An Efficient Algorithm for Buffer Insertion in Large Networks Based on Min-cut

ABSTRACT

With shrinking VLSI feature sizes and increasing overall chip areas, buffering has emerged as a potential solution to the problem of growing interconnect delays in modern designs. The problem of buffer insertion in a single net has been the focus for most researchers. However, efficient algorithms for buffer insertion in large networks are required. In this paper, we propose an algorithm based on *length zoom* and *min-cut* for timing constrained minimal buffer insertion in a large network. We compare our approach to a traditional approach based on Lagrangian relaxation. Experimental results demonstrate that the traditional approach is inefficient and significantly sub-optimal. Our approach is found to be efficient and on the average, achieves 39% reduction on the number of buffers inserted in comparison to the traditional approach.

1. INTRODUCTION

Interconnect delays have become dominant in modern deep sub-micron designs with shrinking VLSI feature sizes and increasing chip areas. Buffer insertion is widely used to reduce interconnect delays [1–4, 7, 9–11, 14, 16–18]. Recent projections of historical scaling trends by Saxena et al. [15] predict synthesis blocks to have 70% of their cell count dedicated to interconnect buffers within a few process generations. Consequently, there is an increasing demand for *efficient* buffer insertion approaches.

Van Ginneken [17] proposed a dynamic programming method to buffer insertion in distributed RC-tree networks for minimal Elmore delay [6]. Shi and Li [16] presented an $O(n \log^2 n)$ algorithm for the optimal buffer insertion problem, where n is the number of legal buffer positions. Chen and Zhou [4] presented a flexible data structure to get a universal speed up in buffer insertion. However, all these approaches considered the buffer insertion problem only in a single net.

In reality, it is often required to insert buffers in a large network under a given timing constraint. It is not necessary to optimally buffer each net to get the minimal delay, that is, nets on the non-critical paths are not required to have minimal delays. Therefore, optimally buffering each net leads to over-buffering. If the given timing constraint is larger than the minimal delay that can be achieved by buffering, assignment of various timing budgets on the critical paths in addition to the others would give multiple buffering solutions. We thus need to insert buffers considering a global view instead of a local view. Liu et al. [13] presented a Lagrangian relaxation based algorithm to solve the buffer insertion problem in large networks. It was extended to consider multiple buffer types and feasible buffer locations in [12]. However, Lagrangian relaxation is inefficient in solving large constrained optimization problems. It is therefore impractical to use their technique to solve the buffer insertion problem in large networks. Further, the coefficients used in the object function of Lagrangian relaxation are sensitive and may greatly influence the final results. To the best of our knowledge, there is no good coefficient selection method to obtain the best solution, and our experiments show that the results obtained using a Lagrangian relaxation based technique is significantly sub-optimal in some cases.

In this paper, we present an efficient and timing constrained minimum buffer insertion algorithm for large networks. We assume that the buffers are of a fixed size, and that no forbidden buffer locations exist. The basic idea is to insert more buffers into wires on the min-cuts of the network. Experimental results show that the number of buffers inserted by our algorithm is always less than the number of buffers inserted by [13] under the same timing constraint, and that our algorithm is more efficient than [13].

The rest of this paper is organized as follows. Section 2 presents the problem formulation. In Section 3, the delay models for the interconnects, modules and buffers are presented. In Section 4, we present our buffer insertion algorithm, which is based on *length zoom* and *min-cut techniques*. In Section 5, we report experimental results. Finally, the conclusions are drawn.

2. PROBLEM FORMULATION

The input to our problem is a placed and routed netlist of modules with drivers and loads. Our objective is to insert buffers into wires in large networks such that the timing constraint is met and the number of buffers is minimized.

Using the same circuit representation as in [13], we use a directed acyclic graph (DAG) $G(V, E)$ to represent the circuit of which the vertices correspond to the primary inputs, the primary outputs, tree junctions and module inputs/outputs. Two dummy nodes s and t are introduced: s is connected to all the primary inputs, and t is connected from all the primary outputs. Let E^S be the set of edges connected to s, and let E^T be the set of edges connected to t. The remaining edges are in set E^I, which includes two disjoint sets of edges E^W and E^M, corresponding to wires and the input/output paths of modules, respectively. For an edge $e(u, v)$, let $input(e)$ be the set of edges connected to u.

