Buffer Insertion With Accurate Gate and Interconnect Delay Computation

Charles J. Alpert
IBM Austin Research Laboratory
Austin, TX 78717
alpert@austin.ibm.com

Anirudh Devgan
IBM Austin Research Laboratory
Austin, TX 78717
devgan@austin.ibm.com

Stephen T. Quay
IBM Server Group
Austin, TX 78717
quayst@austin.ibm.com

Abstract

Buffer insertion has become a critical step in deep submicron design, and several buffer insertion/sizing algorithms have been proposed in the literature. However, most of these methods use simplified interconnect and gate delay models. These models may lead to inferior solutions since the optimized objective is only an approximation for the actual delay. We propose to integrate accurate wire and gate delay models into Van Ginneken's buffer insertion algorithm [18] via the propagation of moments and driving point admittances up the routing tree. We have verified the effectiveness of our approach on an industry design.

1. Introduction

Timing optimization techniques, such as wire sizing, buffer insertion and gate sizing have gained widespread acceptance in deep submicron design. In particular, buffer insertion can reduce interconnect delay and fix slew, capacitance and noise violations while reducing power. Automated buffer insertion is becoming increasingly pervasive as the ratio of device to interconnect delay continues to decrease.

Buffer insertion has been an active area of study in recent years. Closed formed solutions have been proposed in [1][3][6] for inserting buffers on a 2-pin net. The authors of [10] insert buffers on a tree by iteratively finding the best buffer location. The authors of [4] present an algorithm for simultaneous wire sizing and buffer insertion on a 2-pin net.

Van Ginneken [18] proposed a dynamic programming algorithm which finds the optimal solution under the Elmore wire delay model and a linear gate delay model. The algorithm only permits a single, non-inverting buffer type to be considered. Several variants to this algorithm have been proposed [1][2][9][13]. Lillis et al. [9] showed how to simultaneously perform wire sizing and buffer insertion with a buffer library that contains both inverting and non-inverting buffers. They also can control the number of buffers inserted. The authors of [2] show how to incorporate noise avoidance while suffering a small delay penalty.

All variants to Van Ginneken's algorithm and most other works in buffer insertion use both simplified gate and wire delay models. The Elmore delay model often overestimates interconnect delay, and using l*umped* instead of *effective capacitance* [15] often overestimates gate delay since resistive shielding is ignored.

Consider the RC network shown in Figure 1. With a ramp input of 300 ps at A, RICE [16] predicts an A-B delay of 10 ps and an A-C delay of 697 ps. The corresponding Elmore delays are 110 ps and 1110 ps, respectively. Liu et al. [11] also observe that Elmore delay has over 100% overestimation error when compared to SPICE. The total lumped capacitance seen at A is 1100 ff, whereas for a step input, RICE gives an effective capacitance of 158 ff.

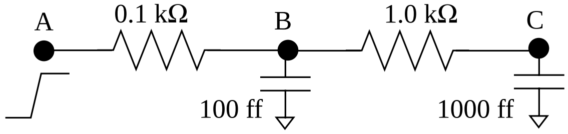

Figure 1 A simple RC network.

Simplified delay models can hurt buffer insertion algorithms in two ways. First, even optimal solutions for simple models may be inferior since actual delay is not being optimized. Second, simplified delay modeling can cause a poor evaluation of the trade-off between the total number of buffers and slack reduction.

We propose a new extension of [18] that uses both accurate interconnect and gate delay models. For interconnect delay, we compute moments via a bottom-up incremental technique [11], perform moment matching, and then compute delay using Newton-Raphson iterations. For gate delays, we store the downstream driving point admittances at each node in the tree, then propagate them up the tree using the technique of [12]. Experiments for an industry design show that the runtime penalties for using the accurate models are not prohibitive. Further, our approach produces better solutions than Van Ginneken's algorithm.

2. Preliminaries

We assume that the routing tree topology is given. A routing *tree* $T = (V, E)$ contains a set of $n-1$ *wires* E and a set of n *nodes* $V = \{\{so\} \cup SI \cup IN\}$ where so is the unique *source* node, SI is the set of *sink* nodes, and IN is the set of *internal* nodes. A wire $e \in E$ is an ordered pair of nodes $e = (u, v)$ for which the signal propagates from u to v. The lumped capacitance and resistance for wire e are denoted by C_e and R_e, respectively. Each node $v \neq so$ has 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. Let the left and right children of v be denoted by $T.left(v)$ and $T.right(v)$, respectively. If v has only one child, then it is $T.left(v)$. $B = \{b_1, b_2, \ldots, b_m\}$ denotes the buffer library.

