Buffer Insertion for Noise and Delay Optimization

Charles J. Alpert
IBM Austin Research Laboratory
Austin, TX 78660
alpert@austin.ibm.com

Anirudh Devgan
IBM Austin Research Laboratory
Austin, TX 78660
devgan@austin.ibm.com

Stephen T. Quay
IBM Microelectronics Division
Austin, TX 78660
quayst@austin.ibm.com

Abstract

Buffer insertion has successfully been applied to reduce delay in global interconnect paths; however, existing techniques only optimize delay and timing slack. With the increasing ratio of coupling to total capacitance and the use of aggressive dynamic logic circuit families, noise is becoming a major design bottleneck. We present comprehensive buffer insertion techniques for noise and delay optimization. Our experiments on a microprocessor design show that our approach fixes all noise violations that were identified by a detailed, simulation-based noise analysis tool. Further, we show that the performance penalty induced by optimizing both delay and noise as opposed to only delay is 2%.

1. Introduction

Performance optimization has always been a critical step in the design of integrated circuits. Process technology scaling has made interconnect performance more dominant than transistor and logic performance. With the continued scaling of process technology, the interconnect resistance per unit length continues to increase, the capacitance per unit length remains roughly constant and logic delay continues to decrease. These trends have caused interconnect delay to become more dominant than logic delay. Process technology options, such as copper wires, can only provide temporary relief. The trend of increasing interconnect dominance is expected to continue.

Interconnect-driven timing optimization techniques, such as wire sizing, buffer insertion and gate sizing have gained widespread acceptance in deep submicron design [7]. In particular, buffer insertion techniques have been successful in reducing interconnect delay. To the first order, interconnect delay is proportional to the square of the length of the wire. Inserting buffers effectively divides the wire into smaller segments, which makes the interconnect delay almost linear in terms length (plus the buffer delays). Additional advantages of buffer insertion will make this optimization even more pervasive as the ratio of device to interconnect delay continues to decrease.

Several works study delay-driven buffer insertion. Closed formed solutions for 2-pin nets are proposed in [1] [4] [5] and [9]. In [16], Van Ginneken develops a dynamic programming algorithm which finds the *optimal* buffer placement under the Elmore delay model [10]. In [12], Lillis et al. extends this algorithm to simultaneously perform wire sizing, while also minimizing the total number of buffers. Finally, Alpert and Devgan [1] propose a wire segmenting pre-processing algorithm to handle the one buffer per wire limitation of Van Ginneken's algorithm, which results in a smooth trade-off between solution quality and run time.

Although timing optimization has always been critical in the design process, present day design techniques and process technologies are making noise analysis and avoidance as important. The shrinking of the minimum distance between adjacent wires has caused an increase in their coupled capacitance. Furthermore, as the ratio of wire thickness to width continues to increase, so will the ratio of coupling to total capacitance. Coupling capacitance can cause a switching net to induce noise onto a neighboring net, resulting in an incorrect functional response. Further, the widespread use of dynamic logic circuits has made noise avoidance even more critical since these logic families are more susceptible to noise failure. It is no longer sufficient or even acceptable to optimize only for delay. Noise avoidance techniques must become an integral part of the performance optimization environment. Buffer insertion provides a suitable platform for optimizing both timing and noise.

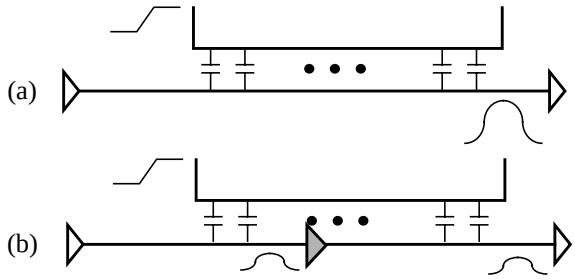

Figure 1 Noise on a victim net (a) without and (b) with a buffer.

Figure 1(a) shows the noise effect that an aggressor net (top) can have on a victim net (bottom). The coupling capacitance may cause an input signal on the aggressor net to induce a noise pulse on the victim net. If the resulting noise is greater than the tolerable noise margin of the sink, then an electrical fault results. Figure 1(b) shows how inserting a buffer can distribute the capacitive coupling between the two newly created wires, resulting in smaller noise pulses on the input of the inserted buffer and on the sink. If the amplitude of these noise pulses are less than he noise margins for the sink and the buffer, then the circuit will function correctly.

Noise analysis is typically performed through detailed circuit simulation or through reduced order interconnect analysis (e.g., AWE[13] and RICE[15]). Although the latter is more efficient, it is still too slow to be used within an optimization tool. Instead, we adopt the noise metric of [8].

