UC Santa Barbara

UC Santa Barbara Previously Published Works

Title

Timing driven gate duplication

Permalink

https://escholarship.org/uc/item/83v774jp

Journal

IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, 12(1)

ISSN

1063-8210

Authors

Srivastava, A
Kastner, Ryan
Chen, C H
et al.

Publication Date

2004

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Timing Driven Gate Duplication

Ankur Srivastava, *Member, IEEE*, Ryan Kastner, Chunhong Chen, and Majid Sarrafzadeh, *Fellow, IEEE*

***Abstract*—In the past few years, gate duplication has been studied as a strategy for cutset minimization in partitioning problems. This paper addresses the problem of delay optimization by gate duplication. We present an algorithm to solve the gate duplication problem. It traverses the network from primary outputs(PO) to primary inputs(PI) in topologically sorted order evaluating tuples at the input pins of gates. The tuple's first component corresponds to the input pin required time if that gate is not duplicated. The second component corresponds to the input pin required time if that gate were duplicated. After tuple evaluation the algorithm traverses the network from PI to PO in topologically sorted order, deciding the gates to be duplicated. The last and final traversal is again from PO to PI, in which the gates are physically duplicated. Our algorithm uses the dynamic programming structure. We report delay improvements over other optimization methodologies. Gate duplication, along with other optimization strategies, can be used for meeting the stringent delay constraints in today's ultra complex designs.**

***Index Terms*—Delay optimization, gate duplication, logic synthesis.**

I. Introduction

LOGIC synthesis is the process of transforming a set of boolean equations into a circuit comprising of gates that implement the logic while minimizing some cost function (usually area or delay). Logic synthesis is decomposed into two phases: *technology independent and technology dependent*. The main objective of the technology independent phase is to simplify the logic. At this stage, we do not have an accurate estimate of the circuit parameters, hence, optimization techniques in this phase use prediction metrics like *number of literals*. The goal of technology dependent phase is the implementation of logic in well characterized logic gates from a given technology library [28]. There are many optimization objectives of logic synthesis (eg. area, delay, and power) but the present work deals with delay optimization.

Many timing optimization strategies have been proposed over the past few years. Some of the *rule based* techniques are LSS [13] and SOCRATES [15]. These use predefined set of local transformations based on design style and technology to improve delay. Some of the popular strategies that use algorithms have been suggested in [17] and [23], which exploit the concept of restructuring for improvement in circuit performance. The choice of nodes on which restructuring is applied depends on the maximum delay improvement achievable with minimum area penalty. [16] is another strategy which uses permissible functions and network "don't cares" to optimize the circuit. [1] reduces the delay of the longest sensitizable path in the circuit. Speed up is another topology based optimizer and relies on a number of local transformations which includes tree height reduction [17]. These transformations are local and attempt to resynthesize a small part of the network. The best timing improvement of all the nodes is estimated and a set of nodes is selected on which the transformations should be applied.

Other strategies for timing optimization are applicable during or after technology mapping. These include gate sizing and buffer insertion. Gate sizing is the process of deciding the driving strength of a gate. A high-driving strength corresponds to smaller delay but larger area. [25] and [26] address the issue of gate sizing for performance optimization. Buffer insertion is a strategy that decreases capacitive loading at critical gates such that the delay is reduced. [19], [24] and [30] address the problem of buffer insertion.

New developments in logic synthesis strive to combine synthesis with physical design. Some of these strategies depend on prelayout estimation of the effects of the layout without actually performing it, and using these estimates to guide the optimization steps. [22] suggests a methodology which has a fast global placement guiding the wiring estimate and technology mapping. The approach was later extended in [21] to include logic restructuring and technology decomposition and further extended in [11] to include post mapping fanout optimization. These methodologies predict or generate a layout and base their logic optimizations on this information. The final layout might be entirely different causing inaccuracy in the estimates. Another set of methodologies try to do physical design and logic optimization simultaneously; [14] and [31] are attempts of combining technology mapping and placement. [2] combines floorplanning, technology mapping and gate placement; [3], and [29] combine fanout optimization and routing tree construction. [27] is another layout driven logic optimization strategy which incrementally does placement and logic resynthesis.

In this paper, we present a new algorithm for timing driven gate duplication after technology mapping. As we show in Section II, gate duplication is the process of duplicating a gate which has large number of fanouts. In this process, the number of fanouts of each gate (the original and the replica) decreases hence reducing the individual gate delays. This can potentially reduce the overall circuit delay. A preliminary version of this paper appeared in ICCAD-2000 [8]. The research community has looked at gate duplication extensively as a method of reducing the cutset of partitions. Strategies of logic (gate)

Manuscript received November 9, 2001; revised May 19, 2003.

A. Srivastava is with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742 USA (e-mail: ankurs@glue.umd.edu).

R. Kastner is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA (e-mail: kastner@ece.ucsb.edu).

C. Chen is with the Department of Electrical and Computer Engineering, University of Windsor, ON N9B, 3P4 Canada (e-mail: cchen@uwindsor.ca).

M. Sarrafzadeh is with the Department of Computer Science University of California Los Angeles, Los Angeles, CA 90001 USA (e-mail: majid@cs.ucla.edu).

Digital Object Identifier 10.1109/TVLSI.2003.820527

1063-8210/04$20.00 © 2004 IEEE

duplication for cutset minimization were suggested in [18] and [20]. The strength of gate duplication as a cutset minimizing strategy has been demonstrated. However, applicability of this strategy in reducing the circuit delay has not been studied in detail. One of the few works is [12], which addresses the gate duplication problem in a performance driven perspective. It integrates cell replication into a layout driven framework. Another strategy of gate duplication that addresses this problem in the technology independent phase has been proposed in [9] and extended in [4].

Both the global (pertaining to the whole circuit) and local delay optimization problems (in which we are interested only in a gat and its immediate fanouts) by gate duplication are NP-complete [7]. We present an algorithm for gate duplication which is based on the dynamic programming approach. It traverses the network from primary outputs (PO) to primary inputs (PI) in topologically sorted order evaluating tuples at the input pins of gates. The tuple's first component corresponds to the input pin required time if that gate is not duplicated. The second component corresponds to the input pin required time if that gate were duplicated. After tuple evaluation the algorithm traverses the network from PI to PO in topologically sorted order, deciding the gates to be duplicated. The last and final traversal is again from PO to PI in which the gates are physically duplicated. Our algorithm uses the dynamic programming structure.

The rest of this paper is organized as follows. Section II deals with the delay model and provides basic definitions. Section III reviews the complexity of the global gate duplication problem. Section IV presents the routine for local optimization by gate duplication. This routine will be repeatedly called by the global algorithm. Section V describes the global algorithm, followed by a heuristic for constraining the area penalty in Section VI. Results are reported in Section VII. This is followed by some observations and conclusion in Section VIII.

II. Preliminaries

A. Delay Models

Given a single output gate g, let $\delta(i,g)$ denote delay from an input pin i of the gate g to the output of g. The load C_g denotes the cumulative capacitance seen at the output of g. It is the sum of the individual input pin capacitances γ_p for all fanouts p of g. A commonly used delay model for gate level circuits is the load dependent delay model **LDDM** [30] according to which the delay in the gate g is given by

$$\delta(i,g) = \alpha_{i,g} + \beta_{i,g} C_g. \tag{1}$$

Here

- C_g load capacitance at the output of the gate g;
- $\alpha_{i,g}$ intrinsic delay from pin i to output of g;
- $\beta_{i,g}$ drive capability or load coefficient of the path from i to the output of g.