DAC 99, New Orleans, Louisiana

A solution to the buffer insertion problem is a mapping $M: IN \to B \cup \{\bar{b}\}$ which either assigns a buffer or no buffer, denoted by $\bar{b}$, to each internal node of T. Let $|M| = |\{v \in IN: M(v) \in B\}|$ denote the number of buffers inserted by M. Assigning k buffers to T induces $k+1$ nets and $k+1$ subtrees, each with no internally placed buffers. Let $T(v) = (\{v\} \cup SI_{T(v)} \cup IN_{T(v)}, E_{T(v)})$, the *subtree rooted at* v, be the maximal subtree of T such that v is the source and $T(v)$ contains no internal buffers. Observe that if $v \in SI$, then $T(v) = (\{v\}, \varnothing)$.

Assume for now that models for gate and wire delays are given. The *path* $p(u, v)$ from node u to v is an ordered subset of wires $(u, u_1), (u_1, u_2), \ldots, (u_n, v)$ of E. A *wire path* $wp(u, v)$ from u to v is a path $p(u, v)$ such that there are no buffers assigned to nodes $u_1, u_2, \ldots, u_n$, but u and v are both gates. Let $Delay(v)$ and $Delay(wp(u, v))$ denote the gate delay through v (in which $M(v) \in B$) and wire delay through $wp(u, v)$, respectively. The total delay $Delay(p(so, si))$ from so to a sink $si \in SI$ is given by

$$\sum_{wp(u,v) \in p(so,si)} Delay(u) + Delay(wp(u,v)) . \tag{1}$$

For a given net, the *required arrival time* $RAT(si)$ for each sink si is the actual arrival time at si, minus the actual arrival time at so, plus the slack at si. The condition

$$\forall si \in SI, Delay(p(so, si)) \le RAT(si) \tag{2}$$

must hold for the net to meet its timing requirements. Our problem formulation seeks to satisfy timing constraints while minimizing the total number of inserted buffers.

Buffer Insertion Problem: Given a buffer library B and a tree $T = (\{so\} \cup SI \cup IN, E)$, find a solution M which minimizes $|M|$, such that Equation (2) holds.

Note that gate and wire delay are currently undefined, making the problem formulation general. An alternative formulation is to minimize the delay on the most critical path, i.e., $max_{si \in SI}(RAT(si) - Delay(p(so, si)))$. One problem here is that unnecessary additional buffers may be inserted, which wastes area and power.

3. Review of Van Ginneken's Algorithm (VG)

We review Van Ginneken's algorithm since it forms the basis for our new approach. The algorithm [18] cannot control the number of buffers inserted; however, we adopt the extension in [9] that addresses our problem formulation.

VG as well as [1][2][3][4][9][10][13] use the Elmore delay model [7]. Let C_{si} denote the input capacitance of each sink si. The total lumped capacitance $C_{T(v)}$ at v is given by the sum all sink and wire capacitances downstream from v. The Elmore delay $Delay(wp(u, v))$ for a wire path is $R_e((C_e/2) + C_{T(x)})$, summed over each wire $e = (w, x) \in wp(u, v)$.

Many different models are used for gate delays, but a linear model is typical. Let R_v be the intrinsic resistance and K_v the intrinsic delay of a gate v. Van Ginneken adopted the linear model, $Delay(v) = K_v + R_v C_{T(v)}$.

VG proceeds in bottom-up fashion starting at the sinks and ending at the source. The main idea is to store *candidate* solutions at each node in the tree and to propagate candidates up the tree while also generating new candidates. A *candidate* α is a 3-tuple $(C_{T(v)}, q, M)$ where $C_{T(v)}$ is the lumped capacitance seen at v, q is the *slack* at v, and M is the current solution for $T(v)$. When a node with two children is encountered, $M = M_l \cup M_r$ denotes the new solution that results from merging solutions M_l and M_r for the left and right branches of v. There are three types of nodes that are encountered.

- Sink: for this base case, a single candidate is generated.
- Node with one child: candidates for the child are copied up to the parent.
- Node with two children: candidates for the left and right children are merged using a linear pruning technique.

