Buffer Insertion for Combinational Circuits

Abstract- We propose a buffer insertion algorithm for combinational circuits. The algorithm finds a buffering solution for the entire circuit such that the buffer cost is minimized and the timing requirements are satisfied. The algorithm iteratively partitions the circuit along the most critical path to arrive at smaller subcircuits. The core of the algorithm is the procedure that finds an optimal solution for the subcircuits in an efficient manner. Experimental results on ISCAS85 circuits show our algorithm can reduce the number of buffers by 90% compared with the traditional net-based algorithms.

I. Introduction

Buffer insertion is an effective technique for reducing interconnect delay. As interconnect delay is posing a limit to the performance of VLSI circuits, cost of required buffering resources to meet the timing constraints is exploding [1]. Consequently, optimizing buffer cost has become paramount. For buffer insertion at the net level, van Ginneken [2] proposed an $O(n^2)$ time dynamic programming algorithm to maximize the slack. Lillis *et al.* [3] extended it to minimize the buffer cost while satisfying the timing requirements. A great amount of research has been done in the past few years, such as [6], [7], [9], [8].

In real application, however, the objective is to reduce path delay in combinational circuits. Therefore, buffer insertion should be performed at the circuit level. A simple minded approach is to apply the net-based van Ginneken algorithm to circuit, one net at a time from primary output to primary input. Although this approach guarantees the slacks at the primary input nodes are maximized, too many buffers will be used since van Ginneken's algorithm does not control buffer cost. To use Lillis' net-based algorithm that controls buffer resources will also run into problem, due to the reconvergence in the combinational circuit, and the fact that Lillis algorithm has exponential time complexity even for a single net. A Lagrangian relaxation algorithm for circuit level buffer insertion was proposed in [4], [5]. However, both works have a restrictive assumption that buffers must be inserted at every Steiner node. In practice, whether or not buffers are inserted at certain node depends on timing constraints and availability of space. Furthermore, their algorithms do not scale very well. In [4], the CPU time explodes for larger test cases.

The rest of the paper is organized as follows. Section II presents notations and gives an overview of the algorithm. Later sections describe the steps of the algorithm in detail. Section III describes the procedure to solve the smaller subcircuits optimally. Section IV describes the criterion for deleting certain nets from the circuit for reducing the problem complexity. Section V describes the procedure to partition the circuit in smaller subcircuits. The experimental results and conclusions are presented in Section VI.

II. Preliminaries and Overview

We represent a combinational circuit as a Directed Acyclic Graph (DAG) $G = (V, E)$. The set of nodes $V = V_{pi} \cup V_{po} \cup V_{gi} \cup V_{go} \cup V_t$, where V_{pi}, V_{po}, V_{gi} and V_{go} are the sets of nodes for primary input, primary output, gate input and gate output, respectively, and V_t is the set of nodes in the interconnect routing tree. The set of edges $E = E_w \cup E_g$, where

$$E_w \subset (V_{pi} \cup V_{go} \cup V_t) \times (V_{po} \cup V_{gi} \cup V_t)$$

is the set of interconnect wires and $E_g \subset V_{gi} \times V_{go}$ is the set of input-to-output paths within the gates. Fig. 1 shows an example combinational circuit.

We are given a buffer library $\mathbf{B}$. The locations where buffers can be inserted are given as a function $f : V_t \rightarrow 2^{\mathbf{B}}$. Under this definition, each node in the interconnect routing tree allows certain types of buffers, or no buffer. Each buffer type $b_i \in \mathbf{B}$ is modeled by intrinsic delay $K(b_i)$, driving resistance $R(b_i)$, input capacitance $C(b_i)$ and cost $W(b_i)$. The cost of a buffer can be either the area or the power or other criteria, depending on our optimization objective.

Each interconnect wire is modeled as a π type RC circuit: each edge $e \in E_w$ is associated with resistance $R(e)$ and capacitance $C(e)$. Each gate in the circuit is modeled similar to the buffers. Thus each edge $e \in E_g$ is associated with intrinsic delay $K(e)$, each node $v \in V_{gi}$ is associated with input capacitance $C(v)$ and each node $v \in V_{go}$ is associated with driving resistance $R(v)$.

