [10]). In contrast, [11], [7] and [6] attack the problem after layout information is available. For reasons of routability and rising interconnect delay, we propose that such an approach is preferred.

In [11] van Ginneken gave an elegant polynomial time algorithm for delay-optimal buffer insertion into a given topology. He extended his algorithm to minimize the number of buffers subject to timing constraints. Implementation details of this extension were not given and signal slew was not taken into account. In [7], a delay-optimal algorithm for simultaneous buffer insertion and wire sizing was presented. However, power and signal slew were not taken into account.

The main contributions of this paper are:

- Optimal polynomial-time algorithms for the min power wire sizing/buffer insertion problems, computation of the entire power-delay curve and an efficient data-structure for pruning sub-optimal solutions.
- Incorporation of the effect of signal slew by use of piece-wise linear functions. The slew contribution to delay can be large [5] and should not be ignored.

Timing constraints are given explicitly as required arrival times rather than as weighting coefficients of sink delays. The ability to use inverters as buffers rather than pairs of inverters to ensure proper signal polarity is also of practical interest.

2 Models and Problem Formulation

Delay Models: As in previous works, we adopt the Elmore delay model [4] for interconnects and standard RC models for buffers. For wire e, let l_e, w_e, c_e and r_e be its length, width, capacitance and resistance respectively. Further, let e_v denote the wire entering node v from its parent. We use the following basic models:

$$c_e = \alpha l_e \cdot w_e \qquad r_e = \beta l_e / w_e$$
$$\text{elmore}(e_v) = r_{e_v}(\frac{c_{e_v}}{2} + c(T_v))$$
$$\text{buf\_delay}(b, c_l) = d_b + r_b c_l$$

where α and β are constants, $c(T_v)$ is the lumped capacitance of subtree T_v rooted at v, d_b and r_b are buffer b's intrinsic delay and output resistance respectively and c_l is the load on buffer b.[1]

[1] We note that we do not rely on this model for c_e and r_e; for instance, phenomena such as fringe capacitance can be accommodated

A more accurate buffer delay model includes an additive term which is a function of the *slew* of the signal entering the buffer. One such model is:

$$\text{buf\_delay}_S(b, c_l) = \text{buf\_delay}(b, c_l) + \lambda_b D_{\text{Lprev}}. \quad (1)$$

where λ_b is a characteristic constant of b and D_{Lprev} is the RC delay of the previous stage. This and similar models have been proposed in various contexts (e.g., [9], [6]). An extension of our algorithms to accommodate this delay model is discussed in Section 6.

Problem Formulations: We adopt *required arrival time* $q(T_v)$ as our timing metric. We define $q(T_v)$ as

$$q(T_v) = \min_{u \in \text{leaves}(T_v)} (q_u - \text{delay}(v, u))$$

where q_u is the required arrival time of sink u. This measure is useful since such a scenario is typical when optimizing combinational networks. If $q(T)$ at the root is non-negative, the timing requirements are met.[2]

The dynamic power dissipation P_d for CMOS technology is $P_d = C_L V_{DD}^2 f_p$ where C_L is load capacitance and f_p is switching frequency [12]. With respect to buffer insertion and wire sizing, total capacitance is the correct measure of dynamic power dissipation since f_p and V_{DD} are fixed. This gives the problem of minimizing total capacitance C_{total} subject to $q(T) \geq 0$.

Alternatively, an attractive approach is to provide the designer with the entire power-delay trade-off curve allowing added flexibility.

3 Algorithmic Framework

We first give the framework for a high level dynamic programming algorithm into which all subsequent algorithms fit. The specific algorithms differ in their implementation of the basic routines called by the general algorithm and the contents of the sets they compute.

This General Dynamic Programming algorithm, *GDP(T,B,W)* where B is a buffer library and wire widths $1..W$ are allowed, is given as pseudo-code in Figure 1. The algorithm computes solution sets $S_{bot}(v)$ and $S_{top}(v)$. $S_{bot}(v)$ can be thought of as the set of solutions for subtree T_v, including the possibility of inserting a buffer at v. Similarly, $S_{top}(v)$ can be thought of as the set of solutions for T_v augmented by wire e_v from v's parent including possible sizing of e_v. We use $l(v)$ and $r(v)$ to denote v's left and right children. *Optimal_Soln()* computes the best overall solution (or set of solutions) by considering the driver properties.