After the candidates for a particular node v are generated, buffer insertion at v is considered for each candidate. The one yielding the lowest slack is kept as a new candidate. Then, the delay of the parent wire for v is subtracted from the slack of each candidate. The algorithm repeats recursively until the source is encountered, at which point the driver delay is added to each candidate, and the best solution is chosen. See [2] or [9] for a complete description.

4. π-Models and Effective Capacitance

The linear gate delay model is inaccurate in two ways. First, if there is significant resistive shielding, the lumped capacitance will be much higher than the effective capacitance. Second, gate delay is not a strictly linear function of capacitance. It is more accurate to pre-characterize each gate over a range of capacitances, then perform curve-fitting. The resulting equations are called k-factor equations [15]. Errors from curve fitting will be less than for a linear delay model. To compute gate delay, we take the following three-step approach (see, e.g., [15]).

1. Compute a π-model of the driving point admittance for the RC interconnect.
2. Given the π-model and the characteristics of the driver, compute an effective capacitance C_{eff}.
3. Use C_{eff} instead of $C_{T(v)}$ when computing $Delay(v)$. Also use k-factor equations instead of a linear model.

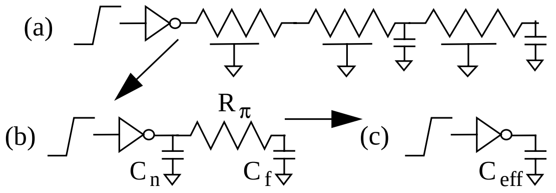

Figure 2 (a) An RC network is reduced to (b) an equivalent π-model, which is used to (c) compute effective capacitance.

Figure 2(a) shows an RC network driven by an inverter. The network can be reduced to a π-model (shown in Figure 2(b)) equivalent to the driving point admittance. A π-*model* is a 3-tuple (C_n, R_π, C_f) where C_n and C_f are the near and far capacitances and R_π is the resistance. The effective capacitance of the π-model can then be computed, e.g., using the technique of [15].

Instead of using the lumped capacitance for each candidate, we store the π-model for the downstream RC network. If the network is simply a sink si, then the π-model is $(C_{si}, 0, 0)$. To propagate π-models up the tree, we need to handle two cases shown in Figure 3.

Case 1 shows a wire e (modeled as a uniform RC line) with downstream π-model $\pi^d = (C_n^d, R_\pi^d, C_f^d)$. We apply the technique of [12] to compute the new π-model $\pi = (C_n, R_\pi, C_f)$ resulting from merging π^d with e. The corresponding procedure is shown in Figure 4.

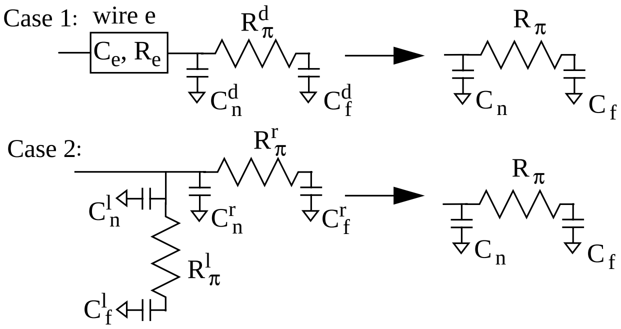

Figure 3 Two cases for updating the π-model. Case 1 is when the π-model is preceded by a uniform wire e, and Case 2 is when two π-models must be merged to form a single π-model.

```
New_$\pi$-model $(e, \pi^d)$ Procedure

Input:  $e$ ≡Wire with capacitance $C_e$, resistance $R_e$
        $\pi^d = (C_n^d, R_\pi^d, C_f^d)$ ≡ Downstream $\pi$-model
Output: $\pi = (C_n, R_\pi, C_f)$ ≡ Resulting $\pi$-model

1. $y_1^d = C_n^d + C_f^d, y_2^d = -R_\pi^d (C_f^d)^2, y_3^d = (R_\pi^d)^2 (C_f^d)^3$
2. $y_1 = y_1^d + C_e, y_2 = y_2^d - R_e\left[(y_1^d)^2 + C_e y_1^d + (C_e^2/3)\right]$
   $y_3 = y_3^d - R_e[2y_1^d y_2^d + C_e y_2^d] +$
         $R_e^2\left[(y_1^d)^3 + \frac{4}{3}C_e(y_1^d)^2 + \frac{2}{3}C_e^2 y_1^d + \frac{2}{15}C_e^3\right]$
3. Return $\pi = (y_1 - (y_2^2/y_3), -y_3^2/y_2^3, y_2^2/y_3)$
```

Figure 4 New_π-model procedure for Case 1.