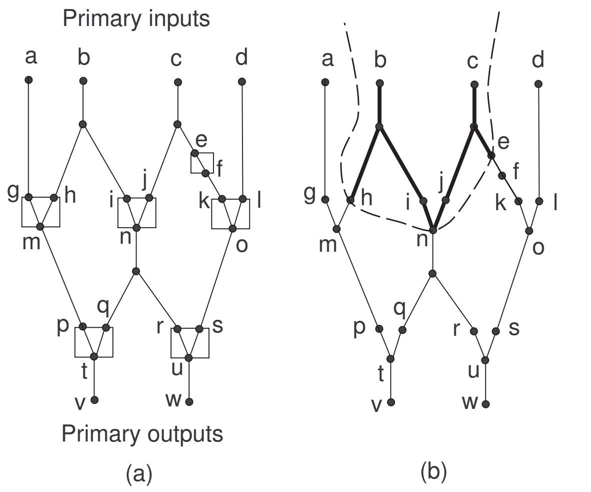

Fig. 1. (a) DAG representation of a combinational circuit. (b) Sample front $\{h, n, e\}$.

Timing requirements are specified in terms of Required Arrival Time $RAT(v)$ for each primary output node $v \in V_{po}$, and Departure Time $DT(v)$ for each primary input node $v \in V_{pi}$. Following previous researchers [2], [3], [6], [7], [10], [11], we use Elmore delay for the interconnect and the linear delay for gates and buffers. For each edge $e = (v_i, v_j)$, signals travel from v_i to v_j. The Elmore delay of e is

$$D(e) = R(e)\left(\frac{C(e)}{2} + C(v_j)\right), \qquad (1)$$

where $C(v_j)$ is the downstream capacitance at v_j. For any gate or buffer b at vertex v_j, the gate or buffer delay is

$$D(v_j) = K(b) + R(b) \cdot C(v_j), \qquad (2)$$

where $C(v_j)$ is the downstream capacitance at v_j. For a gate or inserted buffer b, the capacitance viewed from the upper stream is $C(b)$.

Consider a connected subgraph $G' = (V', E')$ where $V' \subset V$, $E' \subset E$ and $V'_t = V \cap V_t$. A buffer assignment is a function $\alpha(G') : V'_t \to \mathbf{B} \cup \{\emptyset\}$ that specifies the type of buffer inserted for each node in V'_t. The cost of α is denoted as $W(\alpha)$, which equals the total buffer cost assigned by α.

A circuit graph with all the required parameters, timing requirements and buffer library is the input to our algorithm. The algorithm then recursively partitions the circuit to smaller subcircuits until the subcircuit can be solved efficiently.

Following is an outline of the algorithm:

```
1: if $G$ is "small-enough" then
2:     Solve $G$ directly
3: else
4:     Find the most critical path $P^*$
5:     Insert buffers in $P^*$
6:     Partition $G$ into $G_1, \ldots, G_k$
7:     Recurse for each $G_i$
8: end if
```

III. Solving the subcircuits directly

In this section, we present an algorithm that solves the problem optimally, meaning that the algorithm finds the buffer insertion of a combinational circuit with minimum total buffer cost, and with all timing requirements satisfied. Since the worst-case running time is exponential, we call this algorithm as a subroutine only for circuit with no more than 30 gates.

The traditional net-based algorithms that propagate a single node can not be applied to circuits due to reconvergence. The main idea of this algorithm is to propagate a *front* from primary inputs to primary outputs or in the opposite direction. The *front* F is defined as a cut in G such G is partitioned in in two subgraphs G_1 and G_2 and all edges directed toward the nodes in F are in G_1 and all edges directed away from F are in G_2. Figure 1(b) shows a sample front (h, n, e). In this case, G_1 is the part of the circuit represented by thick edges and G_2 is represented as thin edges.

Let primary inputs to primary outputs be called a *forward direction* and the opposite be called *reverse direction*. Fig. 1(b) illustrates the front propagation. It shows a sample front (h, n, e) by the dotted curve. If this front is being propagated in forward direction, then it has been propagated over the subcircuit represented by thick edges. Conversely, if this front would have been propagated in reverse direction, it would have covered the subcircuit represented by thin edges.

As the front is propagated, we evaluate the possible buffer assignments for the nets being covered. For a front $F = \{v_1, v_2, \ldots, v_k\}$, any buffer assignment $\alpha(F)$ while propagating in forward direction is represented as a vector

$$(W(\alpha), D(v_1, \alpha), \ldots, D(v_k, \alpha)),$$

where $D(v_1, \alpha)$ is the Departure Time (DT) at v_i under α. Similarly, while propagating in reverse direction, the assignment is represented as a vector

$$(W(\alpha), Q(v_1, \alpha), \ldots, Q(v_k, \alpha)),$$

where $Q(v_1, \alpha)$ is the Required Departure Time (RDT) at v_i under α.