The delay of a path that goes from a primary input (PI) to a primary output (PO) is the sum of the pin-to-pin delays through all the gates lying on the path [24]. The delay in the circuit is the maximum of all the individual path delays. Let $r(g)$ denote the required time at the output of a gate g. The following equation illustrates the method of computing $r(g)$ if the required times of the fanouts of g are available

$$r(g) = \min_{x \in FO(g)} \{r(x) - \alpha_{g,x} - \beta_{g,x} C_x\}. \tag{2}$$

Here

- $\alpha_{g,x}$ intrinsic delay of gate x with respect to the pin connected to g,
- $\beta_{g,x}$ load coefficient of gate x with respect to the pin connected to g

Required time at the input pin of a gate is define as follows:

$$r(i,g) = r(g) - \delta(i,g). \tag{3}$$

In this paper, we neglect the wire capacitance. We also assume all gates in the circuit to be single output combinational gates. In this paper we set the required times at the POs to zero and the arrival times (time at which the signal arrives) at PIs to zero. Hence, the slack at all the gate is always negative. In this scenario, the objective of our delay optimization algorithm is the maximization of the minimum slack in the circuit.

B. Gate Duplication Problem

Gate duplication can be used for delay optimization. The idea is illustrated with the following example.

Consider the circuit shown in Fig. 1(a) *in which the parameters* α, β, and γ *along with the required times have been indicated. The subscript* i *has been omitted from gate parameters as all the gates have just one input pin.. This circuit structure consists of only buffers. Let us assume that these are the only type buffers in the design library. Hence, gate sizing is in-effective. Since this is a balanced circuit structure, buffer insertion does not improve the delay either. We will show that the delay through this circuit can be improved by duplicating some gates In the unduplicated case,* Fig. 1(a) *the capacitive loading* $C_D = 1+1+1+1$ *and* $C_E = 1$. *Hence, the required time at the input of* E *can be calculated to be* -5. *When* D *is duplicated* Fig. 1(b)*, the capacitive loading* $C_D = 1+1$ *and* C_E =2. *Hence the new required time at the input of* E *becomes* -4. *Gate duplication was hence, instrumental in improvement of circuit delay.*

III. Complexity Issues of Gate Duplication

In this section, we will briefly outline the complexity issues associated with gate duplication. Both the global and local problems have been proved NP-complete [5]. The global problem is concerned with the delay optimization of the entire network through gate duplication. It can be defined more formally as follows:

1) given a network η consisting of gates and nets;
2) given the delay parameters $\alpha_{i,g}$ $\beta_{i,g}$ and γ_i for each gate g where i is the ith input pin of g;.
3) find a duplication strategy that maximizes the minimum required time at the PIs (assuming the required time at POs is zero).

MONO3SAT problem was transformed to an instance of global gate duplication problem [5] and hence, proved NP-complete in **LDDM**; [5] shows that the problem of partitioning a set

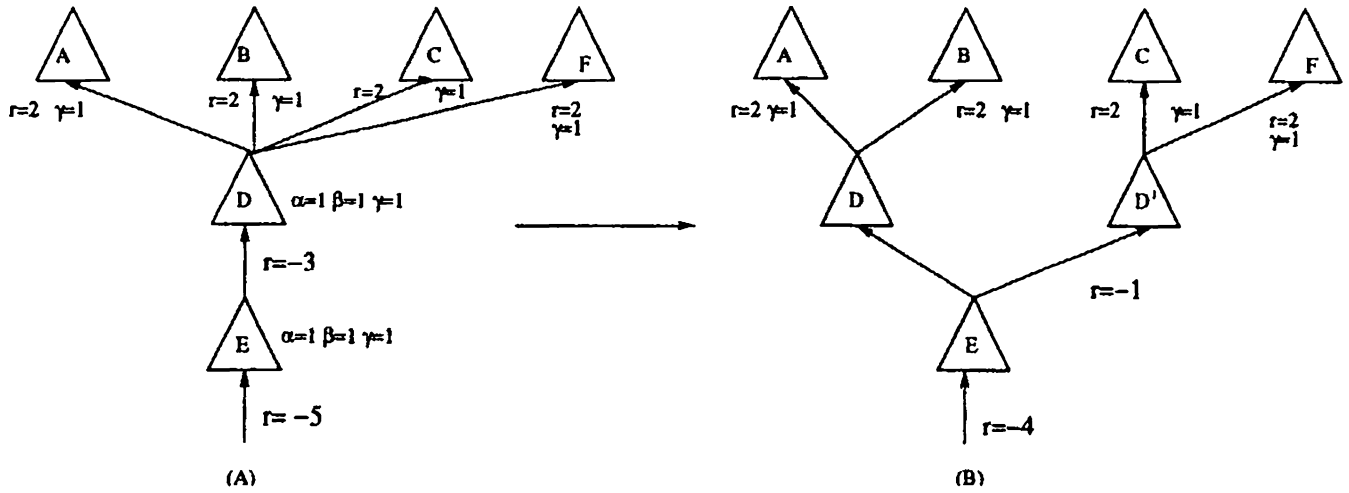

Fig. 1. Delay optimization by gate duplication.

of fanouts between a node and its replica for maximizing input pin required time is also NP-complete. This makes the local problem NP-complete. For a more formal definition and proofs please refer to [5]. In this paper, we are primarily interested in algorithms for solving the problem. Any gate duplication algorithm will have to make at least two decisions on which the final result will depend:

1) decide the gates to be duplicated;
2) decide the fanouts of the duplicated gates

In Sections IV–VII, we will describe an algorithm to solve the global gate duplication problem. Both local and global aspects of the algorithm will be discussed in detail.

IV. Delay Optimization in the Local Scenario

In this section, we will discuss the local formulation of the problem of gate duplication. This formulation is used in the global algorithm to optimize the overall delay of the circuit. The local problem can be defined as [8].

1) Given a node n and a set of fanouts
2) Two numbers are associated with each fanout, one corresponds to the required time at the input pin if that fanout gate is duplicated and the other corresponds the the required time if it is not duplicated
3) Find the highest required time at the input pin of gate n if n is duplicated and if n is not duplicated