Case 2 in Figure 3 shows the reduction of left and right π-models $\pi^l = (C_n^l, R_\pi^l, C_f^l)$ and $\pi^r = (C_n^r, R_\pi^r, C_f^r)$ into a single π-model $\pi = (C_n, R_\pi, C_f)$. Figure 5 shows the corresponding New_π-model procedure.

π-models are propagated up any tree by iteratively applying the appropriate New_π-model procedure. For a node v with π-model $\pi = (C_n, R_\pi, C_f)$, we will always have $C_{T(v)} = C_n + C_f$, i.e., the total lumped capacitance is preserved in the π-model, but now R_π can be used to illustrate the magnitude of resistive shielding.

5. Interconnect Delay

We now show how to accurately compute the delay for a wire path $wp(u, v)$ as opposed to using the Elmore delay. Let k be the number of moments to be used in the computation.

```
New_$\pi$-model $(\pi^l, \pi^r)$ Procedure

Input:  $\pi^l = (C_n^l, R_\pi^l, C_f^l)$ ≡ $\pi$-model for left branch
        $\pi^r = (C_n^r, R_\pi^r, C_f^r)$ ≡ $\pi$-model for right branch
Output: $\pi = (C_n, R_\pi, C_f)$ ≡ Resulting $\pi$-model

1. $y_1^1 = C_n^l + C_f^l, y_1^r = C_n^r + C_f^r, y_2^l = -R_\pi^l (C_f^l)^2$,
   $y_2^r = -R_\pi^r (C_f^r)^2, y_3^l = (R_\pi^l)^2 (C_f^l)^3, y_3^r = (R_\pi^r)^2 (C_f^r)^3$
2. $y_1 = y_1^l + y_1^r, y_2 = y_2^l + y_2^r, y_3 = y_3^l + y_3^r$
3. Return $\pi = (y_1 - (y_2^2/y_3), -y_3^2/y_2^3, y_2^2/y_3)$
```

Figure 5 New_π-model procedure for Case 2.

5.1 Moment Computation

Since Van Ginneken’s algorithm is bottom-up, bottom-up moment computations are required. Figure 6(a) shows a wire e connected to a subtree rooted at B. Assume that the moments $m_{BC}^{(1)}, m_{BC}^{(2)}, \ldots, m_{BC}^{(k)}$ have already been computed for the path from B to C. We wish to compute the moments $m_{AC}^{(1)}, m_{AC}^{(2)}, \ldots, m_{AC}^{(k)}$ so that the A-C delay can be derived. This computation is done via moment multiplication in a manner similar to [11] and [5].

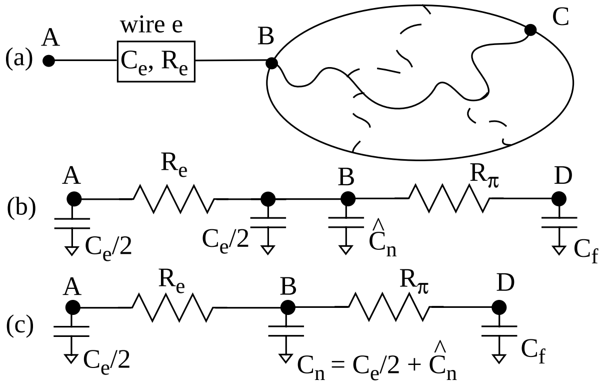

Figure 6 Illustration of the moments computation.

The techniques in Section 4 are used to reduce the subtree at B to a π-model $(\hat{C}_n, R_\pi, C_f)$ (Figure 6(b)). The wire e is also represented as a π-model, $(C_e/2, R_e, C_e/2)$. Node D just denotes the point on the far side of the resistor connected to B and not an actual physical location. The RC network can be further simplified to the network shown in Figure 6(c). The capacitances $\hat{C}_n$ and $C_e/2$ at B are merged to form a capacitor with value C_n. The moments from A to B can be recursively computed by the equation

$$m_{AB}^{(i)} = -R_e(m_{AB}^{(i-1)} C_n + m_{AD}^{(i-1)} C_f) \quad (3)$$

where the moments from A to D are given by

$$m_{AD}^{(i)} = m_{AB}^{(i)} - m_{AD}^{(i-1)} R_\pi C_f \quad (4)$$

and $m_{AB}^{(0)} = m_{AD}^{(0)} = 1$. Now the moments from A to C can be computed via moment multiplication as follows.