The rest of the paper is as follows. Section 2 presents notation and definitions. In Section 3, we derive a formula for the maximum wire length such that no noise violation is induced and also present two optimal algorithms for noise

1-58113-049-x-98/0006/$3.50

avoidance. Section 4 presents a third algorithm for minimizing delay such that all noise constraints are satisfied. Finally, Section 5 presents experimental results.

2. Preliminaries

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 $(e = (u, v)) \in E$ with length l_e is an ordered pair of nodes in which the signal propagates from u to v. Each node $v \in SI \cup IN$ 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.[1] Let the left and right children of v be denoted by $T.left(v)$ and $T.right(v)$ respectively. Assume that if v has only one child, then it is $T.left(v)$. The path from node u to v, denoted by $path(u, v)$, is the set of wires that connect u to v. We are also given a buffer library $B = \{b_1, b_2, \ldots, b_m\}$.

A buffer insertion solution is a mapping $M: IN \rightarrow B \cup \{\bar{b}\}$ which either assigns a buffer or no buffer, denoted by $\bar{b}$, to each internal node of T.[2] Let $|M| = |\{v \in IN: M(v) \in B\}|$ denote the number of internal nodes with inserted buffers. Wires are segmented as in [1] to create as many internal nodes as necessary to form a reasonable set of buffer locations. Assigning k buffers to T induces $k+1$ nets, and hence $k+1$ subtrees, each with no internally placed buffers. For each $v \in V$, 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 $v \in SI$, $T(v)$ contains only one node.

2.1 Delay Optimization

As in [1][12][16], we adopt the Elmore delay model [10] for interconnect delays and a linear model for gate delays. For each gate v, let C_v denote the input capacitance, R_v the resistance and K_v the intrinsic delay of v. Let C_e and R_e respectively denote the lumped capacitance and resistance for each wire $e \in E$. The capacitive *load* seen at node v is the total lumped capacitance $C_{T(v)}$ of $T(v)$, i.e.,

$$C_{T(v)} = \sum_{w \in SI_{T(v)}} C_w + \sum_{e \in E_{T(v)}} C_e . \quad (1)$$

The Elmore delay for a wire $e = (u, v)$ is given by $Delay(e) = R_e(C_e/2 + C_{T(v)})$. The delay through a gate $v \in \{so\} \cup B$ is given by $Delay(v) = K_v + R_v C_{T(v)}$. If $v \in IN, M(v) = \bar{b}$, then $Delay(v) = 0$. The total delay from $v \in V$ to $si \in SI$ is given by

$$Delay(v - si) = \sum_{e = (u, w) \in path(v, si)} Delay(e) + Delay(u) . \quad (2)$$

Each sink si has a given required arrival time $RAT(si)$, and assume that the input signal arrives at the source node at time zero. The condition $\forall si \in SI, Delay(so - si) \leq RAT(si)$ must hold for the circuit to meet timing requirements. For every $v \in V$, let $q(v) = min_{si \in ds(v)} RAT(si) - Delay(v - si)$ be the *slack* at v where $ds(v)$ is the set of all sinks that are downstream from v. Observe that the circuit meets its timing if and only if $q(so) \geq 0$.

[1] A non-binary tree can be converted into a binary tree by inserting wires with zero resistance and capacitance where appropriate.

[2] A buffer placed on an internal node with degree d is interpreted as having one input, one output, and $d-1$ fanouts.

2.2 Noise Avoidance

In [8], Devgan proposes a coupled noise estimation metric which is an upper bound for RC and overdamped RLC circuits. The metric depends on the resistance of the victim net, the resistance of the gate driving the victim net, coupling capacitances to the aggressor nets, and the rise times and the slopes of the signals on the aggressor nets. For example, consider the three aggressor nets and the single 2-pin victim net in Figure 2. The wire in the victim net is segmented into seven new wires such that each new wire is completely coupled to either 0, 1 or 2 of the aggressor nets.

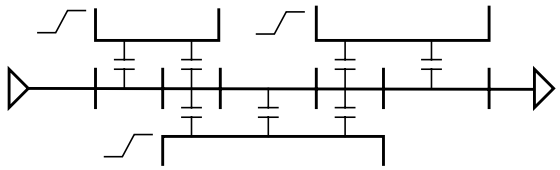

Figure 2 Wire segmenting scheme for multiple aggressor nets.