Before we proceed any further, let us define a data structure which we call tuple. A tuple is represented as $((r1),(r1,r2)')$. It has two subcomponents $(r1)$ and $(r1,r2)'$. The values $r1, r2$ are of floating data type. Let us define the following functions on tuple

$$\text{NODUP}(((r1),(r1,r2)')) = (r1) \tag{4}$$

$$\text{DUP}(((r1),(r1,r2)')) = (r1,r2)'. \tag{5}$$

Functions DUP and NODUP let us access the internal fields of a tuple. Hence, $\text{DUP}((((r1),(r1,r2)')).r1$ (or r2) represents the r1 (or r2) component of $(r1,r2)'$. Similarly, $\text{NODUP}(((r1),(r1,r2)')).r1$ gives the value stored in the first subcomponent of tuple. This data structure will be used extensively in this paper to explain the algorithmic concepts. Now let us come back to the problem of local optimization. Please refer to Fig. 2 for a graphical explanation of the points enumerated above. Fig. 2(a) illustrates the existence of a tuple data structure (i,n) associated with each input pin i of each gate n. The pair (i,n) just illustrates the fact that we are referring to the tuple associated with the ith input pin of gate n. As mentioned before, each tuple has two components. The first component $\text{NODUP}(i,n).r1$ is the best (largest) required time at the input pin i if gate n is not duplicated Fig. 2(b). This could be achieved by duplicating some fanouts and not duplicating others. Finding the best duplication methodology for the fanouts such that $\text{NODUP}(i,n).r1$ gets maximized is the objective here. The second component $\text{DUP}(i,n)$ is associated with duplication of the node. If the node n is duplicated into n and n' [see Fig. 2(c)], then the two values $\text{DUP}(i,n).r1$ and $\text{DUP}(i,n).r2$ correspond to required times at the ith input of n and n' as shown in Fig. 2(c). Here, $r1$ component is smaller of the two and $r2$ is larger of the two. While computing this value we assume that when gate n gets duplicated, both the original and the replica get connected to the same fanin. The objective is the maximization of $\text{DUP}(i,n).r1$. There are two problems that need to be solved in order to achieve the desired values. The first is identification of the best fanout duplication methodology and the second problem is the partitioning of these fanouts between the original gate and the replica. Algorithms to solve these problems will be discussed in Sections IV-A and B. There might be cases in which a gate cannot be duplicated, (for instance if it has only single fanout). In such a situation $\text{DUP}(i,n)$ is set to NULL. In our formulation, the local problem corresponds to the following

Given the tuples (k,f) for all the fanouts f of a gate n. Here k denotes the input pin at which the fanout f is connected to n. Compute the tuples (i,n) for all input pins i of gate n. Value $\text{NODUP}(i,n).r1$ *is the largest required time at the ith input pin if gate n is not duplicated. This can be achieved by choosing the appropriate fanout duplication methodology. Values* $\text{DUP}(i,n).r1$ *and* $\text{DUP}(i,n).r2$ *corresponds to the input pin required times at gates n and n' which is the replica of n. The*

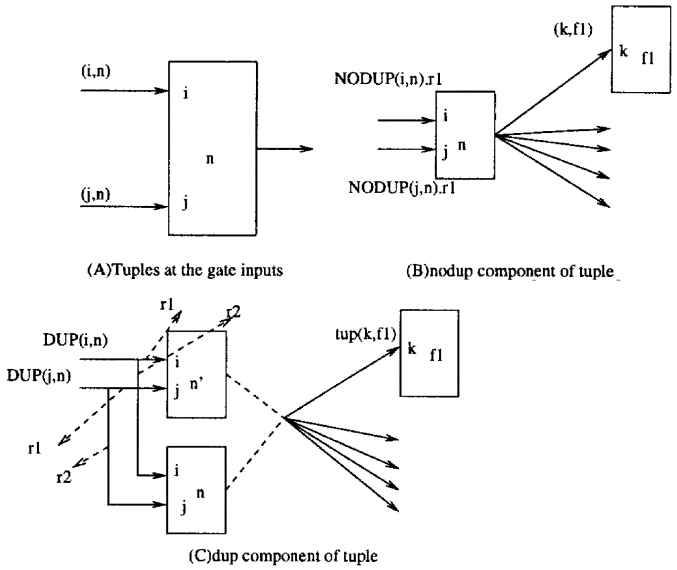

Fig. 2. Local optimization.

Algorithm 1: $EVAL_{NODUP(i,n)}$

n - node under consideration
i - input pin under consideration
A = Sort the fanouts f in increasing order of their $NODUP(k,f).r1$ entries
$/*(k,f)$ is the tuple of fanout gate f at input pin connected to n $*/$
maximum = Required time at pin i of gate n
maxindex = 0
for p=1; p< $numberoffanoutsofn$; p++
    duplicate the first p fanout gates in A
    recompute the capacitive loading for gate n
    compute the required time R at the input pin i of gate n
    if $R \geq maximum$
        maximum = R
        maxindex = p
$NODUP(i,n).r1 = maximum$
/* maxindex contains the information on fanout script /*
/* The first maxindex terms in the original sorted array should be duplicated */
end

Fig. 3. NODUP$(i,n).r1$ evaluation.

objective is the maximization of DUP$(i,n).r1$ *which is smaller of the two values.*

Before we proceed any further, let us define the term *fanout script*

Fanout script associated with an input pin of a node is a vector of boolean variables. Each boolean variable corresponds to a fanout. If the boolean variable is TRUE *then that fanout is designated for duplication else it is not duplicated.*

A. Computation of NODUP(i,n)

While computing NODUP(i,n), the assumption is that the gate n cannot be duplicated. We need to find the best duplication methodology of the fanouts (fanout script) such that the required time at the input pin i of gate n is maximized. As mentioned before, tuples (k,f) for all the fanouts f of n are known. Again the index k indicates the input pin of fanout f at which n gets connected. Hence, we know the required times at the input pins of the fanouts if they are set for duplication and if they are not set for duplication. Using these values we need to find the best duplication methodology of the fanouts such that the input pin required time of gate n is maximized with respect to the input pin i. This duplication methodology will correspond to the designated fanout script for n at pin i.

Fig. 3 has a formal description of the algorithm used for the computation of NODUP$(i,n).r1$. Basically it sorts the fanouts f in increasing order by their NODUP$(k,f).r1$ value. Then it duplicates the first fanout in this list and keeps the rest unduplicated and computes the required time at the ith input pin of n. Then it duplicates the first two fanouts and recomputes the required time. This is followed by the duplication of first three and so on. The one that gives the largest input pin required time is picked. This duplication methodology is the designated fanout script. Now let us first prove the following lemma

Lemma 1: Given a gate n and an input pin i of n, then NODUP$(i,n).r1 \leq$ DUP$(i,n).r1 \leq$ DUP$(i,n).r2$.

Proof: By definition DUP$(i,n).r1 \leq$ DUP$(i,n).r2$. Duplication will always lead to a reduction in the number of fanouts of a gate. Hence, the capacitive loading of that gate will be reduced (of course there will be an increase in the capacitive loading of the fanin gate). This reduction will cause a reduction in the gate delay. Now, consider NODUP(i,n) and the corresponding fanout script. Now duplicate the gate assuming the fanouts are the ones defined by this fanout script. Partition these fanouts between the gate and its replica and compute the input pin required time R. Any partitioning will work. Since the gate delay has reduced, this value will be larger than NODUP$(i,n).r1$. By definition, DUP$(i,n).r1$ is the largest input pin required time if the gate n is duplicated. This makes $R \leq$ DUP$(i,n).r1$. Hence, we can conclude

$$\text{NODUP}(i,n).r1 \leq \text{DUP}(i,n).r1 \leq \text{DUP}(i,n).r2.$$

□

Theorem 1: Given a gate n and its input pin i, a set of fanouts which need to be driven and the tuples associated with the fanouts. Algorithm $EVAL_{NODUP(i,n)}$ gives the optimal value of NODUP$(i,n).r1$. Here, optimality is defined as the largest possible value of NODUP$(i,n).r1$.