The notations used in this paper are given in Table 1. The

Table 1: Notations

R_b	the output resistance of a buffer
C_b	the input capacitance of a buffer
t_b	the intrinsic delay of a buffer
K_e	the number of buffers on wire e
L_e	the length of wire e
p_e	the wire length from the input of wire e to the first buffer of wire e
q_e	the wire length from the last buffer of e to the output of wire e
m_e	the wire length between two adjacent buffers in wire e
$\hat{R}_e$	the upstream resistance at the input of wire e
$\hat{C}_e$	the downstream capacitance at the output of wire e
D_e	the delay of the wire e
S_e	the timing slack of the wire e
R	the resistance of a unit-length wire
C	the capacitance of a unit-length wire
L_{th}	wire length threshold
S_{th}	timing slack threshold

problem is formulated as:

$$\min \sum_{e \in E^W} K_e$$

s.t.

$$a_e \leq R_e \ \forall e \in E^T \quad (1)$$

$$a_{e'} + D_e \leq a_e \ \forall e \in E^I \wedge e' \in input(e) \quad (2)$$

$$D_e \leq a_e \ \forall e \in E^S \quad (3)$$

$$K_e \geq 0 \ \forall e \in E^W \quad (4)$$

where K_e is the number of buffers on edge e, D_e is the delay of edge e and is influenced by K_e, a_e is the arrival time at e, and R_e is the required time at e.

In the buffer insertion problem, the change of the delay of one element influences the delays of many other elements. As shown in Figure 1, a net is composed by wires a, b and c. If a buffer B is inserted into wire c, the delay of c is changed, and according to Elmore delay model, the delay of a also change, so the delay from s_0 to s_2 also changes. Actually, since the delay of the driver gate is related with the load capacitance, it may change too.

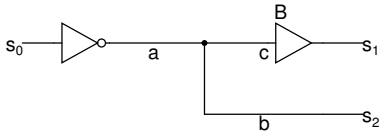


Figure 1: The influence of an inserted buffer.

Our idea to insert minimal number of buffers in large networks is stimulated by the min-cut concept. As shown in Figure 2, we want to insert minimal number of buffers such that the maximal delay from s to t satisfies the timing constraint. One possible solution of the buffer positions is shown by dotted line. But actually there is a better solution that is shown by bold line, where the buffers are inserted into the wires on the min-cut. According to this, our basic idea is to iteratively insert buffers into wires on min-cuts.

3. DELAY COMPUTATION

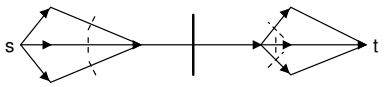
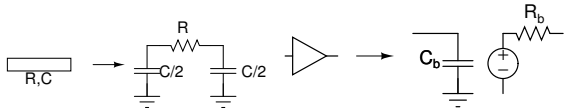
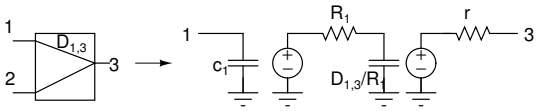


Figure 2: The benefit of min-cut based technique.

We use the same models of wires, modules and buffers with [13]. For convenience, we present the models in Figure 3.



(a) Wire model (b) Buffer model



(c) Gate model (from pin 1 to pin 3)

Figure 3: Delay models of wires, buffers and gates.