The coupling capacitance from an aggressor net can be modeled as some fraction of the wire capacitance of the victim net. Given t simultaneously switching aggressor nets $1, \ldots, t$ near wire e, let $\lambda_1, \ldots, \lambda_t$ be the ratios of coupling to wire capacitance from the aggressor nets to e, and let $\dot{\mu}_1, \ldots, \dot{\mu}_t$ be the *slopes* (i.e., power supply voltage over input rise time) of the aggressor net signals. The total *current* I_e induced by the aggressor nets on e is

$$I_e = C_e \sum_{j=1}^{t} (\lambda_j \cdot \dot{\mu}_j) \quad (3)$$

Often, information about neighboring aggressor nets is incomplete, especially when buffer insertion is performed before routing. When performing buffer insertion in *estimation mode*, one might assume that (i) each wire is coupled to exactly one aggressor net, (ii) the slope of all aggressor nets is $\dot{\mu}$, and (iii) some fixed ratio λ of the total capacitance of each wire is due to coupling capacitance. Under these assumptions $I_e = \lambda \dot{\mu} C_e$ for each wire e.

Let $I_{T(v)}$ be the total downstream current seen at v, i.e.,

$$I_{T(v)} = \sum_{e \in E_{T(v)}} I_e . \quad (4)$$

Each wire adds to the noise induced on the victim net. The amount of additional *noise* induced from a wire $e = (u, v)$ is given by $Noise(e) = R_e(I_e/2 + I_{T(v)})$. The total additional noise seen at a sink si starting at some upstream node v is given by

$$Noise(v - si) = I_{T(v)} R_v + \sum_{e \in path(v - si)} Noise(e) \quad (5)$$

where $R_v = 0$ if there is no gate at v. The path from v to si has no intermediate buffers. If an intermediate buffer is present, the noise computation begins from the output of the buffer, since the buffer is a restoring stage. Each node $v \in SI \cup B$ has a predetermined *noise margin* $NM(v)$. The condition $\forall v \in \{so\} \cup \{IN | M(v) \in B\}$, $\forall si \in SI_{T(v)}$, $Noise(v - si) \leq NM(si)$ must hold for there to be no electrical faults. We define the noise slack for every $v \in V$ as $NS(v) = min_{si \in SI_{T(v)}} NM(si) - Noise(v - si)$. Noise slack serves equivalently as a noise margin for internal nodes. Noise constraints for downstream sinks in $T(v)$ are satisfied

if and only if $NS(v)$ is greater than the noise seen at v. Observe that $NS(v) \geq 0$ for each $v \in \{so\} \cup \{IN | M(v) \in B\}$ if there are no noise violations.

2.3 Problem Formulations

We study two different buffer insertion problems. The first problem seeks to fix all noise violations with the fewest possible buffers. Delay is not considered

Problem 1: Given a tree $T = (\{so\} \cup SI \cup IN, E)$, a buffer library B, and noise margins $NM(v)$ for each $v \in SI \cup B$, find a solution $M: IN \rightarrow (B \cup \{\bar{b}\})$ which minimizes $|M|$, such that $NS(v) \geq 0$ for each $v \in \{so\} \cup \{IN | M(v) \in B\}$.

This problem may be useful for non-critical nets, for which delay optimization is unnecessary. For timing-critical nets, we must consider both noise and delay at the same time.

Problem 2: Given a tree $T = (\{so\} \cup SI \cup IN, E)$, a buffer library B, and noise margins $NM(v)$ for each $v \in SI \cup B$, find a solution $M: IN \rightarrow (B \cup \{\bar{b}\})$ which minimizes $q(so)$ such that $NS(v) \geq 0$ for each $v \in \{so\} \cup \{IN | M(v) \in B\}$.

A third formulation can seek to minimize the total number of buffers inserted by while satisfying both noise and timing constraints. Algorithm 3, which is used to solve Problem 2, can also be applied to address this third formulation using an extension of Lillis et al, [12] to Van Ginneken [16].

3. Noise Constrained Buffer Insertion

We begin with the simplest case of a wire with uniform width and neighboring coupling capacitance, as shown in Figure 3. For each wire e, let $R = R_e/l_e$ and $I = I_e/l_e$ respectively be the wire resistance per unit length and the current per unit length. Since current is a constant times wire capacitance, we can use a π-model to represent its distribution.

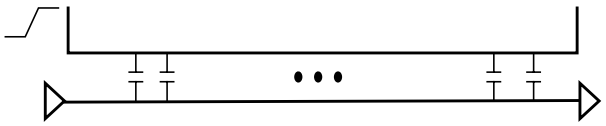

Figure 3 Uniform coupling capacitance for a single wire.