Proof: According to Lemma 1, whenever a gate is duplicated the required time at its input pin always improves. Let A be the sorted array of fanouts and let us associate a boolean variable x_i with the ith entry in A. The algorithm only enumerates fanout scripts in which a string of TRUEs is followed by a string of FALSE in the sorted array A. If the algorithm indicates that the first L fanouts should be duplicated (ie maxindex $= L$), then the variables x_0 to x_{L-1} are set to TRUE in the fanout script(ie the corresponding fanouts should be duplicated) and rest are set to FALSE. We show that the fanout script of the optimal solution should always consist of a string of TRUEs followed by a string of FALSE in the sorted array A. Let us assume that such a solution is not an optimal solution. Let us assume an optimal solution with a boolean variable x_j corresponding to fanout j. Variable x_j is chosen such that it is the first boolean variable that is set to TRUE after a string of FALSE (note that algorithm $EVAL_{NODUP(i,n)}$ will not enumerate such fanout duplication methodologies). Since NODUP$(k,j).r1 \geq$ NODUP$(m,y).r1$

for all $y < j(m)$ is the input pin index with which fanout y gets connected to node n) in the sorted array, the required time $r(n)$ at the output of n will never be dictated by fanout j. This is because some of more critical fanouts (that appear before j in array A) have not been duplicated. Hence, by Lemma 1 duplication of gate j does not help the required time $r(n)$ at all. But duplication of gate j does add some extra capacitive loading on n, hence, increasing its circuit delay. If we decide not to duplicate this gate, the required time $r(n)$ will not be affected but the gate delay $\delta(i,n)$ will improve as the extra capacitive loading will be removed. Hence, such a solution cannot be optimal which is a contradiction. So an optimal solution must always have a fanout script with a string of TRUEs followed by a string of FALSE in the sorted array. Basically duplicating a less critical fanout will not give any improvement if a more critical fanout has not been duplicated. The algorithm $EVAL_{NODUP(i,n)}$ enumerates all possible cases in which there is a string of TRUEs followed by a string of FALSE (there are $F+1$ of them, where F is the number of fanouts of n) and picks up the best for that particular input pin. Hence, it is optimal. □

This completes the discussion on the computation of NODUP(i,n). The next step is the computation of DUP(i,n).

B. Computation of DUP(i,n)

In this case the node n is in the duplicated state. As shown in Fig. 2(c) node n and n' have the same fanins. The objective is the computation of DUP(i,n) such that DUP$(i,n).r1$ is maximized (by definition DUP$(i,n).r1 \leq$ DUP$(i,n).r2$). As mentioned before, DUP$(i,n).r1$ and DUP$(i,n).r2$ correspond to the required time at the input pin i of gate n, if n is in duplicated state. Two things need to be done in order to achieve this

1) First evaluate the appropriate fanout duplication methodology (fanout script)
2) Since node n is in duplicated state, some of the fanouts of n will be driven by n'. Hence, the second problem is the partitioning of fanouts between the node n and n'. Note that as the fanout script changes, the fanouts also change. This is because some gates might get duplicated (the ones to which the boolean value TRUE has been assigned), hence changing the fanouts

[6] shows that the problem for partitioning a set of fanouts between a gate and its replica in order to maximize the input pin required time is NP-complete. Hence, we use heuristics to obtain a good value for DUP(i,n).

The objective is the maximization of DUP$(i,n).r1$. The algorithm tries to pick the appropriate fanout duplication methodology and partitioning strategy which achieves this. We enumerate the same fanout duplication methodologies (fanout scripts) that are considerd by $EVAL_{NODUP(i,n)}$ (ie the ones in which a string of TRUEs is followed by a string of FALSE in the sorted array of fanouts). This is a heuristic decision (since in the worst case we can have an exponential number of fanout scripts). For each fanout script we evaluate the fanouts that need to be driven. The fanout set will change with the fanout script. For each fanout script, the algorithm partitions the fanouts between the node and its replica such that the input pin required time is maximized. The fanout script that gives the largest input pin required time is chosen. The algorithm that partitions the fanout set is as follows.

Algorithm 2: EVAL_PARTITION

```
n - node under consideration
i - input pin under consideration
A - Array of fanouts and their required times at input pins
F - Cardinality of A
Sort the array A in increasing order
If two fanouts have same required time then insert the fanout with higher
input pin capacitance first in the sorted list
/*Let the ith entry A correspond to the ith fanout */
r1 = -INFINITY
r2 = -INFINITY
for p=1; i< F; p++
      Fanouts from index 1 to p will be driven by n
      Fanouts from index p+1 to F will be driven by n'
      Compute the capacitive loading for n and n'
      req(n) = required time at the input pin i of n
      req(n') = required time at the input pin i of n'
      if (r1 ≤ min(req(n), req(n')))

            r1 = min (req(n), req(n'))
            r2 = max (req(n), req(n'))
            partindex = p

partindex - contains the partitioning location in the sorted list
end
```

Fig. 4. Partitioning the fanout set into two.

For the current fanout script generate the fanout set that needs to be driven and populate an array A with this fanout set. If a fanout f is duplicated then insert DUP$(k,f).r1$ and DUP$(k,f).r2$ in A else insert NODUP$(k,f).r1$. These correspond to the required times at the input pins of all the fanouts that need to be driven. The next step is the partitioning of this fanout set and computation of $r1$ and $r2$ (which correspond to the required time at the input pin of the node n and its replica). Sort the array A in increasing order. Partition this sorted array by simply cutting the array into two. There can be $(F-1)$ such partitions where F is the number of entries in the array A. One of the partition will be driven by the original gate and the other by the replica. Evaluate the required time at the input pins of the original gate and its replica for the current partitioning of fanouts. Assign the lower of the two required times to $r1$ and higher to $r2$. Repeat this for all the $(F-1)$ partitions and pick the partitioning which maximizes $r1$. This procedure is formally illustrated in Fig. 4. It chooses the partition for which $r1$ is maximized.

Algorithm $EVAL\_PARTITION$ is called repeatedly for the different fanout duplication methodologies (fanout scripts). The script that gives the highest value input pin required time is the chosen one.

Theorem 2: Consider an array of fanouts, the required time at their input pins, a node n and its duplicate n' (with common fanins). $EVAL\_PARTITION$ generates an optimal solution if the input pin capacitances for all the fanouts are the same. If r_n is the required time at the input pin of gate n and r'_n is the input pin required time of n'. Optimal solution is the one which has the maximum value of $\min(r_{n'}, r_n)$.

Proof: The algorithms works on the sorted list of input pin required times at the fanouts. The output is a partition with the property that all the fanouts in one partition say n have input pin required times $\geq$ the ones in partition n'. Let us call this property

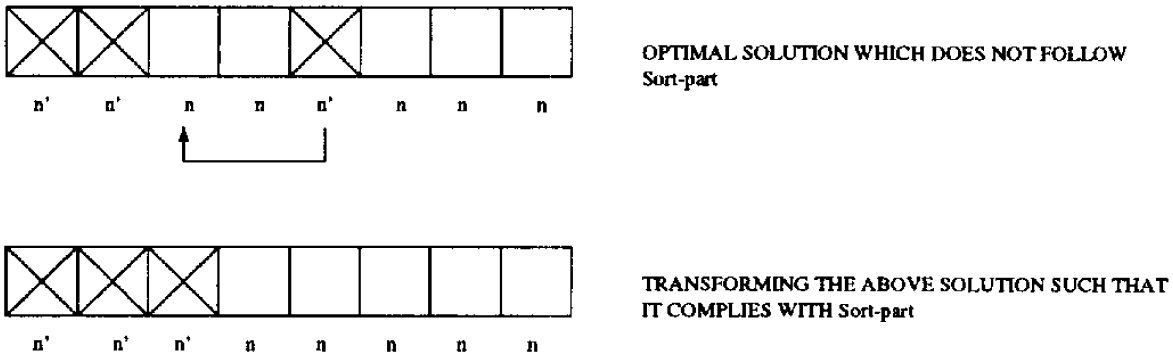

Fig. 5. Transforming an optimal solution to comply with sort-part.

sort-part. $EVAL\_PARTITION$ enumerates all the partitions that comply with property *sort-part* and picks the best (ie the one that maximizes $\min(r'_n, r_n)$). We prove that the optimal solution always complies with property *sort-part*.