Algorithm GDP(T,B,W)

Foreach $v \in T$ in topological order from leaves to root
  If v is a leaf
    $S_{bot}(v) \leftarrow Base\_Case(v)$
  Else
    $S_{bot}(v) \leftarrow Bottom\_Solutions(v, S_{top}(l(v)), S_{top}(r(v)))$
  If v is not the root
    $S_{top}(v) \leftarrow Top\_Solutions(v, S_{bot}(v))$
  Else /* v is the root */
    Compute $Optimal\_Soln(v, S_{bot}(v))$

Figure 1: General Algorithm Structure

[2] If $q_u = 0$ for each sink u then $max\ delay = -q(T)$.

4 Maximizing Required Arrival Time

In this case, we maximize $q(T)$ the required arrival time at the root of T under the basic RC model. To deal with signal inversion, the sets $S_{bot}(v)$ and $S_{top}(v)$ are partitioned into disjoint subsets:

$$S_{bot}(v) = S^+_{bot}(v) \cup S^-_{bot}(v)$$
$$S_{top}(v) = S^+_{top}(v) \cup S^-_{top}(v).$$

A superscript "+" indicates that we assume the incoming signal is *non-inverted*; a superscript "-" indicates that signal inversion is assumed.

The solutions themselves are *load, required-time*, or (c, q), pairs. The meaning of these sets is, for example:

$(c, q) \in S^+_{bot}(v) \Leftrightarrow$ "There exists an assignment to T_v with upward load c and required time q at v when the incoming signal is not inverted."

An important initial observation made by van Ginneken [11] is the following.

Property 4.1 For $(c, q), (c', q') \in S$, if $c' \geq c$ and $q' < q$ then (c', q') is sub-optimal.

This is clear since a larger load can only worsen delay. Thus, we always prefer smaller load and larger required time. If these sets are in increasing order of load, we have the following.

Property 4.2 Any load-required_time set S in increasing order of load, may be replaced by an $S' \subseteq S$ where S' is strictly increasing in load and required time.

We maintain this sorted order as an invariant so that we may easily exploit this property.

In the context of our algorithmic framework, we fully specify the algorithm as follows. Let c_v and q_v be the input capacitance and required arrival time of sink v respectively. First, $Base\_Case(v)$ simply sets $S^+_{bot}(v) = (c_v, q_v)$ and $S^-_{bot}(v) = \emptyset$.

Bottom_Solutions() appears in Figure 2. The algorithm computes optimal (c, q) pairs for unbuffered solutions in lines 2-13. For each achievable arrival time q, we find the smallest load achieving q. This is done in linear time in a manner similar to merging sorted lists and ensures that Property 4.2 holds (notice that i and j are strictly advancing). Next we find the optimal buffer configurations (lines 14-20) by pairing buffers b with unbuffered solutions. Finally we perform merging and additional pruning (lines 21-23). The final pruning step is also linear since the sets are in sorted order.

Top_Solutions() appears in Figure 3. We examine all pairs of widths for wire e_v and solutions (c, q) at v. Since $c' = c + wZ$ where Z is fixed and $w \in \{1..W\}$, we can visit all c''s in order without explicitly sorting them. For each such pairing, we obtain a new required

Algorithm: *Bottom_Solutions*$(v, S_{top}(l(v)), S_{top}(r(v)))$

1. /* First compute unbuffered solutions */
2. $S^+_{bot}(v) \leftarrow \emptyset$
3. **Let** $S_l = S^+_{top}(l(v))$
4. **Let** $S_r = S^+_{top}(r(v))$
5. /* S_l, S_r are indexed and ordered by c */
6. $i \leftarrow 1$; $j \leftarrow 1$
7. **While** $(i \leq |S_l|$ and $j \leq |S_r|)$
8. **Let** $(c_l, q_l) = S_l[i]$
9. **Let** $(c_r, q_r) = S_r[j]$
10. $S^+_{bot}(v) \leftarrow S^+_{bot}(v) \cup \{(c_l + c_r, \min(q_l, q_r))\}$
11. **If** $(q_l \leq q_r)$ /* Left Critical */
    $i \leftarrow i + 1$
12. **If** $(q_r \leq q_l)$ /* Right Critical */
    $j \leftarrow j + 1$
13. **Compute** $S^-_{bot}(v)$ analogously
14. /* Now compute buffered solutions */
15. **Foreach** buffer $b \in B$
16. **If** b is an inverter
17. **Find** $(c, q) \in S^-_{bot}(v)$ s.t.
    $q^+_b = q - d_b - r_b c$ is max
18. **Else**
19. **Find** $(c, q) \in S^+_{bot}(v)$ s.t.
    $q^+_b = (q - d_b - r_b c)$ is maximized
20. Analogously compute q^-_b
21. $S^+_{bot}(v) \leftarrow S^+_{bot}(v) \cup \{(c_b, q^+_b) | \forall b \in B\}$
22. $S^-_{bot}(v) \leftarrow S^-_{bot}(v) \cup \{(c_b, q^-_b) | \forall b \in B\}$
23. Prune $S^+_{bot}(v)$ and $S^-_{bot}(v)$ by Property 4.2