Theorem 1 For a given wire $e = (u, v)$ in a routing tree T, a buffer b needs to be inserted on e to satisfy noise constraints if and only if

$$l_e \leq -\frac{R_b}{R} - \frac{I_{T(v)}}{I} + \sqrt{\left(\frac{R_b}{R}\right)^2 + \left(\frac{I_{T(v)}}{I}\right)^2 + \frac{2NS(v)}{IR}}. \qquad (6)$$

Proof: For noise constraints to be satisfied, we must have $R_b(Il_e + I_{T(v)}) + Rl_e((Il_e/2) + I_{T(v)}) \leq NS(v)$ which is a quadratic in l_e. Solving for l_e yields the theorem. □

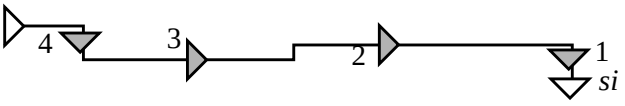

Figure 4 Iterative application of Theorem 1.

Theorem 2 A net that has been optimally buffered to minimize *delay alone* may be susceptible to noise violations.

Proof: Consider a wire $e = (u, v)$ in which u and v are gates in the buffered net (a similar analysis holds for a path from u to v). Let $\dot{\mu}$ be the slope of an aggressor net, and let λ be the ratio of coupling to total capacitance for e. If there is a noise violation at v, then the following must hold.

$$l_e > -\frac{R_b}{R} + \sqrt{\left(\frac{R_b}{R}\right)^2 + \frac{2NM(v)}{RC\lambda\dot{\mu}}} \qquad (7)$$

Solving for $NM(v)$ yields

$$NM(v) < \frac{RC\lambda\dot{\mu}}{2}\left(l_e^2 + \frac{2l_eR_b}{R}\right). \qquad (8)$$

For any fixed values for the parameters on the right side of the inequality, a noise violation will occur if the noise margin is small enough. Even if $NM(v)$ is reasonably large, an aggressor net can have a very large slope $\dot{\mu}$ and a high coupling ratio λ which would also cause a violation. □

3.1 Noise Avoidance for Single-Sink Trees

Theorem 1 suggests how to insert buffers for single-sink trees. Begin at the sink and work up the tree, updating the total downstream current and noise slacks of visited internal nodes. At each internal node, use Theorem 1 to decide if a buffer should be inserted. The algorithm terminates when the source node is reached.

Input: $T = (\{so\} \cup \{si\} \cup IN, E)$ ≡Routing tree
b ≡ buffer type
Output: $M: IN \rightarrow B$ ≡ Buffer Insertion Solution

1. Set $I_{T(si)} = 0, NS(si) = NM(si)$, $v = si$.
2. while $(v \neq so)$ do
   Let $e = (u, v)$ be the parent wire for v.
3. if $R_b(I_{T(v)} + I_e) + R_e(I_{T(v)} + I_e/2) < NS(v)$
   Set $I_{T(u)} = I_{T(v)} + I_e$,
   Set $NS(u) = NS(v) - R_e(I_{T(v)} + I_e/2)$. Set $v = u$.
4. else
   Let $I = I_e/l_e$, $R = R_e/l_e$.
   Set $l = -\frac{R_b}{R} - \frac{I_{T(v)}}{I} + \sqrt{\left(\frac{R_b}{R}\right)^2 + \left(\frac{I_{T(v)}}{I}\right)^2 + \frac{2NS(v)}{IR}}$.
   Create internal node w (with parent u and child v) on e at distance l from v. Set $M(w) = b$.
   Set $I_{T(w)} = 0$, $NS(w) = NM(b)$, and $v = w$.
5. $NS(so) = NS(so) - R_{so}I_{T(so)}$
   if $NS(so) < 0$, create internal node w at so. Set $M(w) = b$.

Figure 5 Algorithm 1, Noise Avoidance for Single-Sink Trees.

Algorithm 1, Noise Avoidance for Single-Sink Trees, is presented in Figure 5. The algorithm accepts a routing tree, and a single buffer type b. Step 1 initializes the current and noise slack of the sink node, then Steps 2-4 climb up the tree visiting each node in turn. Step 3 examines whether or not a buffer needs to be inserted on the current wire $e = (u, v)$, by computing the noise from placing a buffer at u. If this noise is less than the noise slack, no buffer needs to be inserted, so the algorithm computes the downstream current and noise slack for node u, then moves to the next wire. But, if the noise is larger then the noise slack, then a buffer must be inserted. Step 4 computes the maximum length that this buffer may be inserted from v, and inserts it there at a new internal node w. Finally, Step 5 computes the noise slack at the driver and inserts a buffer right after the driver if there is a noise violation (which can only occur if $R_{so} > R_b$). Optimality follows from the fact that buffers are always inserted their maximal distance up the tree, according to Theorem 1.