Consider an optimal solution which does not comply with sort-part. Such a solution is illustrated in Fig. 5. It shows an array of fanouts sorted in increasing order by their input pin required times. Partition n' has three entries and partition n has five entries. Partition n' comprises of the gates that will be driven by gate n' and partition n will be driven by gate n. The required time of n' will be decided by the first entry in the array, while the third entry will decide the required time for n. Move the third entry to partition n' and move the fifth entry to partition n. There is no change in the number of fanouts of n and n'. There is no change in the required time of n'. But there is a potential increase in the required time of n. If this increases the input pin required time then the initial solution was not optimal which is a contradiction. If this transformation does not affect the solution then the transformed solution is also optimal although it complies with sort-part. This transformation can never make the solution worse. Hence, at least one optimal solution should comply with sort-part. □

This completes the discussion on the computation of tuples. Now let us illustrate the computation of the tuples using an example. Consider the circuit shown in Fig. 6. The tuples for the fanout gates A, B, and C are known. The gate parameters are also known. The problem is to find the tuple of gate D using this information. In this example, all gates have exactly one input pin. Hence, the input pin index has been dropped from the tuple information.

Let us first illustrate the computation of NODUP(D). Note, that since D has exactly one input pin, the input pin index term has been dropped.

After sorting the fanouts by their nodup values we get the ordering A, B, C. Following are the fanout scripts that $EVAL_{NODUP(i,n)}$ considers

1) $A =$ FALSE, $B =$ FALSE, $C = FALSE$
2) $A =$ TRUE, $B =$ FALSE, $C =$ FALSE
3) $A =$ TRUE, $B =$ TRUE, $C =$ FALSE. Since B cannot be duplicated (as DUP(B) =)NULL, this fanout script becomes $A =$ TRUE, $B =$ FALSE, $C =$ FALSE
4) $A =$ TRUE, $B =$ FALSE, $C =$ TRUE

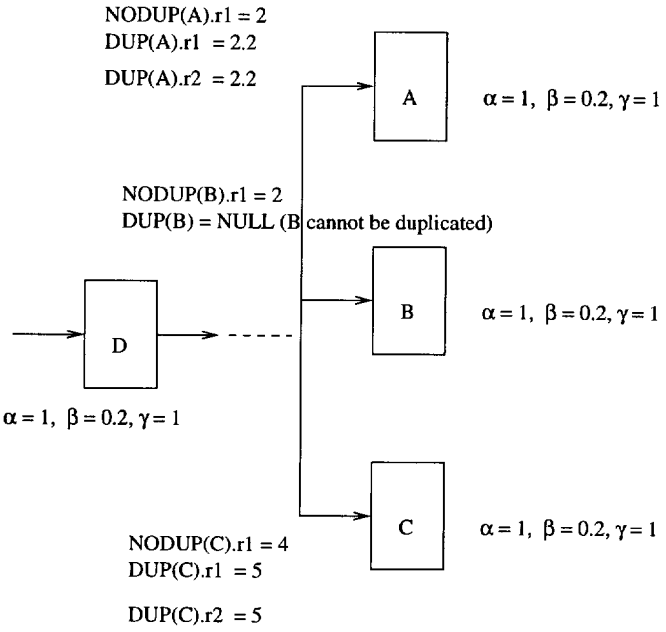

Fig. 6. An example.

Fig. 7 illustrates the computation of NODUP(D) from the three nonredundant fanout scripts. The numbers in the boxes correspond to the input pin required times of the fanout gates which D will have to drive for various fanout scripts. The optimal NODUP(D) comes from the first fanout script.

Next we illustrate the computation of DUP(D). As mentioned before, we consider only those fanouts scripts that are considered by $EVAL_{NODUP(i,n)}$. This is a heuristic decision. For each of these scripts, we try to find the best partitioning strategy such that the input pin required time is maximized. Fig. 8 shows these evaluations. For each of the fanout scripts, algorithm EVAL_PARTITION is used to compute the best partitioning of fanouts such that DUP(D).$r1$ is maximized. For this particular example this value has come out to be the same for all the fanout scripts. In this case our algorithm will pick the script with minimum area penalty, which for this case is the first fanout script.

This completes the description of the local optimization routine. In Section V, we will describe the use of this methodology for optimizing the global circuit delay.

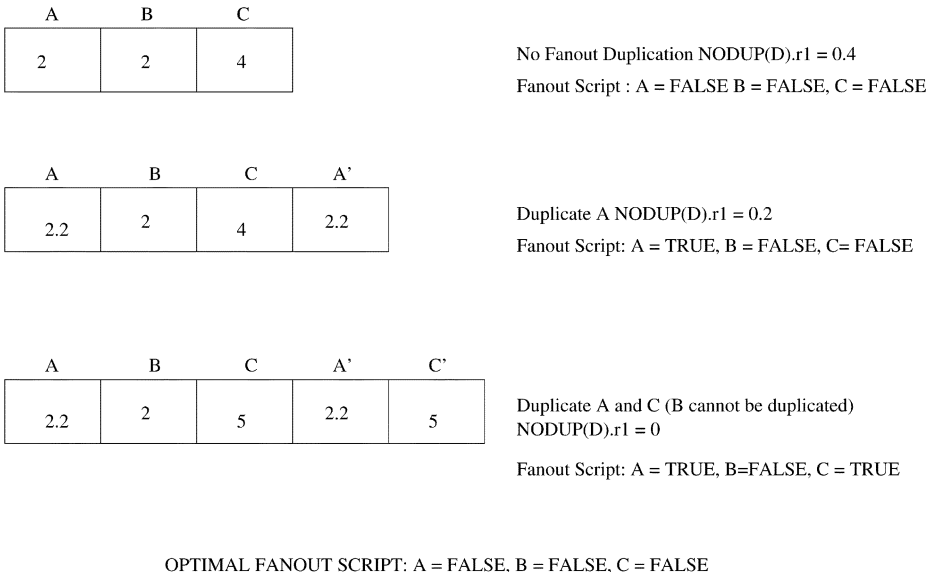

Fig. 7. Computation of NODUP(D).