Figure 2: *Bottom_Solutions() for Max Required Time*

time. A final sweep applies Property 4.2 maintaining the ordered invariant for both c and q.

Finally, *Optimal_Soln*$(v, S_{bot}(v))$ simply pairs all of the previously computed *un-buffered* $(c, q) \in S^+_{bot}(v)$ with the output resistance of the driver and selects the solution with the largest required arrival time.

To construct the optimal solution we recursively revisit the tree to determine which buffering and wire-sizing choices led to the optimal solution.

Comments: For simplicity, we have presented our algorithm in terms of a binary tree, but note that the algorithm is easily applied to general trees by addition of dummy nodes and 0 length wires.

As described, the algorithm assumes exactly one sizeable wire segment between nodes and buffer insertion only at fanout nodes. When this is not the case, we can introduce artificial nodes. In such cases, *Bottom_Solutions()* constructs unbuffered solutions by copying the appropriate S_{top} set from its child.

Run-Time: We analyze the run time in three scenarios: (1) wire sizing alone ($|B| = 0$, $W > 1$), (2) buffer insertion alone ($|B| \geq 1$, $W = 1$), and (3) both methods together ($|B| \geq 1$, $W > 1$).

In scenario 1 we introduce the notion of "basic grid-length" to analyze the complexity.[3]

Algorithm: *Top_Solutions*$(v, S_{bot}(v))$

1. $S^+_{top}(v) \leftarrow \emptyset$
2. **Foreach** $(c, q) \in S^+_{bot}(v), w \in \{1..W\}$ in increasing order of $c' = c + \alpha w l_{e_v}$
3. $S^+_{top}(v) \leftarrow S^+_{top}(v) \cup \{(c', q - \text{elmore}(e_v))\}$
   /* elmore delay evaluated at width w */
4. **Compute** $S^-_{top}(v)$ analogously
5. Prune $S^+_{top}(v)$ and $S^-_{top}(v)$ by Property 4.2

Figure 3: *Top_Solutions() for Max Required Time*

Property 4.3 In scenario 1 the size of each load-required_time set S is bounded by mW where m is the total number of basic grid lengths in the tree.

This can be seen by noticing that the load at node v is determined by $\sum_{i=1}^{i=m} w_i$ which takes on integer values in $m..mW$. This bounds the the size of (c, q)-sets by bounding the number of *distinct* load values. Thus, while there is an exponential number of width assignments, there is a polynomial number of distinct resulting loads and relevant solutions. The resulting run-time is $O(nmW^2)$. When every sizeable segment is of identical length ($n = m$), we have $O(n^2W^2)$.

Scenario 2 is a generalization of the situation in [11]. Since $W = 1$, computation of S_{top} is trivial. Thus, the size of S_{bot} sets is the key factor in the run-time. We first state some properties alluded to earlier:

Property 4.4 Let S_l and S_r be the S_{top} sets of node v's left and right children respectively (of the same polarity). The following inequality holds: $|S_{bot}(v)| \leq |S_l| + |S_r| + |B|$.

Property 4.5 In scenario 2, for all load-required_time sets S, $|S| \leq n + n|B|$.

This property coupled with the fact that the merging operation is linear in $|S_l| + |S_r|$ gives an overall worst-case complexity of $O(n|B|(n + |B|n)) = O(n^2|B|^2)$.