3.2 Noise Avoidance for Multi-Sink Trees

Some difficulty arises in extending Algorithm 1 to multiple sinks. Let $e_l(e_r)$, $I_l(I_r)$, and $NS_l(NS_r)$ respectively be the wire, current and noise slack for the left (right) branch of an internal node v with two children. It is possible that $R_b I_l \le NS_l$, $R_b I_r \le NS_r$ and $R_b(I_l + I_r) > min(NS_l, NS_r)$, i.e., the noise constraints for the left and right branches are satisfied but merging the left and right branches will induce a noise violation. Thus, a buffer must be placed on either the left or right branch immediately after v. One cannot immediately deduce which branch to choose since one needs to know the characteristics and location of the gate driving v. Since the algorithm is bottom-up, the location of this gate has not yet been determined.

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

1. if $v \in SI$, then set $S = \{(0, NM(v), \bar{b})\}$
2. else if v has only one child then
   $S = Algorithm\ 2(T.left(v))$
3. else if v has two children
   $S_l = Algorithm\ 2(T.left(v))$
   $S_r = Algorithm\ 2(T.right(v))$
   Set $i = 1$ and $j = 1$.
4. while $j \le |S_l|$ and $k \le |S_r|$ do
   Let $\alpha_l = (I_l, NS_l, M_l)$ be the i[th] candidate in list S_l.
   Let $\alpha_r = (I_r, NS_r, M_r)$ be the j[th] candidate in list S_r.
5. if $R_b(I_l + I_r) > min(NS_l, NS_r)$ then
   Insert nodes w_l, w_r just after v on the left and right branches.
   Set $M_l^b = M_l$, $M_r^b = M_r$.
   Assign $M_l^b(w_l) = b$, $M_r^b(w_r) = b$
6. $S = S \cup \{(I_r, min(NM(b), NS_r), M_l^b \cup M_r)\}$
   $S = S \cup \{(I_l, min(NM_l, NS(b)), M_l \cup M_r^b)\}$
   else
7. $S = S \cup \{(I_l + I_r, min(NS_l, NS_r), M_l \cup M_r)\}$
8. if v is the source then
   for each $\alpha = (I_{T(v)}, NS(v), M) \in S$ do
   $NS(so) = NS(so) - R_{so} I_{T(so)}$
   if $NS(so) < 0$ create internal node w at so.
   Set $M(w) = b$.
   return S
9. Let $e = (u, v)$ be the parent wire of v
   for each $\alpha = (I_{T(v)}, NS(v), M) \in S$ do
   if $R_b(I_{T(v)} + I_e) + R_e(I_{T(v)} + I_e/2) < NS(v)$ then
   Set $\alpha = (I_{T(v)} + I_e, NS(v) - R_e(I_{T(v)} - I_e/2), M)$
   else
10. Let $I = I_e/l_e$, $R = R_e/l_e$.
   Set $l = -\frac{R_b}{R} - \frac{I_{T(v)}}{I} + \sqrt{\left(\frac{R_b}{R}\right)^2 + \left(\frac{I_{T(v)}}{I}\right)^2 + \frac{2NS(v)}{IR}}$
   Create internal node w (with parent u and child v) on wire e at distance l from v.
   $M(w) = b$, $\alpha = (I(l_e - l), NM(b) - IR_e(l_e - l), M)$
11. Prune S of inferior solutions and return S

Figure 6 Algorithm 2, Noise Avoidance for Multi-Sink Trees.

We propose to generate a set of *candidate* solutions for each node and propagate these candidate solutions up the tree, in the same spirit as in Van Ginneken's algorithm [16]. A *candidate* α is a 3-tuple $(I_{T(v)}, NS_{T(v)}, M)$ where $I_{T(v)}$ is the downstream current seen at v, $NS_{T(v)}$ is the noise slack for v, and M is the current solution for the subtree $T(v)$. Let $M = M_l \cup M_r$ denote the resulting solution from merging the left solution M_l and the right solution M_r for an internal node v with two children. For each w in $T(v)$, we assign $M(w) = b$ if either $M_l(w) = b$ or $M_r(w) = b$ and $M(w) = \bar{b}$ otherwise. Whenever a node with two children is encountered and a buffer needs to be inserted, both a left and right candidate is generated. The candidates are sorted in non-decreasing order by downstream current so that inferior solutions can be pruned [16]. Given two candidates $\alpha_1 = (I_1, NS_1, M_1)$ and $\alpha_2 = (I_2, NS_2, M_2)$, α_1 is *inferior* to α_2 if and only if $I_1 > I_2$ and $NS_1 \le NS_2$.