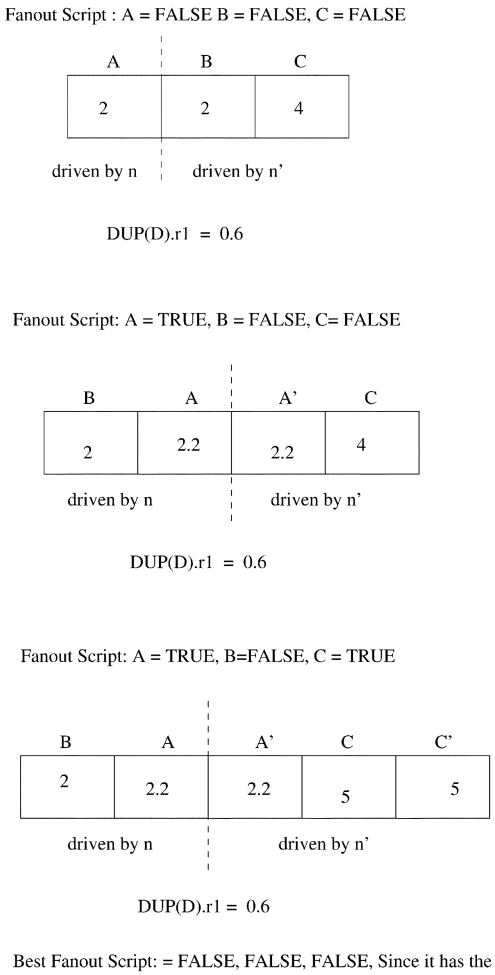

Fig. 8. Computation of DUP(D).

V. The Global Algorithm

Now we describe the global algorithm for optimization. We will call this algorithm DUP_HEU in the rest of the paper. It is based on dynamic programming and is divided into three stages.

A. Stage 1

In this stage, the network is traversed in reverse topologically sorted order from POs to PIs. This ensures that when a gate is being evaluated, all its fanouts have already been evaluated. If n is the node under consideration, this stage computes tuples (i, n) for all input pins of the node and their corresponding fanout scripts. Since the network is being topologically traversed, the tuples of all the fanouts have already been computed. Hence, using the theory and algorithms described in Section IV, the value (i, n) can be computed. After the completion of this stage, we have tuples and corresponding fanout scripts for all the input pins for all the gates in the network. Note that DUP(i, n) for PIs and POs is set to NULL. This is because they cannot be duplicated. We also set DUP(i, n) for gates with only one fanout to NULL.

B. Stage 2 and 3: Duplication Decisions

Stage 2: Now that we have computed the tuples and fanout scripts for each input pin of all the gates, we traverse the network from PI to PO in topological order. This traversal ensures that when a node is visited, all its fanins have already been visited. The purpose of stage two is to decide which gates to duplicate. For every node, we first decide whether it should be duplicated or not. Then we choose a fanout script (from those associated with all the input pins) and set the duplication preference of the fanouts using this script. If the fanout script sets a particular fanout to TRUE, (i.e., it should be duplicated) then the duplication preference of that fanout is set to TRUE. Every fanin of a node sets a particular duplication preference. If the node is type INTERNAL (ie it is neither PI nor PO), the duplication decision is set to the duplication preference of the most critical fanin. Next we have to choose the fanout script of this node (from all the fanout scripts associated with all the input pins). We choose the fanout script of the input pin that is connected to the most critical fanin. Using this script, we set the duplication preference of the fanouts. PIs and POs cannot be duplicated.

TABLE I
DELAY IMPROVEMENTS AND AREA PENALTY OVER MAP -N 1 FOR DUP_HEU AND $DUP\_EPSILON$

Bench	orig ar	orig delay	area inc	delay dec	area inc	delay dec
			DUP_HEU		$DUP\_EPSILON$	
c880	413888	34.87	29.5%	26.0%	18.49%	25.75%
c7552	2373360	48.75	50%	42.3%	7.19%	33.85%
apex6	883456	18.73	20.2%	14.5%	2.25%	15.91%
c8	144768	9.01	14.74%	17.63%	8.65%	17.87%
cc	72848	6.28	11.46%	21.17%	7.64%	21.17%
cht	166112	6.74	14.8%	19.4%	2.23%	13.7%
cm138a	33408	3.83	16.67%	10.9%	13.88%	10.90%
cm150a	59856	7.20	6.2%	6.7%	2.33%	8.47%
count	152192	22.05	18.6%	34.6%	13.7%	34.60%
cu	70992	6.16	16.5%	10.1%	5.2%	10.22%
dalu	1485728	51.26	28.4%	22.5%	9.02%	21.77%
des	3988080	80.95	35.2%	51.28%	8.37%	50.12%
frg1	147088	8.57	16.72%	12.4%	13.56%	14.35%
frg2	1024512	25.34	19.8%	38.8%	4.48%	16.10%
i10	2787712	56.78	39.1%	30.0%	12.9%	23.65%
i7	943776	35.49	7.85%	43.87%	1.0%	54.35%
my_adder	196272	26.08	46.8%	32.6%	20.8%	32.59%
pair	1783152	30.96	29.8%	30.11%	9.15%	25.35%
pcle	74240	9.08	21.3%	23.2%	11.875%	17.07%
pcler8	116000	11.249	13.6%	34.5%	8.4%	34.53%
tcon	45472	4.34	18.4%	11.9%	2.0%	11.98%
ttt2	242672	16.32	25.6%	25.1%	10.33%	24.81%
too_large	391616	14.12	23.9%	22.4%	11.37%	21.53%
x4	460288	15.26	20.9%	30.8%	3.9%	29.1%
x3	829168	13.96	24.2%	29.6%	2.18%	21.04%
average			**22.8%**	**25.7%**	**8.1%**	**23.63%**

TABLE II
DELAY IMPROVEMENTS AND AREA PENALTY OVER MAP -N 1 -AFG FOR $DUP\_EPSILON$

Bench	orig ar	orig delay(ns)	area inc	delay dec
c880	444976	26.57	20.2%	7.3%
c7552	2390528	26.84	23.6%	7.6%
apex6	839376	13.68	15.87%	6.06%
c8	150336	7.85	17.90%	10.58%
cc	71456	5.55	12.98%	7.02%
cht	178640	5.78	13.76%	5.1%
cm138a	33408	3.82	22.22%	8.90%
cm150a	58464	7.04	10.33%	6.9%
count	158688	16.06	13.74%	8.6%
cu	74240	5.34	6.25%	5.99%
dalu	1417520	36.38	17.08%	6.27%
des	3684160	18.41	8.28%	5.16%
frg1	151264	7.41	4.9%	4.18%
frg2	935424	15.47	10.01%	5.33%
i10	2616960	39.42	21.77%	6.2%
i7	768384	7.66	4.95%	6.1%
my_adder	210192	19.09	18.5%	6.49%
pair	1705200	21.08	22.3%	7.63%
pcle	77488	7.55	17.36%	11.5%
pcler8	115072	8.07	17.33%	9.54%
tcon	38512	3.21	2.4%	4.45%
ttt2	249168	12.785	8.37%	9.03%
too_large	393936	11.55	9.77%	6.40%
x4	426416	9.00	14.47%	5.22%
x3	835200	11.38	10.94%	3.42%
average			**13.8%**	**6.84%**

Stage 3: The third and last stage of the algorithm traverses the network from PO to PI, duplicating the gates whose duplication decision was TRUE. Fanouts partitioning is done using algorithm $EVAL\_PARTITION$. This completes the description of the algorithm. It must be mentioned that this is a polynomial time algorithm.