Given a wire with buffers inserted, we can easily compute the delays. In this paper, we only consider the single buffer type, then from [13], we know that the wire lengths between two consecutive buffers on one wire are equal. As shown in Figure 4, when $K_e > 0$, the delay of wire e is equal to

$$\begin{aligned} D_e &= Rp_e(Cp_e/2 + C_b) \\ &\quad +(K_e - 1)(R_b(C_b + Cm_e) + t_b + Rm_e(C_b + Cm_e/2)) \\ &\quad +t_b + R_b(\hat{C}_e + Cq_e) + Rq_e(\hat{C}_e + Cq_e/2), \end{aligned} \quad (5)$$

We need to consider the contribution of the capacitance of e to the delay of $input(e)$, so let

$$F_e = D_e + \hat{R}_e(Cp_e + C_b). \quad (6)$$

Then let

$$\frac{\partial F_e}{\partial p_e} = 0,$$

and

$$\frac{\partial F_e}{\partial q_e} = 0,$$

then we can compute the optimal values of p_e and q_e for a given K_e such that F_e is minimized. The optimal values of p_e and q_e are given in [13], and for convenience, we list them again in Table 2. When we get the optimal p_e and q_e for a given K_e, the delay of wire e for K_e is easily computed according to Eq. 5.

Figure 4: A wire with buffers.

4. MIN-CUT BASED BUFFER INSERTION

Satisfying the given timing constraints at the primary outputs of a circuit using minimal number of buffers requires minimizing the maximal delay of the critical paths by buffering. However, given a network, we do not know the critical paths before we get the final buffer insertion solution. As a result, we need to push down the arrival time at node t iteratively until it meets the timing constraint. In each iteration, we find the critical paths and insert buffers into them. As illustrated earlier, the non-critical paths influence the delays on the critical paths. As a result, we need to insert decoupling buffers into non-critical branches.

A subgraph containing only the critical paths in the circuit is called the *critical subgraph* of the circuit. We need to insert a minimal number of buffers in the critical subgraph while maximally pushing down the arrival time at node t. We define the *gain* of one buffer insertion activity to be the timing reduction at node t when that activity is performed. We thus need to find the best buffer insertion positions to maximize the summation of the gains in a sequence of buffer insertion activities.

We have two observations:

- The gain in inserting a buffer in a wire is likely larger for a longer wire in comparison to a shorter one. Based on this, we select a *length threshold*, denoted as L_{th}, such that all wires satisfying

  $$L_e/(K_e + 1) \leq L_{th}$$

  are ineligible for the buffer insertion. In each iteration L_{th} is decreased. This technique, called *length zoom* (LZ), can always keep a small number of wire candidates for buffer insertion. This greatly reduces the size of the buffer insertion problem.

- When more critical paths pass through a wire e than another wire e', the number of buffers inserted in e is expected to be less than the number of buffers inserted in e' for the same gain. Based on this, we insert the buffers into the wires in the *min-cut* of the critical subgraph.

We next present our timing constrained algorithm called CutBIN for minimal buffer insertion. The flow of CutBIN is shown in Figure 5. We assume no buffers in the circuit at the start of our algorithm.

```
Algorithm CutBIN(G)
 1  maxdelay ← ComputeTimingAndSlack(G);
 2  L_th ← half of the maximal wire length;
 3  SetCapacity(G);
 4  while maxdelay>REQ_TIME and L_th > lower bound
 5        do Find the min-cut of G;
 6           Insert buffers into the nets in the min-cut;
 7           if no buffers inserted in this iteration
 8              then L_th ← L_th/α (α > 1);
 9              else maxdelay ← UpdateTimingSlack(G);
10                   UpdateCapacity(G);
```

Figure 5: Min-cut based buffer insertion algorithm

4.1 Min-cut based buffer insertion

In CutBIN, we do not explicitly extract the critical subgraph of G. The critical subgraph here contains not only the wires on the critical paths, but also the wire candidates to be critical in next iterations. First, we compute the slacks and timing information in ComputeTimingAndSlack. We use the given timing constraint REQ_TIME as the criterion to compute slacks.