Scenario 3 is complicated by the fact that the input capacitance of the buffers may not be simple multiples of the capacitance of a unit-length wire. However, in practice it is reasonable to assume that capacitive values can be mapped to a polynomially-bounded integer domain with sufficient precision (or are given as such). Let $c_{\max}$ be the largest capacitance possible for any individual component of the tree. Under this formulation we bound the size of the load-required_time sets by $nc_{\max}$ and the overall run-time by $O(n^2 c_{\max}(\max(W, |B|)))$. Observed run-times are typically much less than this bound.

5 Minimizing Power

We now extend the algorithm to minimize power under timing constraints. For clarity we present this extension without regard to signal polarity.

[3] We note that this analysis relies on the linear model for wire capacitance in Section 2, but the algorithm itself does not.

Solution sets $S_{bot}(v)$ and $S_{top}(v)$ now contain pairs (p, S_p) where p is power (as a capacitive value) and S_p is an ordered set of (c, q) pairs as in the previous algorithm. For example, $(p, S_p) \in S_{bot}(v)$ indicates that for power p and every $(c, q) \in S_p$ there exists an assignment for T_v consuming power p, presenting load c upward and yielding required time q at v.

The sets are ordered in increasing order of power. Each set S_p is ordered by load c as in the basic algorithm. *Base_Case*(v) sets $S_{bot}(v) = \{(c_v, \{(c_v, q_v)\})\}$ since c_v is the "power" associated with sink v.

Algorithm: *Bottom_Solutions*$(v, S_{top}(l(v)), S_{top}(r(v)))$

1. **Let** $B' = B \cup \{\phi\}$
   /* ϕ indicates "no buffer", $c_\phi = 0$ */
2. $S_{bot}(v) \leftarrow \emptyset$
3. **Foreach** triple $(p_l, S_{p_l}) \in S_{top}(l(v))$, $(p_r, S_{p_r}) \in S_{top}(r(v))$, $b \in B'$ in increasing order of $p = p_l + p_r + c_b$
4. Combine S_{p_l}, S_{p_r} as in lines 7-12 of Figure 2 to give S'
5. **If** $(b \neq \phi)$
6. Find $(c, q) \in S'$ s.t. $q' = q - \text{buf\_delay}(b, c)$ is max
7. $S' \leftarrow \{(c_b, q')\}$
8. **Else**
9. $S' \leftarrow S' - \{(c, q) \in S' | \exists (c', q') \in S_{p'}, p' < p, c' \leq c, q' \geq q\}$
10. **If** $(p, S_p) \in S_{bot}(v)$ (previous triple gave same p)
11. $S_p \leftarrow S_p \cup S'$
12. Prune S_p by property 4.2
13. **Else**
14. $S_{bot}(v) \leftarrow S_{bot}(v) \cup \{(p, S')\}$

Figure 4: *Bottom_Solutions() Routine for Low Power*

Bottom_Solutions() is given in Figure 4. We visit all possible values of total power consumption at v. These values are from both buffered and unbuffered configurations. We introduce the notion of a "non-buffer" ϕ to unify notation. We sort the values $p = p_l + p_r + p_b$. However, we observe that the number of *distinct* values p is often orders of magnitude less than the worst case (quadratic). Hence, we utilize a hash table to make an initial pass over all power values p and then sort the *distinct* values, avoiding an expensive sorting operation. *Top_Solutions*() is implemented in a similar manner in Figure 5.

These algorithms implement two types of pruning. First we prune solutions $(c, q) \in S_p$ for some power p as before by Property 4.2. An additional pruning condition is utilized in Figures 4 and 5 on lines 9 and 6 respectively and is captured in the following property:

Property 5.1 For solution (c, q) consuming power p, if $\exists$ solution (c', q') consuming power $p' < p$ where $c' \leq c$ and $q' \geq q$ then solution (c, q) is sub-optimal.

The efficient application of this property has proven essential in giving efficient running times in practice.