Algorithm 2 is similar to Algorithm 1, except for Steps 4-7 which handle nodes with two children. Step 4 iterates through each candidate for the left branch and each candidate in the right branch using the Van Ginneken's linear merging technique. Step 5 tests whether merging the two candidates results in a noise violation, and if not, Step 7 merges the two sets of candidates without inserting a buffer. If there is a violation, then two new solutions, one with a buffer on the left branch and one a new buffer on the right branch, are generated and inserted into the current list of candidates. When the algorithm terminates, the solution(s) in S with the fewest number of buffers is chosen.

The algorithm returns an optimal solution to Problem 1 in time quadratic in $|V|$. As for Algorithm 1, one can obtain an optimal solution for a buffer library with multiple buffer types by selecting the buffer type with least resistance.

4. Optimizing Noise and Delay

To address Problem 2, we modify the approach of Van Ginneken [16] to include noise avoidance. Whenever Van Ginneken's algorithm considers inserting a buffer, we check the noise constraints; if they have been violated, the buffer is not inserted. Hence, our algorithm generates fewer solutions than Van Ginneken's algorithm since it *prunes* solutions which have noise violations.

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

1. $T = Segment\_Wires(T, B)$ [1].
2. $S = Find\_Cands(so)$.
3. for each $\alpha = (C_{T(so)}, q(so), \boldsymbol{I_{T(so)}}, \boldsymbol{NS(v)}, M) \in S$ do
   Set $q(v) = q(v) - K_{so} - R_{so} C_{T(v)}$
   Set $NS(so) = NS(so) - I_{T(so)} R_{so}$.
4. return $(C_{T(so)}, q(so), I_{T(so)}, NS(so), M) \in S$
   with maximum $q(so)$ **such that $NS(so) \ge 0$**

Figure 7 Algorithm 3: Optimizing Noise and Delay.

As before, a list of candidates is computed for each node, except that a *candidate* $\alpha = (C_{T(v)}, q(v), I_{T(v)}, NS(v), M)$ is now a 5-tuple where $C_{T(v)}$ is the load seen at v, $q(v)$ is the slack at v, $I_{T(v)}$ is the downstream current seen at v, $NS(v)$ is the noise slack at v, and M is the current solution. Figure 7 illustrates Algorithm 3, which is the same as Van Ginneken's[3] except for the modifications in boldface. Step 1 of Figure 7 segments the wires to generate sufficient possible buffer locations. Step 2 calls Find_Cands which

[3] The algorithm can be modified to handle inverting buffers [12].

returns a list of candidate solutions. Step 3 adds the driver delay and computes the noise slack, then the candidate with the best timing slack, such that noise constraints are satisfied, is returned in Step 4.

The Find_Cands procedure shown in Figure 8. It takes the node v as input, recursively computes the lists of possible candidates for all the nodes in $T(v)$, and then returns the candidates for v. Find_Cands consists of four main parts:

- Steps 1-4 constructs candidates for the children of v and merges them to form S, the set of candidates for v.
- Step 5 inserts considers each buffer type in the library and adds the buffer which yields the largest slack *such that noise constraints are satisfied*. A buffer will not be inserted if there is a noise violation. This step is the fundamental difference between Algorithm 3 and Van Ginneken's algorithm [16].
- Step 6 computes the new load, slack, current and noise slack for each candidate induced by the parent wire of v.
- Finally, Step 7 prunes inferior candidates from S, using the pruning schemes of [12][16].

Modifications for noise avoidance do not increase the $O(|V|^2|B|^2)$ time complexity of Van Ginneken's algorithm since the modifications are mostly constant time operations for bookkeeping and pruning.

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

$S = S_b = \emptyset$
1. if $v \in SI$ then $S = \{(C_v, RAT(v), \mathbf{0}, \boldsymbol{NM(v)}, M)\}$
2. else if v has only one child then
  for each $\alpha \in Find\_Cands(T.left(v)))$ do
    $S = S \cup \{\alpha\}$
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 $\alpha_l = (C_l, q_l, \boldsymbol{I_l}, \boldsymbol{NS_l}, M_l)$ be j[th] candidate in S_l.
    Let $\alpha_r = (C_r, q_r, \boldsymbol{I_r}, \boldsymbol{NS_r}, M_r)$ be k[th] candidate in S_r.
    Let $q = min(q_l, q_r)$, $\boldsymbol{NS} = \boldsymbol{min(NS_l, NS_r)}$
    $S = S \cup \{(C_l + C_r, q, \boldsymbol{I_l + I_r}, \boldsymbol{NS}, M_l \cup M_r)\}$
    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
    Find $(C_{T(v)}, q(v), \boldsymbol{I_{T(v)}}, \boldsymbol{NS(v)}, M) \in S$ that maximizes $q(v) - K_b - R_b C_b$ **such that** $\boldsymbol{NS(v) - I_{T(v)} R_b \ge 0}$
    if such a candidate exists then
      Set $M(v) = b$ and
      $S_b = S_b \cup (C_b, q(v) - K_b - R_b C_b, \mathbf{0}, \boldsymbol{NM(b)}, M)$
  $S = S \cup S_b$
6. Let $e = (u, v)$ be the parent wire for v.
  for each $\alpha = (C_{T(v)}, q(v), \boldsymbol{I_{T(v)}}, \boldsymbol{NS(v)}, M) \in S$ do
    Set $q = q(v) - R_e(C_e/2 + C_{T(v)})$
    Set $\boldsymbol{NS} = \boldsymbol{NS(v) - R_e(I_e/2 + I_{T(v)})}$
    $S = S \cup \{(C_{T(v)} + C_e, q, \boldsymbol{I_{T(v)} + I_e}, \boldsymbol{NS}, M)\} - \alpha$
7. Prune S of inferior solutions and return S.