Then we set the capacities of edges. For any edge e, if $S_e > S_{th}$, it should be excluded from the critical subgraph, so we set the capacities of the modules and the wires satisfying $S_e > S_{th}$ to be 0. Modules should be excluded from the min-cut, so the capacities of the other modules are set to be INFINITY. Each net can be considered as a tree. The algorithm to set the capacities of wires in a net is shown in Figure 6. The argument *root* is the root of the net. The pseudocode of the subroutine RecursiveSetNetCapacity is shown in Figure 7. Before the call of SetNetCapacity, the capacity of any wire e satisfying $S_e > S_{th}$ is set to be 0, and the capacities of other wires are set to be INFINITY, which is done in the timing and slack computation or update step. At the end of SetNetCapacity, the capacities of wires in a net satisfy the following condition: if all the branches of the net are not in the current critical subgraph, the capacity of the root of the net is 0; otherwise if there exists a wire e satisfying $L_e/(K_e + 1) > L_{th}$ and $S_e \leq S_{th}$, the capacity of the root of this net is 1. This setting guarantees that the found min-cut contains only the *roots* of nets with branches on the critical subgraph. Also, the integer property of the capacities can speed-up the execution of the max-flow algorithm [8].

After the setting of capacities, we use Ford-Fulkerson algorithm [8] to compute the maximal flow of G and then to find the min-cut of G.

As shown in Figure 8, the delay of the module G_1 is equal to

$$t_{G_1} + rC(L_{e_1} + C_{G_2}) + r(Cp_{e_2} + C_b),$$

where t_{G_1} is the intrinsic delay of G_1, r is the output resistance of G_1, and C_{G_2} is the input capacitance of G_2. When p_{e_2} is larger, the delay of G_1 increases, so the delay of the critical path increases greatly. In order to minimize this influence, we need to insert decoupling buffers into the non-critical branches.

Now we insert buffers into the nets with their roots in the min-cut. We use a recursive procedure to insert buffers.

Table 2: Optimal p_e and q_e for a given $K_e > 0$

	$p_e = 0$	$0 < p_e \leq L_e$
$q_e = 0$	$p_e = 0$ $q_e = 0$	$p_e = \frac{L_e+(K_e-1)(R_b-\hat{R}_e)/R}{K_e}$ $q_e = 0$
$0 < q_e \leq L_e$	$p_e = 0$ $q_e = \frac{L_e+(K_e-1)(C_b-\hat{C}_e)/C}{K_e}$	$p_e = \frac{L_e+K_e(R_b-\hat{R}_e)/R-(C_b-\hat{C}_e)/C}{K_e+1}$ $q_e = \frac{L_e-(R_b-\hat{R}_e)/R+K_e(C_b-\hat{C}_e)/C}{K_e+1}$

```
Procedure SetNetCapacity(root)
  if root.slack > S_th
    then root.capacity ← 0
    else root.capacity ← INFINITY
         if root.l /(root.k +1) > L_th
           then root.capacity ← 1
         if root.capacity ≠ 1 and root.childnum > 0
           then flag ← 0
                RecursiveSetNetCapacity(root, flag)
                if flag = 1
                  then root.capacity ← 1
```

Figure 6: An algorithm to set the capacities of wires in a net rooted at *root*

```
Procedure RecursiveSetNetCapacity(root, flag)
  for each child of root
      do if child.slack > S_th
           then child.capacity ← 0
                continue
         child.capacity ← INFINITY
         if child.l /(child.k +1) > L_th
           then flag ← 1
                return
           else RecursiveSetNetCapacity(child, flag)
                if flag = 1
                  then return
```

Figure 7: Recursively set capacities of wires in a net rooted at *root*.

The phseudocode is shown in Figure. 9. We perform depth-first traversal on the net starting from the *root*. For each encountered wire e, if $L_e/(K_e+1) > L_{th}$, we insert buffers until $L_e/(K_e+1) \leq L_{th}$. Then for each child of e, if its slack is larger than S_{th}, it is on a non-critical branch, so we insert one decoupling buffer at a position near the start point of this child, and the depth-first traversal does not traverse the children of this child; otherwise the recursive procedure continues the traversal. Here when we insert one buffer, we do not fix the buffer position, actually we only record p_e, q_e and K_e for each wire e, and the positions of the inserted buffers are flexible.