$$m_{AC}^{(i)} = \sum_{j=0}^{i} (m_{AB}^{(j)} \cdot m_{BC}^{(i-j)}) \qquad (5)$$

The first three moments can be used to predict delay in an RC interconnect tree with reasonable accuracy [8], so our implementation uses $k = 3$. We use the method of [17] to map the first three moments to a two-pole approximation. The path delay is then computed via a Newton-Raphson iteration with or without a saturated ramp transition time.

5.2 Slack Computation

One property of Elmore delay that makes it attractive for timing optimization is that the delays are additive, i.e., the Elmore delay along a path from A to C through B equals the delay from A to C plus the delay from B to C. This property does not hold for higher-order delay models.

For example, consider the two-sink RC network in Figure 7. The required arrival times at C and D are 500 and 740 ps, respectively, and their Elmore delays from B are 250 and 500 ps, respectively. Under Elmore delay, the slack at node B is $min(500 - 250, 740 - 500) = 240$, which makes D the more critical sink. Observe that the critical path can be deduced without knowing the topology upstream from B, i.e., D is the critical sink regardless of the resistance of R.

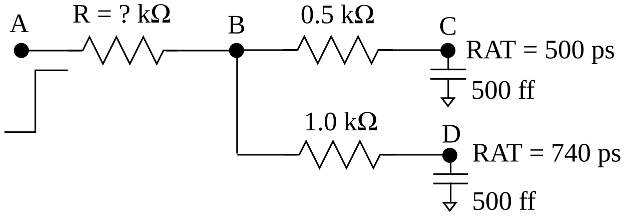

Figure 7 An RC network with source A and sinks C and D.

If moment matching is used to compute higher-order delays, then one cannot deduce whether B or D is the critical sink without first knowing R. Assuming a step response at A and R=0.25, then RICE reports an A-C delay of 317 ps and an A-D delay of 547 ps. The slack at node A is thus 183, and C is the more critical sink. However, if R=1, then the higher-order delays from A to C and A to D are 801 and 1090 ps, respectively. The slack at node A becomes -350, and D is now the more critical sink. Thus, different scenarios for the upstream resistance can lead to different critical sinks, a situation which does not occur for the Elmore model![1]

At a particular node B, one cannot only maintain the moments for the most critical path downstream from B since the most critical path is not known. Instead, one must store the moments for *all* the paths to sinks that are downstream from B. In a tree with n nodes and p sinks, the total number of moments that would have to be stored at internal nodes in the tree is $O(np)$. Our experiments verify that the extra runtime for the computation is not prohibitive.

6. The VGIG Algorithm

The VGIG (VG + accurate Interconnect + accurate Gate) algorithm is our proposed extension of VG. A *candidate* was previously defined as a 3-tuple $(C_{T(v)}, q, M)$. Now, a candidate is a 4-tuple $(\pi_{T(v)}, q, m, M)$. Here, $\pi_{T(v)}$ replaces $C_{T(v)}$ as the downstream capacitance model, and m stores the first three moments for each gate in $SI_{T(v)}$, where $SI_{T(v)}$ is the set of gates downstream from v.

```
Input:   $T = (\{so\} \cup SI \cup IN, E)$ ≡ Routing tree
         $B$ ≡ Buffer library
Output:  $\alpha$ ≡ Best candidate solution for source $so$

1. $S = $ Find_Cands$(so)$.
2. for each $\alpha = (\pi_{T(so)}, q, m, M) \in S$ do
       Compute $C_{eff}$ at $so$ with downstream $\pi_{T(so)}$
       Compute $Delay(so)$ using $C_{eff}$ capacitance
       Compute slack $q_w$ to each $w \in SI_{T(so)}$ using
           moments $m$ and $Delay(so)$
       Define $\tilde{q} = \min\{q_w \| w \in SI_{T(v)}\}$.
3. return $M$ such that $(C_{T(so)}, \tilde{q}, m, M) \in S$ has maximum $\tilde{q}$.
```

Figure 8 The VGIG (T, B) algorithm.

Figure 8 and Figure 9 show the description of the VGIG algorithm and its corresponding Find_Cands procedure. We now highlight the main differences between VGIG and VG.