Figure 8 Find_Cands Procedure(v).

Theorem 3 If $|B| = \{b\}$ and $NM(b) \ge NM(si), \forall si \in SI$, then Algorithm 3 returns an optimal solution to Problem 2.

Proof: Refer to [2] for a proof. With multiple buffers in the buffer library, the optimality of Algorithm 3 is no longer guaranteed. However, in practice, Algorithm 3 generates solutions that are very close to optima; our experimental results in the next section strongly support this claim.

5. Experimental Results

The algorithm we use in our experiments is an extended version of Algorithm 3, called BuffOpt, which can trade-off between delay reduction, noise avoidance, and the total number of buffers [12]. We refer to the algorithm which optimizes only delay [1][12][16] as DOpt. DOpt is the same as Algorithm 3 but without the boldface text. For our experiments, we selected a set of 500 critical nets from a modern Power PC microprocessor design. Table 1 shows the distribution of the sizes of these nets. To verify BuffOpt, we also ran a detailed, simulation-based noise analysis tool, called 3dnoise [14]. 3dnoise was run both before and after BuffOpt and DOpt.To perform noise analysis, 3dnoise uses accurate moment-matching based techniques that are similar to RICE [15].

#Sinks	1	2	3	4	5-7	8-10	11-20	21-40	41-57
#Nets	165	33	33	38	55	69	63	39	12

Table 1: Sink distribution of the 500 test nets.

We ran BuffOpt, DOpt and 3dnoise all in estimation mode (see Subsection 2.2) assuming a 0.7 coupling to total capacitance ratio from a single aggressor net with rise time 0.25 nanoseconds and a power supply voltage of 1.8V. The tolerable noise margin for every gate was set to 0.8V. The buffer library contained 5 inverting and 6 non-inverting buffers of varying power levels. Our experiments show that

- BuffOpt eliminated all noise problems in the design,
- DOpt could not fix all noise problems, and
- the average delay penalty from using BuffOpt instead of DOpt was less than 2%.

5.1 BuffOpt Successfully Avoids Noise

We ran BuffOpt on the 500 nets, and observed that BuffOpt identified 423 noise violations and successfully inserted buffers to fix all of them. To verify BuffOpt's identification of noise violations, we ran 3dnoise on the nets before running BuffOpt. The accurate analysis of 3dnoise identified 386 nets with noise violations, all of which were also identified by BuffOpt. BuffOpt identified 423 - 386 = 37 more nets with violations, which shows that the noise metric [8] is slightly conservative. We also ran 3dnoise on the nets after running BuffOpt, and 3dnoise identified no noise violations. This data is summarized in Table 2.

	# Nets	# Optimized	# Noise Violations
Before BuffOpt	500	0	386
After BuffOpt	500	423	0

Table 2: Number of noise violations reported by 3dnoise before and after running BuffOpt.

5.2 Optimizing Delay Alone is Insufficient

We now compare BuffOpt to DOpt (optimal delay-driven buffer insertion) in terms of noise avoidance. Since BuffOpt never inserted more than four buffers on any net, we ran DOpt four times in which no solution was allowed to have more than k buffers where k ranged from 1 to 4. We denote one such run of DOpt by DOpt(k). Table 3 compares DOpt(k) with BuffOpt. TBI stands for "total buffers inserted" (i.e., $\sum d$ times the number of nets with d buffers)[4], and #NVs stands for "number of noise violations".

Method	# Nets with d Buffers					TBI	# NVs	Total CPU
	d=0	d=1	d=2	d=3	d=4			
None	500					0	386	N/A
DOpt(1)	7	493				493	150	216
DOpt(2)	5	19	476			971	37	613
DOpt(3)	5	16	77	402		1376	21	936
DOpt(4)	4	16	29	35	416	1843	13	1135
BuffOpt	77	161	232	28	2	717	0	980

Table 3: BuffOpt versus DOpt(k) for noise.