It is possible that no buffers are inserted in some iteration of CutBIN, so the arrival time at node t does not decrease, then we decrease the length threshold L_{th}. If we have inserted new buffers, we need to update the timing, slacks and capacities.

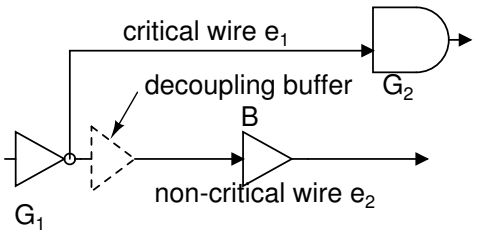


Figure 8: Decoupling technique.

```
Procedure RecursiveInsertBuffers(root)
  if root.l /(root.k +1) > L_th
    then Insert buffers to root until
           root.l /(root.k +1) ≤ L_th
  for each child of root
      do if child.slack > S_th
           then ▷ (child not in the critical subgraph)
                Insert a decoupling buffer at the starting
                  point of child if there is no decoupling
                  buffer in child
           else RecursiveInsertBuffers(child)
```

Figure 9: Recursively insert buffers into a net rooted at wire *root*

5. EXPERIMENTAL RESULTS

5.1 Comparison results

We have implemented the CutBIN algorithm in C. We use the parameters from 100-nanometer technology [5]. We got four test cases from Liu, and we generated additional four cases using the case generator in [13]. All experiments are run on a Linux PC with 2.4G Hz Xeon CPU and 2.0 GB memory.

In order to test the benefit of the min-cut based buffer insertion, we got the source code of the algorithm in [13] from Liu for comparison. The comparison results of CutBIN and [13] are shown in Table 3. The 3rd column is the ratio of the timing constraint over the minimal required time at node t. The "Ratio1" column and the "Ratio2" column are the ratios of the arrival time at t got by [13] and CutBIN over the timing constraint respectively. The "Reduction" column is the reduction percentage of the number of buffers inserted by CutBIN compared with the number of buffers inserted by [13]. The last column is the speed-up of CutBIN over [13]. We can see that CutBIN is much more efficient than [13], and it can achieve 39% reduction of the number of inserted buffers on average compared with [13].

An example to show the benefit of the min-cut based

Table 3: Comparison results of CutBIN and [13]

circuit		Tightness	[13]			CutBIN			Reduction	Speed-up
Module#	Wire#	of constr.	Buffer#	Ratio1	Time (s)	Buffer#	Ratio2	Time (s)		
22	98	1.20	172	1.04	0	146	0.99	0	17%	1×
44	197	1.20	305	1.01	5	202	0.99	0.04	34%	125×
81	398	1.20	606	1.01	16	334	0.99	0.14	45%	17×
159	799	1.20	887	1.02	10	583	1.00	0.67	34%	15×
258	1037	1.20	1195	0.96	27	524	1.00	0.37	56%	73×
505	2039	1.20	2331	0.97	65	1367	1.00	1.8	41%	36×
2514	10039	1.20	10467	0.98	343	6265	1.00	22	40%	16×
5034	20038	1.20	20721	0.98	706	11098	1.00	69	46%	10×
Average									39%	37×