- Effective capacitance, as opposed to a linear model, and moment matching, as opposed to Elmore delay, are used to compute the delays of the candidates for the source (Step 2 of Figure 8).
- Instead of using lumped capacitance, VGIG uses π-models. In Step 1 of Figure 9, the π-model is set to be a lumped capacitor. The New_π-model procedure for merging two π-models is invoked in Step 4, and the procedure for adding a uniform wire is used in Step 6. These steps replace simply adding lumped capacitances.
- To compute interconnect delays in VGIG, each candidate for a node v stores the first three moments to every gate downstream from v. Moments are initialized to zero when no interconnect is present (Steps 1 and 5 of Figure 9). In Step 6, Equations (3), (4), and (5) are used to compute the new set of moments $\tilde{m}$.
- In VG, slack was computed by summing delays of each individual piece of interconnect. With more accurate models, delay is not additive, so the slack to all sinks downstream from v are recomputed. An input slew of 400 ps is assumed for the buffer. The output slew of the buffer is used as the input slew for computing interconnect delay to the sinks. Finally, the minimum slack $\tilde{q}$ over all sinks downstream is computed.

The choice of a 400 ps input slew when computing the buffer delay is arbitrary. When inserting a buffer in a bottom-up algorithm, the topology of the tree upstream from the buffer is still unknown. Since one cannot know the

[1] Interestingly, when R = 0.25, the higher-order delay from A to B is 24 ps, which implies that the higher-order delay from B to C is 293 ps. However, the Elmore delay from B to C is 250 ps, which implies that the Elmore delay is *not* actually an upper bound for a given wire. Rather, it is an upper bound for an entire wire path.

value of the buffer’s input slew, we use a fixed value.

```
Input:    $v \equiv$ Current node to be processed
Output:   $S \equiv$ List of candidate solutions for node $v$
Globals:  $T = (\{so\} \cup SI \cup IN, E) \equiv$ Routing tree
          $B \equiv$ Buffer library

  $S = S_b = \varnothing$
1. if $v \in SI$ then
     $S = \{(\pi = (C_v, 0, 0), RAT(v), m = \{\{0, 0, 0\}\}, M)\}$
2. else if $v$ has only one child then
     for each $(\pi_{T(v)}, q, m, M) \in$ Find_Cands$(T.left(v))$ do
       $S = S \cup \{(\pi_{T(v)}, q, m, M)\}$
3. else if $v$ has two children
     $S_l =$ Find_Cands$(T.left(v))$.
     $S_r =$ Find_Cands$(T.right(v))$.
     Set $i = 1$ and $j = 1$.
4.   while $i \le |S_l|$ and $j \le |S_r|$ do
       Let $\alpha_l = (\pi_l, q_l, m_l, M_l)$ be the i$^{th}$ candidate in $S_l$.
       Let $\alpha_r = (\pi_r, q_r, m_r, M_r)$ be the j$^{th}$ candidate in $S_r$.
       $\pi =$ New_$\pi$-model$(\pi_l, \pi_r)$
       $S = S \cup \{(\pi, \min(q_l, q_r), m_l \cup m_r, M_l \cup M_r)\}$
       if $q_l \le q_r$ then $i = i + 1$.
       if $q_r \le q_l$ then $j = j + 1$.
5. if $v$ is a feasible buffer location then
     for each buffer $b \in B$ do
       for each $\alpha = (\pi_{T(v)}, q, m, M) \in S$
         Compute $C_{eff}$ from $b$ and $\pi_{T(v)}$.
         Use $C_{eff}$ and k-factor equations to compute $Delay(b)$.
         Compute slack $q_w$ to each $w \in SI_{T(v)}$
            using moments $m$ and $Delay(b)$
         Let $\tilde{q} = \min\{q_w \| w \in SI_{T(v)}\}$.
       Let $\alpha$ be such that $\tilde{q}$ is maximum.
       if such an $\alpha$ exists then
          Set $M(v) = b$,
          $S_b = S_b \cup \{((C_b, 0, 0), \tilde{q}, \{\{0, 0, 0\}\}, M)\}$
   $S = S \cup S_b$.
6. Let $e = (u, v)$ be the parent wire for $v$.
   for each $\alpha = (\pi_{T(v)}, q, m, M) \in S$ do
     Compute moments $m$ from $u$ to each $w \in SI_{T(u)}$
        using $m$, $e$, and Equations (3),(4), and (5).
     Compute slack $q_w$ to each $w \in SI_{T(u)}$ using moments $\tilde{m}$.
     $\pi =$ New_$\pi$-model$(e, \pi_{T(v)}))$
     $S = S \cup \{(\pi, \min\{q_w \| w \in SI_{T(u)}\}, \tilde{m}, M)\} - \alpha$
7. Prune $S$ of inferior solutions and return $S$.
```

Figure 9 Find_Cands(v) procedure.