Algorithm: *Top_Solutions*$(v, S_{bot}(v))$

1. $S_{top}(v) = \{\{p, \emptyset\} | p \text{ is a possible power}\}$
2. **Foreach** pair $w \in 1..W$, $(p_{bot}, S_{p_{bot}}) \in S_{bot}$ in increasing order of $p = p_{bot} + \alpha w l_{e_v}$
3. **Foreach** $(c, q) \in S_{p_{bot}}$
4. $S_p \leftarrow S_p \cup \{(c + \alpha w l_{e_v}, q - \text{elmore}(e_v))\}$
   /* elmore delay evaluated at width w */
5. Prune S_p by Property 4.2
   /* $(p', S_{p'}) \in S_{top}(v)$ */
6. $S_p \leftarrow S_p - \{(c, q) \in S_p | \exists (c', q') \in S_{p'}, p' < p, c' \leq c, q' \geq q\}$
7. **If** $(S_p \neq \emptyset)$
8. $S_{top}(v) \leftarrow S_{top}(v) \cup \{(p, S_p)\}$

Figure 5: *Top_Solutions() Routine for Low Power*

Optimal_Soln$(v, S_{bot}(v))$ simply selects the lowest power un-buffered solution at the root for which $q(T) \geq 0$ when paired with the driver. Alternatively, the set of all such pairings gives the full tradeoff curve.

Detection of Property 5.1: We now detail a data-structure to efficiently determine, given $(c, q) \in S_p$ if Property 5.1 holds. Since the solution sets can grow to be of substantial size, a linear scan to detect this property would likely be a disaster.

Since we visit power values in order, we know that the entries in the data structure have power $p' < p$. Thus, the data structure need only concern itself with c and q values. The data structure should efficiently support the operations

- *insert*(c, q): update the data structure to include (c, q)
- *sub_opt*(c, q): return TRUE if $\exists (c', q')$ previously inserted s.t. $c' \leq c$ and $q' \geq q$, FALSE otherwise.

These operations can be performed in $O(\log m)$ time for m entries by use of an augmented binary search tree. We order the tree by load values c. Each node t in the search tree stores a load value c_t *and* q_{l_max}, the *largest* q value in the left sub-tree.

Given this augmentation, *insert()* is easily implemented recursively and *sub_opt()* is implemented by examining the following cases with respect to c, q (given) and c_t and q_{l_max} of the current node in the tree:[4]

$c < c_t, q < q_{l\_max}$	explore left subtree
$c < c_t, q > q_{l\_max}$	return FALSE
$c > c_t, q < q_{l\_max}$	return TRUE
$c > c_t, q > q_{l\_max}$	explore right subtree

By following these rules recursively, we detect the property in time proportional to the depth of the tree.

Run Time: With respect to wire-sizing alone, i.e. $|B| = 0$, we notice that $p = c$ always holds since there is no decoupling by buffers. Thus, the basic algorithm is sufficient: we get power minimization "for free".

[4] boundary conditions are not given for clarity

In the general case of simultaneous wire sizing and buffer insertion we show a pseudo-polynomial bound. As in Section 5.1 let $c_{\max}$ be the largest possible capacitive value of any component. We bound the number of (c, q) pairs at a node by $(nc_{\max})^2$ (since the same c may appear with different p's). This gives an overall bound of $O(n(|B|+W)(nc_{\max})^2 \log(nc_{\max})) = O((|B|+W)(n^3 c_{\max}^2 \log(nc_{\max})))$. The log factor is an artifact of sorting the power values. In practice we observe much better run times as a result of the additional pruning described previously.

6 Accounting for Signal Slew

We now sketch a generalization of the algorithm to account for the effect of signal slew. For space considerations we do not give full pseudo-code here; a more detailed discussion is available as a technical report.

By Equation 1, buffer delay is augmented by $\lambda_b D_{\text{Lprev}}$.[5] Since the algorithm is bottom-up, this is an unknown value when computing the delay of a buffer. Conceptually, we would like to support queries of the form "What is the optimal solution at v with capacitance c *and* $D_{\text{Lprev}} = x$?"

Since $\lambda_b D_{\text{Lprev}}$ is linear in D_{Lprev} we utilize *piece-wise linear* functions. Where we previously had *load-required_time* pairs (c, q), we now have *load-required_time_func* pairs (c, f) where f is a piece-wise linear function; $f(x) = q$ is the optimal required time q at v for load c and $D_{\text{Lprev}} = x$.

Figure 6 illustrates this modeling. The delay at node v in 6a is modeled by the piece-wise linear function in 6b. The left and right subtrees have maximum delays of 5 and 4 units respectively when the previous stage RC delay, $D_{\text{Lprev}} = 0$. However, since the left and right sub-trees are driven by different buffer types, they have different sensitivities λ_l and λ_r. The lines in Figure 6b correspond to the delay functions contributed by the two subtrees. The slopes correspond to the sensitivities λ_l and λ_r. The resulting delay function at node v is the max of the two (solid lines).