VI. HEURISTIC FOR CONSTRAINING AREA

The algorithm outlined in Section V (which we call DUP_HEU) can lead to a lot of area penalty if left unconstrained. In this section we describe an extension of DUP_HEU called $DUP\_EPSILON$. $DUP\_EPSILON$ puts a constraint on the gates that can be duplicated, hence, reducing the area penalty. The challenge is to select such gates without losing the quality of the solution. A critical gate is defined as gate which lies on the slowest PI to PO path. A noncritical gate is defined as a gate whose delay has little or no effect in the overall circuit delay. DUP_HEU tries to reduce the delay of all gates regardless of whether they are critical or not. This can be wasteful in area.

In our extension, we do not consider the duplication of the fanouts of a noncritical gate. Hence, while computing tuples (i, n) of a noncritical gate n we consider only one fanout script, the one in which none of the fanouts are duplicated. Note that we still compute $DUP(i, n)$, which indicates that we keep the flexibility of duplicating this gate. This is because, if the fanin of n is a critical gate, duplication of n might be useful.

VII. RESULTS

We integrated our algorithm in SIS [10] and obtained results for twenty five MCNC benchmarks. We used lib2.genlib as the target technology library. Our experiments showed that the algorithm DUP_HEU was giving high area penalties, hence, we believe $DUP\_EPSILON$ is better at optimizing delay under area constraints. Note that $DUP\_EPSILON$ will behave as

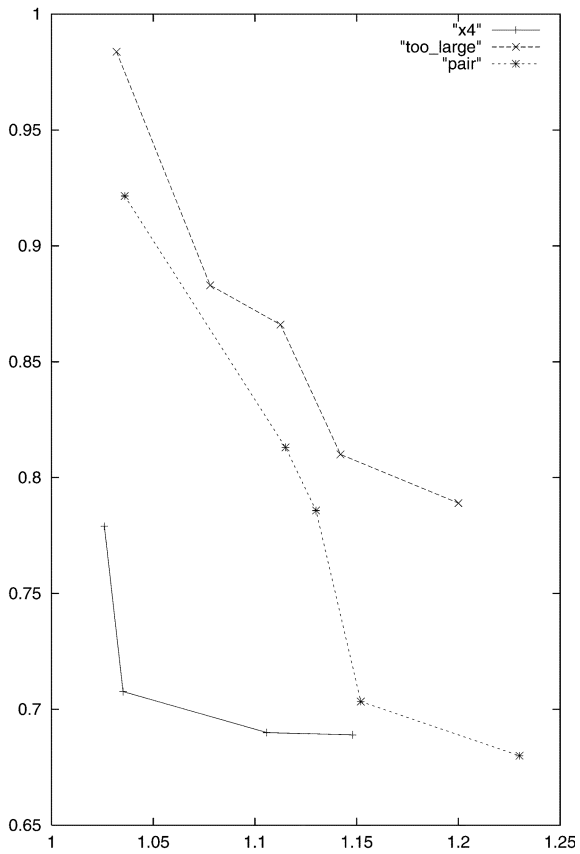

Fig. 9. Area delay curves (x axis normalized area y axis % normalized delay) for $DUP\_EPSILON$.

DUP_HEU if all gates are considered critical. For our experiments we set the required time at the POs to be zero. In such a scenario, the magnitude of the minimum slack in the circuit is also the circuit delay. A user defined parameter ϵ was used to define a critical gate. A critical gate is a gate whose slack s is $\mathrm{min\_circuit\_slack} \leq \mathrm{s} \leq (1-\epsilon)\mathrm{min\_circuit\_slack}$. All other gates are noncritical. Using this criteria for choosing a critical and noncritical gate various experiments were done with $DUP\_EPSILON$.

We obtained the results in two categories. The first category shows the effectiveness of the algorithm on a general network. The second category reports the improvements over highly optimized results generated by SIS [10]. Initially the circuit was optimized using script.rugged followed by technology mapping. For category one we used the map -n 1 option. It is a minimum delay technology mapper based on dynamic programming; for details please refer to the SIS documentation. For category two we use the map -n 1 -AFG option which is the recommended option for obtaining highly optimized networks. This option does better fanout optimization and area recovery. Again for details please refer to the SIS documentation. Note that in this category we use the best delay optimization algorithms of SIS. Mapping is followed by gate duplication. Hence, we use our algorithm as a post processing step. The results of the first category are shown in Table I. Comparisons between DUP_HEU and $DUP\_EPSILON$ can be derived. It can be seen that $DUP\_EPSILON$ can achieve large improvements at minor increase in area. DUP_HEU is not significantly better in terms of delay but incurs much larger area penalty. This illustrates the effectiveness of our algorithm. Given a general network, gate duplication has the potential of giving large improvements in delay. The ϵ value was fixed to 0.05.

The second category of results illustrate the improvements that our algorithm can give over highly optimized results generated by SIS. The circuit (after script.rugged) was mapped using map -n 1 -AFG followed by gate duplication as a post processing step. Table II shows the delay improvements and area penalty by our algorithm. Parameter ϵ was again fixed to 0.05. The improvements in this category are lesser than the previous category as we use a highly optimized mapper which reduces the fanout loading using very good fanout algorithms. Hence, it reduces the effectiveness of gate duplication. Moreover, the reported area penalty is without any area recovery. This penalty will go down if we use some strategies for area recovery (like gate sizing) after duplication.

Next the parameter ϵ was set as an input parameter and it could be varied for getting a range of area, delay values for each benchmark. Fig. 9 shows one such area/delay curve for three benchmarks, (we omit the curves for the rest of the benchmarks for clarity and brevity). Fig. 9 shows that larger delay improvements could be obtained by larger area penalty. But increasing the number of duplicated gates does not indefinetly reduce the delay.

VIII. Conclusion and Future Directions

We presented an algorithm for timing driven gate duplication in the post technology mapping phase. Its effectiveness as an efficient delay optimization methodology was reported in the SIS framework.

There can be a number of future avenues of research. One could be the merger of buffer insertion and gate duplication. Certain sections of the circuit might perform well with buffer insertion and certain might perform better with gate duplication. It would be interesting to recognize the properties of such circuits. Development of layout driven algorithms for gate duplication which address not only delay but other metrics like wirelength and congestion could be another interesting course of future work.

References

[1] A. Saldanha, H. Harkness, P. C. McGeer, R. K. Brayton, and A. L. S. Vincentelli, "Performance optimization using exact sensitization," in *Proc Design Automation Conf.*, June 1994, pp. 425–429.

[2] A. Salek, J. Lou, and M. Pedram, "A DSM design flow: Putting floorplanning, technology mapping and gate placement together," in *Proc Design Automation Conf.*, June 1998, pp. 287–290.

[3] ——, "A simultaneous routing tree construction and fanout optimization algorithm," in *Proc Int. Conf. Computer-Aided Design*, Nov. 1998, pp. 625–630.

[4] A. Srivastava, C. Chen, and M. Sarrafzadeh, "Timing driven gate duplication in technology independent phase," in Proc. Asia South Pacific Design Automation Conf., Jan. 2001, pp. 577–582.

[5] A. Srivastava, R. Kastner, and M. Sarrafzadeh, "On the complexity of gate duplication," IEEE Trans. Computer-Aided Design, Sept. 2001, to be published.