One step that is unchanged in VGIG is Step 7, the candidate pruning scheme. Pruning is still based on total lumped capacitance and slack, which can cause non-inferior solutions to be pruned. Consider two candidates α_1 and α_2 with the same slack and π-models $\pi_1 = (100, 0, 0)$ and $\pi_2 = (0, 100, 101)$. Here, α_2 will get pruned in favor of α_1 since the lumped capacitance for π_1 (100) is less than π_2 (101). However, the effective capacitance of π_2 will likely be much less than π_1 because of resistive shielding. α_1 is the inferior solution, yet α_2 gets pruned.

VGIG could probably be improved by utilizing a more sophisticated pruning scheme. One might try to estimate effective capacitance instead of lumped capacitance. However, effective capacitance is typically computed in the presence of a driver, and the repeated expense of this computation may prove prohibitive. Another alternative might be to prune based on the three values in each π-model. This may lead to too few solutions being pruned, and it is not clear that an efficient pruning scheme can be found.

7. Experimental Results

For our experiments, we chose a subset of nets from an IBM ASIC part with over one million transistors. Nets with high total capacitance were selected since these generally require buffer insertion. Nets were divided into three groups.

- Small: 20 nets with between 2 and 6 sinks were randomly chosen from the set of high capacitance nets.
- Medium: 25 nets with between 7 and 15 sinks were randomly chosen from the set of high capacitance nets.
- Large: 10 nets with between 18 and 186 sinks were noted by designers as particularly troublesome for manual buffer insertion.

We ran four algorithms on each group with a buffer library consisting of 3 inverting and 13 non-inverting buffers.

- **VG** is Van Ginneken’s original algorithm,
- **VGI** is VG with accurate interconnect delays,
- **VGG** is VG with accurate gate delays, and
- **VGIG** is VG with both accurate gate and interconnect delays.

# Buffers	# Nets	Buffer Insertion Algorithm			
		VG	VGI	VGG	VGIG
1	20	569	617	548	612
2	20	709	734	783	794
3	20	663	716	797	815
4	19	653	678	831	855
5	12	726	748	1001	1036
CPU Time		223.4	245.7	235.2	244.3

Table 1: Average slack reduction (ps) for the 20 small nets.

Each algorithm can trade-off solution quality with the number of buffers. As additional buffers are inserted, the marginal improvement is reduced until the critical number of buffers is reached, at which point, adding additional buffers yields inferior solutions. For each net and for up to the critical number of buffers, the improvement in critical path delay versus the zero-buffer solution was recorded. The delay improvement was measured via our own analytic engine. We observe only small differences between our analytic engine and EinsTimer (the IBM static timing tool).

Tables 1, 2 and 3 present the average delay improvement as a function of the number of buffers inserted for each of the algorithms. Total runtimes are reported for an IBM RS6000/S595 with 1Gb of RAM in the last line of each table.

We make the following observations.

- Using moment matching (VGI) yields improvement over VG ranging form 22 to 53 ps for small nets, -2 to 44 ps for medium nets, and -18 to 43 ps for large nets. VGI generally has the most utility for 1 or 2 buffers, with performance declining as the number of buffers increases. That VGI sometimes obtains worse delays may be explained by our use of a fixed input slew when the driver is unknown. A better scheme for choosing input slew may be able to improve performance further.

# Buffers	# Nets	Buffer Insertion Algorithm			
		VG	VGI	VGG	VGIG
1	25	774	816	778	800
2	25	1034	1066	1068	1149
3	25	1075	1119	1212	1249
4	25	1116	1114	1231	1278
5	23	1116	1114	1328	1361
6	21	937	943	1249	1262
7	15	908	932	1364	1372
CPU Time		501.0	569.3	578.2	623.2

Table 2: Average slack reduction (ps) for the 25 medium nets.