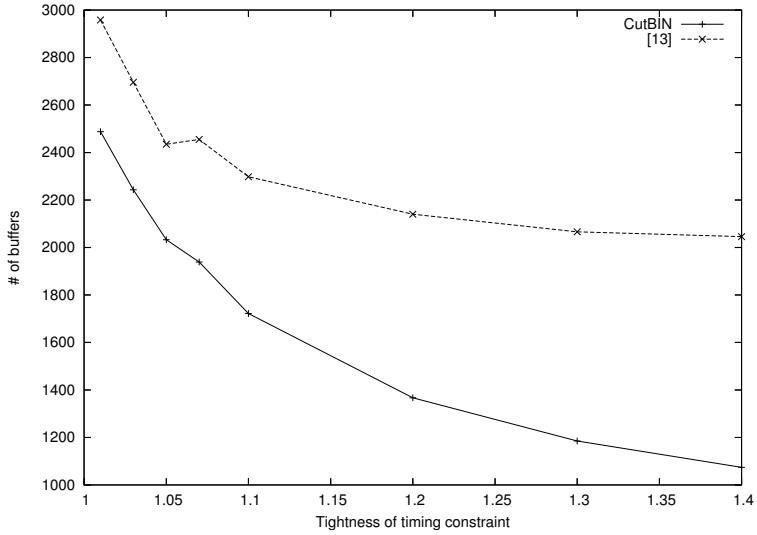


Figure 11: The influence of the tightness of timing constraint on the number of inserted buffers.

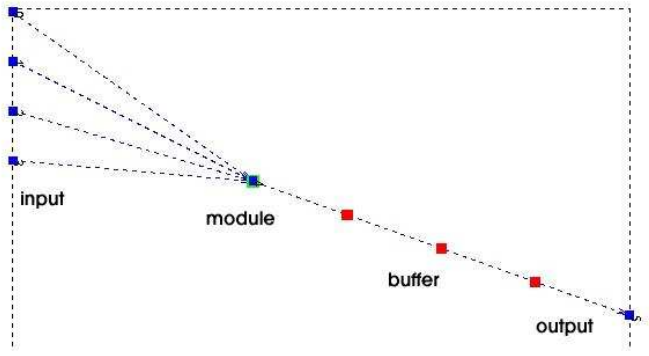


Figure 10: A solution generated by CutBIN.

buffer insertion algorithm is shown in Figure 10. The network contains four inputs, one output, and one module. We can see that our algorithm inserts all the buffers into the wires on the min-cut.

5.2 Analysis

Intuitively, our algorithm is more applicable for the cases with loose timing constraints. Using the case with 505 modules and 2,039 wires as an example, we test the influence of the tightness of timing constraint on the number of inserted buffers. As shown in Figure 11, CutBIN is more sensitive to the changes of the tightness of timing constraint than the algorithm in [13]. Especially, CutBIN performs much better when the timing constraint is looser.

In order to find the best choice for the length threshold decreasing speed (α), we use the test case above as an example to show the influence of α on the number of inserted buffers and the running time. S_{th} is set to be 2,000 ps. As shown in Figure 12, a good choice of α is in [1.3, 2.8], where the running time and the number of buffers are both acceptable. In our experiments, [1.3, 2.8] is always a good candidate range for the selection of α.

In order to find the best choice for S_{th}, we use the same test case as an example to show the influence of S_{th} on the number of inserted buffers and the running time. α is set to be 1.03. As shown in Figure 13, a good choice of S_{th} is about 2,000 ps, where the running time and the number of buffers are both acceptable. In our experiments, we select 2,000 ps as the value of S_{th} and get good results.

6. CONCLUSIONS

With shrinking VLSI feature sizes and increasing overall chip areas, buffering has emerged as a potential solution to the problem of growing interconnect delays in modern de-

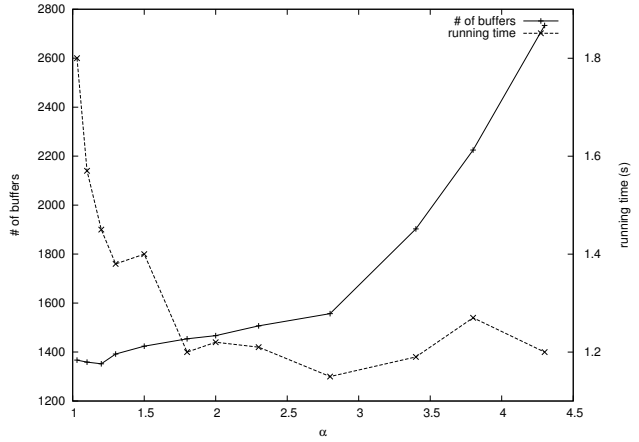


Figure 12: The influence of α on the number of buffers and running time in CutBIN**.**