Two assignments α_1 and α_2 for G', we say α_1 dominates α_2 if

- α_2 has higher cost: $W(\alpha_2) \geq W(\alpha_1)$, and
- α_2 has better timing: $D(v, \alpha_2) \geq D(v, \alpha_1)$ for every $v \in F(G')$ or $Q(v, \alpha_2) \leq Q(v, \alpha_1)$ for every $v \in F(G')$.

The set of *nonredundant assignments* for G', denoted as $N(G')$ is a set of assignments such that no assignment in $N(G')$ dominates any other assignment in $N(G')$ and any buffer assignment for G' is dominated by some assignment in $N(G')$.

The direction of front propagation is determined by the size of nets in the circuit under consideration. If total number of buffer locations on 2-pin nets are more than the total number of buffer locations on multi-pin nets, forward direction is chosen. Otherwise, reverse direction is chosen.

The following subsections describe the algorithm for propagation in forward direction. Similar scenarios apply to propagating in reverse direction as well. The flow of this algorithm is as shown below.

```
1: Let Front $F = \emptyset$
2: Non-redundant assignments $N(F) = \emptyset$
3: **while** Complete circuit is not traversed **do**
4:     Select a subcircuit $G$ to grow the front
5:     Propagate $F$ over $G$
6:     Update $N(F)$.
7: **end while**
8: **return** candidate with minimum cost in $N(F)$
```

A. Propagating the front

Consider a subcircuit G' with set of fan-in nodes $I(G') = (i_1, \ldots, i_l)$ and fan-out nodes $O(G') = o_1, \ldots, o_m$. Given departure time values for each node in I, any buffer assignment $\alpha(G')$ can be expressed as $\alpha(G') = \{W(\alpha), D(o_1, \alpha), \ldots, D(o_m, \alpha)\}$

Consider a front F to be propagated over subcircuit G' such that $I(G') \subseteq F$. Then, front after propagation $F' = \{F \setminus I(G')\} \cup O(G')$. Also an assignment $\alpha(F)$ will have an arrival time value at each node in $I(G')$. Let $\alpha(G')$ be the assignment for G' with arrival time values in $\alpha(F)$. Then the resultant assignment for F' is computed as follows:

$$W(\alpha(F')) = W(\alpha(F)) + W(\alpha(G'))$$

$$D(u, \alpha(F')) = \begin{cases} \emptyset & \text{if } u \in I(G') \\ D(u, \alpha(G')) & \text{if } u \in O(G') \\ D(u, \alpha(F)) & \text{otherwise} \end{cases}$$

The subcircuit G' for propagating the front can be simply be a single net. G' can be chosen such that the assignments in the subcircuit and resultant propagated front can be pruned efficiently.

B. Computing non-redundant candidates for a net

Consider a net represented as a tree T with root node r and leaf nodes $l_1, l_2, \ldots, l_k$. A 2-pin net is a special case tree with only one leaf.

A set of non-redundant candidates $N(T)$ is computed with a framework similar to (Q, C, W) framework [3], [9]. This framework is represented as $(W(\alpha), D(l_1, \alpha), \ldots, D(l_k, \alpha))$ Basic operation of this framework is same as (Q, C, W) framework except the computation of delay values while propagating the candidates. Similar to the (Q, C, W) framework, candidates are generated by traversing the net from the leafs toward the root. Also, the three basic operations during the traversal are adding a wire, adding a buffer and merging two branches. When propagating the candidates in these scenarios, $W(\alpha)$ and $C(\alpha)$ are computed in the exact same manner as the (Q, C, W) framework. To propagate the $D(v_i, \alpha)$ values, while adding a wire or a buffer, delay values are incremented by wire or buffer delay respectively. In merging operations, the delay values remain unchanged.

C. A* pruning techniques

We define $Q_{best}(v_i)$ as maximum value of Required Departure Time that can be achieved at node v_i with all possible buffer assignments. Also $Q_{worst}(v_i)$ is defined as RDT at node v_i when no buffers are inserted in the circuit.

Basic mechanism for pruning the assignments is comparing the assignments for the same front F against each other and deleting the inferior ones by the pruning criterion described in section II earlier. Here we discuss following additional techniques that help in pruning more assignments.