[6] ——, "Complexity issues in gate duplication," in Workshop Notes, International Workshop on Logic Synthesis, May 2000.

[7] ——, "On the Complexity of Gate Duplication, Internal Report," ER-Lab, Univ. California, Los Angeles, 2000.

[8] ——, "Timing driven gate duplication: Complexity issues and algorithms," in Proc. Int. Conf. Computer-Aided Design, Nov. 2000.

[9] C. Chen and C. Tsui, "Timing optimization of logic network using gate duplication," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan. 1999, pp. 233–236.

[10] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Department of EECS, Univ. of California, Berkeley, Memorandum no. UCB/ERL M92/41, 1992.

[11] H. Vaishnav and M. Pedram, "Routability driven fanout optimization," in *Proc Design Automation Conf.*, June 1993, pp. 230–235.

[12] I. Neumann, D. Stoffel, H. Hartje, and W. Kunz, "Cell replication and redundancy elimination during placement for cycle time optmization," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1999, pp. 25–30.

[13] J. Darringer, D. Brand, J. Gerbi, W. Joyner, and L. Trevillyan, "LSS: A system for production logic synthesis," *IBM J. Res. Develop.*, vol. 28, no. 5, pp. 326–328, Sept. 1985.

[14] J. Lou, A. Salek, and M. Pedram, "An exact solution to simultaneous technology mapping and linear placement problem," in *Proc Int. Conf. Compute-Aided Design*, Nov. 1997, pp. 671–675.

[15] K. Bartlett, W. Cohen, A. J. De Geus, and G. D. Hachtel, "Synthesis of multi-level logic under timing constraints," *IEEE Trans. Computer-Aided Design*, October 1986.

[16] K. C. Chen and S. Muroga, "Timing optimization for multilevel combinational networks," in *Proc Design Automation Conf.*, June 1990, pp. 339–344.

[17] K. J. Singh, A. R. Wang, R. K. Brayton, and A. L. S. Vincentelli, "Timing optimization of combinational logic," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1988, pp. 282–285.

[18] C. Kring and A. R. Newton, "A cell replication approach to mincut-based circuit partitioning," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 2–5.

[19] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal elmore delay," in *Proc Int. Symp. Circuits and Systems*, Dec. 1990, pp. 865–868.

[20] M. Enos, S. Hauck, and M. Sarrafzadeh, "Evaluation and optimization of replication algorithms for logic bipartitioning," *IEEE Trans. Computer-Aided Design*, pp. 1237–1248, Sept. 1999.

[21] M. Pedram and N. Bhat, "Layout driven logic restructuring/decomposition," in *Proc Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 134–137.
[22] ——, "Layout driven technology mapping," in *Proc Design Automation Conf.*, June 1991, pp. 99–105.
[23] G. De Micheli, "Performance oriented synthesis of large-scale domino CMOS circuits," *IEEE Trans. Computer-Aided Design*, pp. 751–765, Sept. 1987.
[24] R. Murgai, "On the global fanout optimization problem," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1999, pp. 511–515.
[25] O. Coudert, R. Haddad, and S. Manne, "New algorithms for gate sizing: A comparative study," in *Proc Design Automation Conf.*, June 1996, pp. 734–739.
[26] P. Chan, "Algorithms for library specific sizing of combinational circuits," in *Proc Design Automation Conf.*, June 1990, pp. 353–356.
[27] R. Carragher, R. Murgai, S. Chakraborty, M. R. Prasad, A. Srivastava, and N. Vemuri, "Layout driven logic optimization," presented at the Proc. Workshop Handouts, International Workshop on Logic Synthesis, Dana Pt., CA, May 2000.
[28] R. K. Brayton, G. D. Hachtel, and A. L. S. Vincentelli, "Multilevel logic synthesis," *Proc IEEE*, vol. 78, no. 2, pp. 264–300, Feb. 1990.
[29] T. Okamoto and J. Cong, "Interconnect layout optimization by simultaneous steiner tree construction and buffer insertion," in *Proc. ACM/SIGDA Physical Design Workshop*, 1996, pp. 1–6.
[30] H. Touati, "Performance Oriented Technology Mapping," Ph.D. dissertation, Univ. of Calif., Berkeley, 1990.
[31] Y. Jiang and S. Sapatnekar, "An integrated algorithm for combined placement and libraryless technology mapping," in *Proc Int. Conf. Computer-Aided Design*, Nov. 1999, pp. 102–105.

Ankur Srivastava (M'02) received the B.S. degree from the Indian Institute of Technology, Delhi, the M.S. degree from Northwestern University, Evanston, IL, and the Ph.D. degree from the University of California Los Angeles (UCLA) in 1998, 2000, and 2002.

Since fall 2002 he has been an Assistant Professor with the University of Maryland, College Park. His research interests include all aspects of design automation including logic and high-level synthesis, low-power issues and predictability, and low-power issues pertaining to sensor networks.

Dr. Srivastava received the outstanding Ph.D. Award from the Computer Science Department of UCLA in 2002. He is a Member of ACM.

Ryan Kastner received the B.S. and M.S. degrees in electrical and computer engineering from Northwestern University, Evanston, IL, and the Ph.D. degree in computer science at the University of California, Los Angeles.

He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. His current research interests lie in the realm of embedded systems, in particular reconfigurable computing, compilers, and sensor networks.

Chunhong Chen received the B.S. and M.S. degrees in electrical engineering from Tianjin University China and the Ph.D. degree in electrical engineering from Fudan University, Shanghai, China.

From 1986 to 1996, he was with Zhejiang University of Technology first as an Assistant, and then Associate Professor. From 1997 to 1998, he was Research Associate at Hong Kong University of Science and Technology. From 1999 to 2001, he was a Postdoctoral Fellow first at Northwestern University, Evanston, IL, and then at the University of California, Los Angeles. Since 2001, he has been Assistant Professor with the University of Windsor, ON, Canada.

Majid Sarrafzadeh (M'87–SM'92–F'96) received the B.S., M.S., and Ph.D. degrees, in electrical and computer engineering, from the University of Illinois at Urbana-Champaign, in 1982, 1984, and 1987, respectively.

In 1987, he joined Northwestern University, Evanston, IL, as an Assistant Professor. In 2000, he joined the Department of Computer Science at the University of California, Los Angeles (UCLA). He has published approximately 250 papers, is a coeditor of *Algorithmic Aspects of VLSI Layout* (Singapore: World Scientific, 1994), and co-author of *An Introduction to VLSI Physical Design*" (New York: McGraw Hill, 1996) and *Modern Placement Techniques* (Norwell, MA: Kluwer, 2003). His recent research interests include embedded and reconfigurable computing, VLSI CAD, and the design and analysis of algorithms.

Dr. Sarrafzadeh received a National Science Foundation Engineering Initiation Award, two Distinguished Paper Awards in ICCAD, and the Best Paper Award in DAC. He has served on the technical program committees of numerous conferences in the area of VLSI Design and CAD, including ICCAD, DAC, EDAC, ISPD, FPGA, and DesignCon. He has served as committee chair of a number of these conferences. He is on the executive committees or steering committees of several conferences including ICCAD, ISPD, and ISQED. He is on the Editorial Board of the *VLSI Design Journal*, an Associate Editor of *ACM Transaction on Design Automation* (TODAES) and an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN (TCAD).