# Buffers	# Nets	Buffer Insertion Algorithm			
		VG	VGI	VGG	VGIG
1	10	1556	1605	1493	1572
2	10	1938	1946	1949	1980
3	10	2089	2071	2131	2148
4	10	2062	2057	2175	2172
5	9	2190	2177	2382	2395
6	8	2368	2360	2621	2645
7	8	2927	2930	3437	3468
CPU Time		576.7	958.2	1058.6	1947.2

Table 3: Average slack reduction (ps) for the 10 large nets.

- More accurate gate delay modeling (VGG) yields improvements over VG of up to 275, 312, and 510 ps for small, medium and large nets respectively. VGG clearly performs better as the number of buffers increases; however, it sometimes performs worse than VG for one buffer. This may be due to the k-factor equations not accurately characterizing the buffer for very high loads.
- Using both accurate gate and interconnect delay models (VGIG) consistently outperforms VG. Improvements range from 43 to 310 ps for small nets, 26 to 464 ps for medium nets, and 16 to 541 ps for large nets. Further, VGIG yields the best result of all four algorithms in every case, except for the single buffer solutions.
- The runtime penalties for VGIG are not prohibitive. For the small and medium nets, VGIG uses 9 and 24 percent more CPU time than VG, respectively. For large nets, VGIG takes 3.4 times longer than VG. The increase is fairly evenly distributed between the interconnect and gate delay computations.

We have presented two techniques to improve the accuracy of delay computations within Van Ginneken's algorithm. Experiments show that our modeling leads to reductions in critical path delay without a prohibitive runtime increase. In future work, we plan to integrate simultaneous wire sizing. The use of accurate interconnect delay models may prove to yield significantly higher dividends in this regime.

Acknowledgments

The authors sincerely thank Gopal Gandham and Jose Neves for their useful discussions and continued support of this project.

References

[1] C. J. Alpert and A. Devgan, "Wire Segmenting For Improved Buffer Insertion", *IEEE/ACM DAC,*1997, pp. 588-593.

[2] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer Insertion for Noise and Delay Optimization", *DAC,* 1998, pp. 362-367.

[3] C. C. N. Chu and D. F. Wong, "Closed Form Solution to Simultaneous Buffer Insertion/Sizing and Wire Sizing", *International Symposium on Physical Design*, 1997, pp. 192-197.

[4] C. C. N. Chu and D. F. Wong, "A New Approach to Simultaneous Buffer Insertion and Wire Sizing", *IEEE/ACM Intl. Conference on Computer-Aided Design*, 1997, pp. 614-621.

[5] J. Cong and C.-K. Koh, "Interconnect Layout Optimization Under Higher-Order RLC Model", *ICCAD*, 1997, 713-720.

[6] 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.

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

[8] R. Gupta, B. Krauter, B. Tutuianu, J. Willis and L. T. Pileggi, "The Elmore Delay as a Bound for RC Trees with Generalized Input Signals", *DAC*, 1995, pp. 364-369.

[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 J. Solid-State Circuits*, 31(3), 1996, 437-447.

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

[11] F.-J. Liu, J. Lillis and C.-K. Cheng, "Design and Implementation of a Global Router Based on a New Layout-Driven Timing Model with Three Poles", *ISCAS*, 1997, pp. 1548-1551.

[12] P. R. O'Brien and T. L. Savarino, "Modeling the Driving-Point Characteristic of Resistive Interconnect for Accurate Delay Estimation", *IEEE/ACM ICCAD*, 1989, pp. 512-515.

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

[14] L. T. Pillage and R. A. Rohrer. Asymptotic Waveform Evaluation for Timing Analysis. *IEEE TCAD,* 9(4), 1990, 352-366.

[15] J. Qian, S. Pulllela, and L. Pillage, "Modeling the "Effective Capacitance" for the RC Interconnect of CMOS Gates", *IEEE Trans. CAD*,. 13(12), 1994, pp. 1526-1535.

[16] C. Ratzlaff and L. T. Pillage, "RICE: Rapid Interconnect circuit Circuit Evaluator using Asymptotic Waveform Evaluation", *IEEE Trans. on CAD*, pp. 763-776, June 1994.

[17] B. Tutuianu, F. Dartu, and L. Pileggi, "Explicit RC-Circuit Delay Approximation Based on the First Three Moments of the Impulse Response", *DAC*, 1996, pp. 611-616.

[18] L. P. P. P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay", *Intl. Symp.Circuits and Systems*, 1990, pp. 865-868.