- Q_{best} based filtering:

An assignment $\alpha(F)$, where $F = \{v_1, v_2, ..., v_k\}$, can be deleted if:

$$D(v_i, \alpha) > Q_{best}(v_i).$$

It can be easily seen that such an assignment certainly won't satisfy the timing requirement. This operation is called filtering because this condition is checked in propagation step itself and generation of such assignments having insufficient buffering is prevented. Also, whenever the wave-front engulfs a primary output node v_i, all the assignments that survive Q_{best} based filtering satisfy the RAT at node v_i. Hence node v_i can be dropped from the front so that the exact value of $D(v)$ will be disregarded while comparing the assignments with each other resulting in additional pruning.

To find Q_{best} we run van Ginneken's algorithm [2] on each net from primary output towards primary input, using the RAT for all primary output nodes.

- Q_{worst} based filtering:

Consider an assignment $\alpha(F)$ such that

$$D(v_i, \alpha) \leq Q_{worst}(v_i)$$

In a way, this condition suggests that sufficient buffers have been added in the transitive fan-in of node v_i under α such that even minimum buffering in the net rooted at node v_i is sufficient for α to meet the timing requirements. Thus $\alpha(F)$ is propagated only with the minimum cost assignment on the net rooted at node v_i. This condition is also checked in propagation step itself and generation of such unnecessarily buffered assignments is prevented.

- Using $minQW$ pairs:

A $minQW$ pair represented as a $(Q(v_i), W)$ doublet at any node v_i means to satisfy the timing requirements when $DT(v_i) = Q(V_i)$, at least W amount of buffer-cost has to be spent in the transitive fan-out cone of node v_i. A set of such $minQW$ pairs at a node helps prune more candidates. Following example illustrates this.

Consider two assignments $\alpha_1(F)$ and $\alpha_2(F)$ for a particular wave-front $F = \{v_1, v_2, ..., v_k\}$ such that,

$$W(\alpha_1) = 50, D(v_1, \alpha_1) = 1010,$$

$$W(\alpha_2) = 55, D(v_1, \alpha_2) = 1000,$$

and $D(v_i, \alpha_1) \leq D(v_i, \alpha_2) fori = 2, 3, ..., k$. Since α_2 has better arrival time value at node v_1, it will not get pruned by the basic pruning criterion . If suppose there are two $minQW$ pairs at node v_1 namely

$$(W = 10, Q(v_1) = 1000)$$

$$(W = 11, Q(v_1) = 1100)$$

This means α_1 will make a better candidate in future even though presently it has inferior arrival time value at node v_1. Hence α_2 can be pruned even though it has a better arrival time at node v_1.

We compute the $minQW$ pairs only for the nodes not having reconvergence in their transitive fan-out cone. To find a set of $minQW$ pairs, we run (Q, C, W) framework on each net from primary output towards primary input, using the RAT for all primary output nodes. Also, for a particular front, transitive fan-out cones of the nodes on the front may overlap with each other. Therefore, the $minQW$ data can be used only for a subset of front nodes such that transitive fan-outs of any two nodes in this subset do not overlap with each other.

D. Processing the circuit in reverse direction

Processing the circuit in reverse direction is similar to the forward direction approach described in detail above. While propagating a front over a multi-pin net, (Q, C, W) framework is used. The counterpart of $minQW$ pairs is $minDW$ pairs. Similarly counterpart of Q_{best} based filtering is based D_{best} filtering. Note that there is no counterpart of the Q_{worst} based filtering. This is because the assignment required to satisfy the timings may demand a higher value of arrival time at some leaf v_i of a multi-pin net that is more than the arrival time at v_i when no buffers are inserted in the circuit.

Also an important difference about computation of D_{best} as opposed to computation of Q_{best} is that it is affected by the reconvergence in the circuit and can not be exactly computed by using Van Ginneken [2] like procedure. But note that we do not require the exact value of D_{best}. Suppose that D_{best} is assigned an approximate value that is lesser than its exact value, the effectiveness D_{best} based filtering is reduced but it does not result in deletion of an assignment that should not have been deleted. Such an approximate value lesser than the exact value represents an infeasible

D_{best} but it does not affect the final solution and it can be computed effectively by disregarding the effect of reconvergence and just following a similar procedure used for computing Q_{best}.