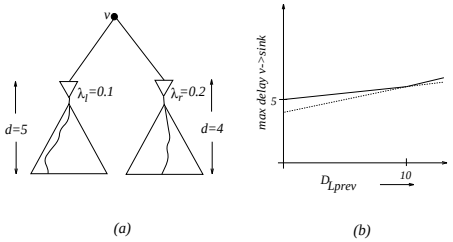

Figure 6: *Piece-wise linear model of signal slew effect*

We represent a piece-wise linear function f by a linked list of quadruples $(x_0, y_0, s, x_{\text{end}})$; each quadruple is a segment starting at point (x_0, y_0), ending at x_{end} and having slope s.

Our manipulation of piece-wise linear functions is based on the two basic operations:

$f = \text{pwl\_max}(f_1, f_2) \Leftrightarrow f(x) = \max(f_1(x), f_2(x)) \ \forall x$
$f = \text{pwl\_min}(f_1, f_2) \Leftrightarrow f(x) = \min(f_1(x), f_2(x)) \ \forall x$

These are linear time operations similar to the merging of sorted lists; we step through the lists examining points of intersection as we go.

The algorithm modifications are summarized as:

(1) Where we had load-required_time pairs (c, q), we now have load-required_time_func pairs (c, f)

(2) Where we computed scalar *max* and *min* of arrival times q, we now compute *pwl_max* and *pwl_min* of piece-wise linear functions.

(3) Where we eliminated sub-optimal solutions (c, q) by Properties 4.2 and 5.1, we now eliminate sub-optimal *portions* of solutions (c, f) by analogous properties.

Modification 3 is captured in the following property.

Property 6.1 Let (c_1, f_1) and (c_2, f_2) be solution pairs at node v (power $p_1 \leq p_2$ in the low power case). If $c_1 < c_2$ then we may replace f_2 with f_2' where

$$f_2'(x) = \begin{cases} f_2(x) & \text{if } f_2(x) > f_1(x) \\ -\infty & \text{otherwise.} \end{cases}$$

We have generalized the search tree data structure of section 6 to efficiently implement Property 6.1.

Because Property 6.1 introduces partially defined functions there is no total order on the (c, f) sets. Therefore, when combining solutions of children, we may look at all such pairs in the worst case rather than performing a simple "merge" as previously (Figure 2, lines 7-12). However, in practice these sets tend to remain linear in size. A related issue is the complexity of the functions themselves. In principle, the size of the functions can grow exponentially. However, again we do not observe this phenomenon in practice.

Figure 7 illustrates one of the algorithm's basic operations. The figure shows function f having load c before and after buffer b is considered for insertion at v.[6] Since b will drive load c, $D_{\text{Lprev}} = cr_b$ for b's descendants. Thus, the required arrival time *at the output* of the buffer is $t1$. Subtracting the buffer delay gives the required arrival time function *at the input* of the buffer shown as a solid line on the right with slope $-\lambda_b$.

With these and similar modifications, we optimally take into account the contribution of signal slew. As stated earlier, we are unable to show that the complexity of the piece-wise linear functions does not grow exponentially. However, in practice the algorithm performs comparably to its simpler counterparts.

7 Experiments and Conclusions

We implemented our algorithms in the C/UNIX environment on a Sun SPARC 20 workstation. We ran our algorithms on random routing topologies with non-uniform segment lengths. Discretization was done on a arbitrarily large integer domain (e.g. $1,000,000$), yet impressive run-times were obtained.

[5] recall that λ_b is a characteristic constant of buffer b and D_{Lprev} is the RC delay of the previous stage

[6] We are dealing with required time here (not delay), hence the negative slopes.

	K=1		K=2		K=4		K=8		K=16		K=32		MAX
n	B	S	B	S	B	S	B	S	B	S	B	S	CPU
10	1.94	1.94	1.95	1.95	1.96	1.96	1.97	1.97	2.0	2.0	2.07	2.07	0.2
15	2.64	2.64	2.65	2.65	2.66	2.66	2.87	2.70	3.19	2.77	3.90	2.87	0.8
20	2.99	2.93	3.04	2.99	3.13	3.02	3.30	3.07	3.66	3.16	4.37	3.47	1.3
25	3.59	3.59	3.60	3.60	3.62	3.62	3.67	3.67	3.94	3.78	4.70	3.95	3.8
30	3.42	3.39	3.49	3.40	3.63	3.42	3.91	3.46	4.47	3.56	5.59	3.73	6.8