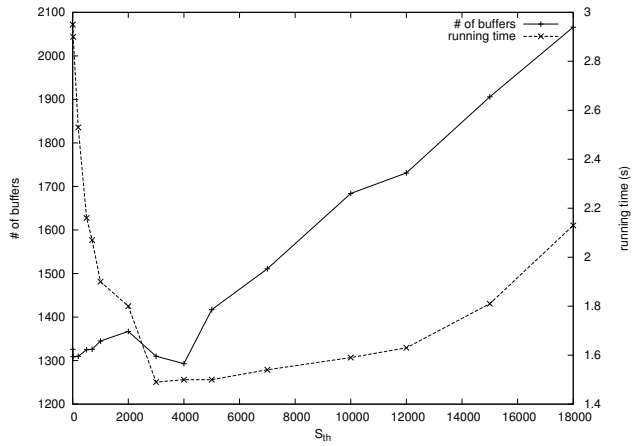


Figure 13: The influence of S_{th} on the number of buffers and running time in CutBIN**.**

signs. The problem of buffer insertion in a single net has been the focus for most researchers. In this paper, we propose an algorithm based on *length zoom* and *min-cut* for timing constrained minimal buffer insertion in a large network. We compare our approach to a Lagrangian relaxation based buffer insertion algorithm proposed by Liu et al. [13]. Experimental results demonstrate that our approach is more efficient and on the average, achieves 39% reduction on the number of buffers inserted in comparison to the latter.

7. REFERENCES

[1] C. J. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In *DAC*, pages 588–593, 1997.

[2] C. J. Alpert, M. Hrkic, and S. T. Quay. A fast algorithm for identifying good buffer insertion candidate locations. In *ISPD*, pages 47–52, 2004.

[3] C. J. Alpert, J. Hu, S. S. Sapatnekar, and C. N. Sze. Accurate estimation of global buffer delay within a floorplan. In *ICCAD*, pages 706–711, 2004.

[4] R. Chen and H. Zhou. A flexible data structure for efficient buffer insertion. In *ICCD*, pages 216–221, 2004.

[5] J. Cong and Z. Pan. Interconnect performance estimation models for synthesis and design planning. In *Workshop Notes of Intl. Workshop on Logic Synthesis*, pages 427–433, 1998.

[6] W. C. Elmore. The transient response of damped linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[7] J. Lillis et al. New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing. In *DAC*, pages 395–400, 1996.

[8] J. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[9] M. Kang, W. W.-M. Dai, T. Dillinger, and D. LaPotin. Delay bounded buffered tree construction for timing driven floorplanning. In *ICCAD*, pages 707–712, 1997.

[10] V. Khandelwal, A. Davoodi, A. Nanavati, and A. Srivastava. A probabilistic approach to buffer insertion. In *ICCAD*, pages 560–567, 2003.

[11] 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 Trans. Solid-State Circuits*, 31:437–447, 1996.

[12] I.-M. Liu, A. Aziz, and D. F. Wong. Meeting delay constraints in DSM by minimal repeater insertion. In *DATE*, pages 436–440, 2000.

[13] I.-M. Liu, A. Aziz, D. F. Wong, and H. Zhou. An efficient buffer insertion algorithm for large networks based on Lagrangian relaxation. In *ICCD*, pages 210–215, 1999.

[14] T. Okamoto and J. Cong. Buffered Steiner tree construction with wire sizing for interconnect layout optimization. In *ICCAD*, pages 44–49, 1996.

[15] P. Saxena, N. Menezes, P. Cocchini, and Desmond A. Kirkpatrick. The scaling challenge: Can correct-by-construction design help? In *ISPD*, pages 51–58, 2003.

[16] W. Shi and Z. Li. An O(nlogn) time algorithm for optimal buffer insertion. In *DAC*, pages 580–585, 2003.

[17] L. P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *ISCAS*, pages 865–868, 1990.

[18] H. Zhou, D. F. Wong, I-Min Liu, and Adnan Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. DAC, 1999.