IV. Trim circuit

We define $D_{worst}(u)$ for node u as the $RDT(u)$ when no buffers are inserted in the circuit. Also a *trimming condition* is defined as follows.

$$D_{worst}(u) < Q_{worst}(u)$$

Any node u satisfying the trimming condition indicates that the arrival time as well as slack at node u are not critical for satisfying the timing requirement of the circuit. Thus, the value $Q_{worst}(u)$ can be assigned as RAT at node u. Such a value of RAT may not be satisfiable in case a buffer assignment necessary to satisfy the timing requirements has arrival time at u more than $D_{worst}(u)$. But such a case is unlikely to occur in a circuit.

Thus, if node u satisfies the trimming condition as well as an assignment $RAT(u) = Q_{worst}(u)$ is satisfiable for the whole circuit, then this value is assigned as $RAT(u)$. A buffer assignment $\tilde{\alpha}$ is fixed for the net T rooted at u such that,

$$W(\tilde{\alpha}) = \min_{\alpha \in N(T)} \{W(\alpha)\}$$

Also net N is deleted from the circuit since $RAT(u)$ is assigned and buffer assignment for N is fixed.

Note that for a gate output node that satisfies trimming condition, all the corresponding gate input nodes satisfy the trimming condition. Every gate input node in the circuit that satisfies this condition will become a *pseudo primary output* node because of the newly assigned RAT value. Trimming reduces the complexity of the problem and may also partition the circuit.

V. Partition Circuit

A. Finding Most Critical Path P^*

The *criticality* of a path $P(u, v)$, where u is a primary input and v is a primary output, is defined as follows:

$$Criticality(P(u,v)) = \frac{1}{RAT(v) - D(v,\alpha)},$$

where α is a buffer assignment that maximizes the slack at every node in the circuit to be partitioned. Note that if $RAT(v) - D(v, \alpha) < 0$, then there is no feasible solution. Therefore we assume for now $RAT(u, \alpha) - DT(u) \geq 0$. A path P^* is *most critical* if

$$Criticality(P^*) = \max_{u \in V_{pi}, v \in V_{po}} \{Criticality(P(u,v))\}$$

To find P^*, we run van Ginneken's algorithm [2] on each net from primary output towards primary input, using the RAT for all primary output nodes. At each Steiner node, we record the branch that gives the Q to be propagated towards the primary input.

After van Ginneken's algorithm is finished, the most critical path P^* must start from the primary input node u that has the minimum $Q(u) - DT(u)$. To find the rest of P^*, we trace the path towards the primary output using the recorded information.

B. Inserting Buffers in P^*

Here, we give a new buffer insertion algorithm for the critical path. The critical path P^* is from a primary input to a primary output and consists of k fanout routing trees $T_1, T_2, \ldots, T_k$. Let n_i be the number of buffer positions in T_i, r_i be the root of tree T_i, and S_i be the set of sinks of tree T_i, which are gate inputs or primary outputs driven by r_i. Each routing tree T_i does not include any other gate except the root r_i and sinks in S_i. We perform buffer insertion with cost minimization on the path P^* as follows.

Initially, a single candidate (Q, C, W) is assigned for each sink s in tree T_k, where Q is the sink RAT, C is the load capacitance and $W = 0$. The sink RAT is obtained from the assignment made while finding P^* as described above. Then we apply (Q, C, W) framework on T_k. At the root r_k, all nonredundant candidates (Q, C, W) will have the same C. This will lead to significant reduction in the number of nonredundant candidates. If the buffer cost is the number of buffers, there are at most n_k nonredundant candidates at the root of tree T_k. For more general buffer cost, the reduction at the root of each tree is still significant.

Then, the set of nonredundant candidates are propagated to tree T_{k-1}. Now for the tree T_{k-1}, a single candidate $(Q, C, 0)$ is assigned for each sink in S_{k-1} except for r_k, where a set of candidates with different (Q, C, W) are given. Applying (Q, C, W) framework again for T_{k-1}, we can get a set of solutions at the r_{k-1}. Repeat the process for trees $T_{k-2}, \ldots, T_1$. At the end of the algorithm, a set of solutions with different cost-RAT tradeoff is obtained for the primary input. Each solution gives the maximum RAT achieved under the corresponding cost bound.

Since for each routing tree, the number of solutions associated with each sink is bounded by $|W|$, with the similar analysis of Lillis *et al.* [3], we can get a pseudo-polynomial time algorithm.

C. Re-assigning Timing Requirements

After we find a buffer assignment α for P^*, we remove gates and interconnect in P^* from G to partition G into one or more subgraphs. However, since each gate and interconnect of P^* may have connection with other parts of the circuit, we need to adjust the circuit. Fig. 2 shows the relation between P^* and the rest of the circuit.