Observe that running DOpt(4) causes the addition of 1126 more buffers than BuffOpt, yet there are still 13 noise violations. Reducing the maximum number of buffers that can be inserted by DOpt only increases the number of noise violations and still causes more buffers to be inserted for $k \geq 2$. Thus, as shown by Theorem 2, noise avoidance *must* be integrated into the algorithm to guarantee no noise violations. Finally, observe that BuffOpt uses less CPU than DOpt(k) for $k \geq 3$. This occurs because BuffOpt prunes candidates with noise violation, which gives BuffOpt fewer total candidates to analyze.

5.3 The Delay Penalty is Small

# Buffers	0	1	2	3	4	Avg DR
# Times	77	161	232	28	2	423
BuffOpt	0	124.8	336.0	1043.4	45.5	301.1
DOpt	0	135.8	338.2	1053.1	50.3	307.1
Delay Penalty	0	8.77	0.64	0.93	10.50	1.99

Table 4: Average delay reduction from buffer insertion.

Finally, we compare DOpt to BuffOpt in terms of total delay. We first ran BuffOpt on the 500 nets and recorded the buffers inserted for each net. We then ran DOpt for the same number of buffers as BuffOpt inserted in order to make an apples to apples comparison. We computed the reduction in total delay for each net and averaged the results by the number of buffers inserted. The cumulative results are presented in Table 4. For example, there were 232 nets for which two buffers were inserted, and on average BuffOpt reduced delay by 336.0 picoseconds while DOpt reduced delay by 338.2 picoseconds. The overall delay penalty is $100(338.2 - 336.0)/338.2 = 0.64$ percent. The average delay reduction over all 423 nets is computed by taking the total delay reduction for all nets and dividing by 423.[5]

Observe from the last column that the overall average delay penalty is only 6 ps, or equivalently 1.99%, from avoiding noise. Thus, Buffopt is able to integrate noise into a delay-driven algorithm with virtually no loss in total delay.

Recall that Algorithm 3 is optimal when the buffer library contains a single buffer type, but we could not guarantee optimality for a larger buffer library. The DOpt results in Table 4 form an upper bound on the optimal solution to Problem 2 since DOpt is optimal for delay alone. Observe that even with a buffer library of size 11, BuffOpt solutions are virtually optimal since they are on average within 2% of an upper bound.

Acknowledgments

The authors would like to thank Joe Rahmeh for his help with 3dnoise and the PowerPC design data.

References

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

[2] C. J. Alpert, S. T. Quay and A. Devgan, "Comprehensive Buffer Insertion Techniques", submitted to *IEEE Trans. CAD*, April, 1998.

[3] C. J. Alpert, S. T. Quay and A. Devgan, "Comprehensive Buffer Insertion Techniques for Noise Avoidance", U.S. patent filed, 1997.

[4] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, 1990.

[5] 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.

[6] C. C. N. Chu and D. F. Wong, "A New Approach to Simultaneous Buffer Insertion and Wire Sizing", *ICCAD*, 1997, pp. 614-621

[7] J. Cong, L. He, C.-K. Koh, and P. H. Madden, "Performance Optimization of VLSI Interconnect Layout", *Integration: the VLSI Journal*, 21, 1996, pp. 1-94.

[8] A. Devgan, "Efficient Coupled Noise Estimation for On-Chip Interconnects", *ICCAD*, 1997, pp. 147-151.

[9] 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.

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

[11] M. Z.-W. Kang, W. W.-M. Dai, T. Dillinger, and D. P. LaPotin, "Delay Bounded Buffered Tree Construction for Timing Driven Floorplanning", *ICCAD*, 1997, pp. 707-712.

[12] 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.

[13] L. T. Pillage and R. A. Rohrer. Asymptotic Waveform Evaluation for Timing Analysis. *IEEE Trans. CAD.* 9(4):352-366, April, 1990.

[14] J. Rahmeh, "The 3d-Noise User Guide", *IBM Internal Report*, 1997.

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

[16] L. P. P. P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay", *ISCAS*, 1990, pp. 865-868.

[4] For example, BuffOpt inserted $(0 \cdot 77) + (1 \cdot 161) + (2 \cdot 232) + (3 \cdot 84) + (4 \cdot 2) = 717$ buffers.

[5] E.g., for BuffOpt, the weighted average of 301.1 is given by $((161 \cdot 124.8) + (336.0 \cdot 232) + (1043.4 \cdot 28) + (45.5 \cdot 2))/423$