Table 1: *Basic Algorithm vs. Slew Algorithm for Various Sensitivities*

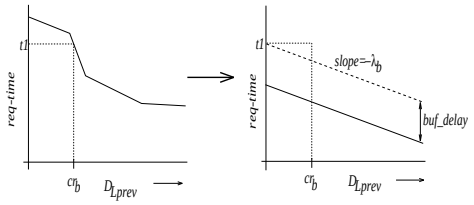

Figure 7: *Effect of inserting a buffer*

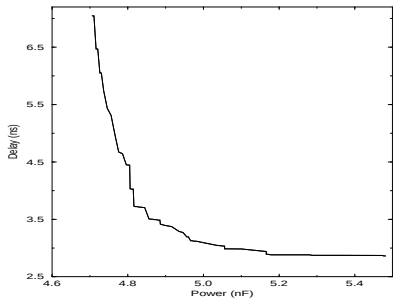

Figure 8: *Power-Delay Curve for a 20 sink net*

Since we derive optimal solutions, experiments focused on run-time, the nature of the trade-off curves and the effect of signal slew.

We used 5 buffer types; the smallest (1X) having $r_b = 3170\Omega$, $c_b = 10$fF, $d_b = 300$ps and $\lambda_b = .08$. The largest buffer was 8X. Intrinsic delay d_b was identical for all buffers and λ_b was assumed to be inversely proportional to width (largest λ_b being 0.08, smallest 0.01). We used wire widths from $0.5\mu m$ to $5\mu m$.

Figure 8 shows the optimal power vs. delay curve for a 20 sink net with both wire sizing and buffer insertion. Observed run times for nets of this size are typically in the 20-30 second range. The unsized/unbuffered delay is the left-most point and the minimum delay solution is the right-most point of the curve. Clearly good engineering choices appear at the "elbow" of the curve.

Our second set of experiments in Table 1 examined the effect of slew. We performed experiments on nets with 10 to 30 sinks. We introduce a scaling factor K as a coefficient for λ_b, replacing λ_b with $K\lambda_b$. For each K we have two columns: B is the min delay result of the basic algorithm evaluated with slew taken into account and S is the result of the extended algorithm of section 6. The right-most column is the worst run-time among all experiments in that row.[7] As K and n grow we see large variation between observed delays, approaching 50% in one case. The purpose of this table is to give an idea of the trend as λ_b increases.[8]

To conclude we would like to thank Matthew Clegg of UC San Diego and Sachin Sapatnekar of Iowa State University for several helpful discussions. We also thank the reviewers for several helpful suggestions.

References

[1] J.J. Cong, K.S. Leung, "Optimal Wiresizing Under Elmore Delay Model," *IEEE Trans. on CAD,* v. 14 no. 3 (1995) pp. 321-336.

[2] J.J. Cong, C.-K. Koh, "Simultaneous driver and wire sizing for performance and power optimization" *IEEE Transactions on VLSI Systems,* Dec. v. 2 no. 4 (1994) pp 408-25

[3] J.J. Cong, K.S. Leung, D. Zhou, "Performance-driven interconnect design based on distributed RC delay model," *Proc. ACM/IEEE Design Automation Conf.,* 1993 pp. 606-611

[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 Transactions on Computer-Aided Design,* Mar. 1987, pp 270-281.

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

[7] J. Lillis, C.K. Cheng, T.T. Lin, "Optimal and Efficient Buffer Insertion and Wire Sizing," *Proc of Custom Integrated Circuits Conference* 1995.

[8] S.S. Sapatnekar, "RC Interconnect Optimization under the Elmore Delay Model," *Proc. ACM/IEEE Design Automation Conf.,* 1994, pp. 387-391.

[9] Synopsys 3.1 Release Manual: Appendix B, "Static Timing Analysis."

[10] H.J. Touati, "Performance-Oriented Technology Mapping," Ph.D dissertation, Memorandum No. UCM/ERL M90/109, Dept. of Electrical Engineering and computer Science, UC Berkeley, 28 November 1990.

[11] 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.

[12] N.H.E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1993, pp. 231-237.

[7] Run-times are of the low-power variant.

[8] Values for λ_b over 0.5 appear to be possible[5].