For the case in Fig. 2(a), the gate in P^* has a side input u. We will make node u a primary output for the remaining circuit and let $RAT(u) = AT(v, \alpha) - K(u, v)$. For the case in Fig. 2(b), the net in P^* has a branch leading away from P^*. We will make node u, which is the first buffer/gate assigned by α downstream from P^*, a primary input for the remaining circuit and let $DT(u) = AT(u, \alpha)$.

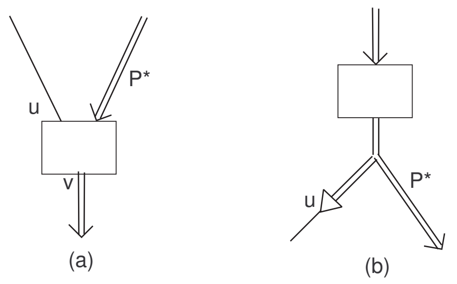

Fig. 2. (a) Creating Primary output. (b) Creating Primary input.

VI. Experimental results

Table I presents the experimental results for the new algorithm on ISCAS benchmark circuits. The wire lengths are obtained from the actual layout and scaled by a constant factor to create requirement for buffering. For simplicity, buffer cost is taken as number of buffers. The complexity of the circuit is indicated by the number of nodes and edges in the resulting DAG and the number of possible buffer locations.

The buffer cost obtained by our algorithm is compared against a net based approach. As seen from Table I, the reduction in the buffer cost for same delay requirements is about 4 to 18 times. The running time for larger circuit is not necessarily higher than a smaller circuit since the number of partitioning cycles required for the circuit depends on the circuit structure rather than the circuit size.

VII. Conclusion

In this paper, we propose a buffer insertion algorithm that inserts buffers into combinational circuits so that the timing requirements are met and buffer cost is minimum. Unlike previous net-based buffer insertion algorithms, our algorithm works with the entire circuit, and therefore uses significantly fewer buffers. Experimental results on ISCAS85 circuits show that our algorithm can reduce the number of buffers by 75% to 95%, compared with the traditional net-based algorithms. The running time of our algorithm is reasonable. Our algorithm can be combined with gate sizing, since both van Ginneken style (Q, C) algorithms and Lillis style (Q, C, W) style algorithms work for gate sizing, by specifying gate position as a buffer position, at the cost of higher running time.

References

[1] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "Repeater scaling and its impact on CAD," *IEEE Trans. on CAD*, 23(4):451463,April 2004.

[2] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree network for minimal Elmore delay," *Proc. 1990 ISCAS*, 865–868.

[3] 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(3), 1996, 437–447.

[4] I-Min Liu, A. Aziz, D.F. Wong, Hai Zhou, "An efficient buffer insertion algorithm for large networks based on Lagrangian relaxation," *ICCD 1999*, 210–215.

[5] I.-M. Liu, A. Aziz, and D. F. Wong, "Meeting delay constraints in DSM by minimal repeater insertion," *Proc. of DATE*, 436–441, 2000.

[6] C. J. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In *DAC*, 1997.

[7] C. C. N. Chu and D. F. Wong. A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):787–798, 1999.

[8] W. Shi and Z. Li, "An $O(n \log n)$ time algorithm for optimal buffer insertion," *Proc. 2003 DAC*, 580–585.

[9] W. Shi, Zhuo Li, C.J. Alpert, "Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost," *Proc. 2004 ASPDAC*, 609–614.

[10] S. Lin and M. Marek-Sadowska. A fast and efficient algorithm for determining fanout tree in large networks. In *EDAC*, pages 539–544, 1991.

TABLE I

SIMULATION RESULTS FOR ISCAS BENCHMARKS.

	Circuit Size			Cost			
Circuit	Nodes	Edges	Locations	Net based	New Algorithm	% Reduction	CPU/s
c432	687	826	868	143	37	74	7.13
c499	881	1045	1216	215	43	80	69.14
c880	1510	1795	1632	266	29	89	14.94
c1355	2193	2669	1868	348	86	75	342.99
c1908	3047	3631	4037	773	95	88	494.26
c2670	4585	5234	7328	1767	111	94	173.11
c3540	5920	7139	7729	1621	102	94	670.43
c5315	9018	10918	11403	2475	150	94	491.21
c7552	12505	14929	16758	3414	191	94	455.53

[11] H. Zhou, D. F. Wong, I. M. Liu, and A. Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. *IEEE Trans. CAD*, 19(7):819–824, 2000.