# Timing Driven Gate Duplication: Complexity Issues and Algorithms

Ankur Srivastava, Ryan Kastner and Majid Sarrafzadeh
Department of Electrical and Computer Engineering
Northwestern University
Evanston, Illinois 60208, USA
ankur,kastner,majid@ece.nwu.edu

## Abstract

This paper addresses the issue of timing driven gate duplication for delay optimization. Gate duplication has been used extensively for cutset minimization but the usefulness in minimizing the circuit delay has not been addressed. This paper studies the complexity issues in timing driven gate duplication and proposes an algorithm for solving the so called global gate duplication problem. Delay improvements over highly optimized results from SIS have been reported.

# 1 Introduction

Delay optimization is a fundamental goal in logic synthesis. This paper presents gate duplication as a strategy for performance optimization. In the past few years, the research community has looked at gate duplication extensively as a method of reducing the cut-set of partitions [6], [7]. However applicability of this strategy in reducing the circuit delay has not been studied in detail. One of the few works is [5] which addresses the gate duplication problem in a performance driven perspective and [3] which proposes gate duplication in the pre-mapping phase of synthesis.

The global (pertaining to the whole circuit) delay optimization problem by gate duplication is NP-Complete [2]. In this paper we prove the problem of *partitioning a set of fanouts between a gate and it's replica (both gates have the same fanins) such that the required time constraint at the input pin is met*, to be NP-Complete. Hence even the local optimization by gate duplication problem (formally defined later) is also NP-Complete. We then present an algorithm for gate duplication which is based on the dynamic programming approach. We report delay improvements as high as 8% over highly optimized results generated by SIS [4].

The rest of this paper is organized as follows. Section 2 deals with the delay model and preliminaries. Section 3 reviews the complexity of the global gate duplication problem. Section 4 studies the complexity of the local gate duplication problem. Section 5 describes a heuristic for gate duplication in detail, followed by results in Section 6 and conclusion in Section 7.

# 2 Preliminaries

## 2.1 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 $\gamma_p$ for all fanouts $p$ of $g$. At certain instances this paper refers to $\gamma i, g$ as the input pin capacitance of gate $g$ at pin $i$. A commonly used delay model for gate level circuits is the load dependent delay model **LDDM** [8] 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 $g$,

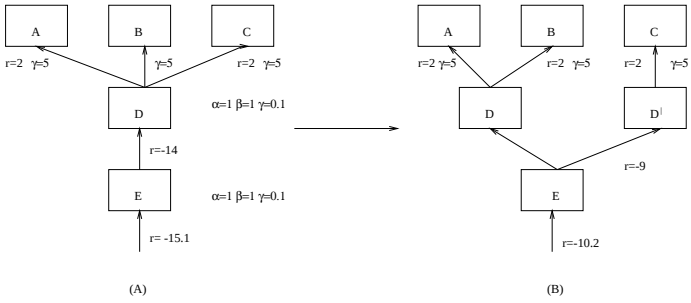


Figure 1: Delay Optimization by Gate Duplication

$\beta_{i,g}$ = drive capability or load coefficient of the path from $i$ to the output of $g$.

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 $x$ w.r.t. the pin connected to $g$,
$\beta_{g,x}$ = load coefficient of gate $x$ w.r.t 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. Although the algorithm for gate duplication can be easily extended to models with wireload delay. 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 gates is always negative. In this scenario the objective of our delay optimization algorithm is the maximization of the minimum slack in the circuit.

## 2.2 The Gate Duplication Problem

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

*Consider the circuit shown in Figure 1(A) in which the parameters $\alpha$,$\beta$ and $\gamma$ along with the required times have been indicated. We will show that the delay through this circuit can be improved by duplicating some gates. In the unduplicated case (Figure 1(A)) the capacitive loading $C_D$ = 5+5+5 and $C_E$ = 0.1. Hence the required time at the input of E can be calculated to be -15.1. When D is duplicated (Figure 1(B)), the capacitive loading $C_D$ = 5+5 and $C_E$ = 0.2. Hence the new required time at the input of E becomes -10.2. Gate duplication was hence instrumental in improvement of circuit delay.*

# 3 Global Optimization by Gate Duplication

The global gate duplication problem is concerned with delay optimization of the entire network.

1. Given a network $\eta$ consisting of gates and nets
2. Given the delay parameters $\alpha_{i,g}$ $\beta_{i,g}$ and $\gamma_i$ 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). The algorithm must follow the constraint that it cannot duplicate the PIs and POs.

We have shown it to be NP-Complete in the **LDDM** [2]. This problem remains NP-Complete even when the delay model is simplified to the following

$$\delta_g = \alpha_g + \beta_g c_g \tag{4}$$

The $\alpha$, $\beta$ etc parameters are the same for different input pins of a gate $g$. [9] shows that the global fanout optimization problem (buffer insertion) with fixed net topology (GFO-NTF) can be solved in polynomial time with the above delay model (same pin to pin parameters). Hence global gate duplication is in general harder than global buffer insertion.

Any gate duplication algorithm will have to make at least two decisions on which the final result will depend. First which gates to duplicate and second what the fanouts of the duplicated gates should be.

In the next section we prove that given a node and it's duplicate, the problem of partitioning a set of fanouts between the original and the duplicated node for meeting the required time constraint at the input pin is also NP-Complete.

# 4 Local Optimization by Gate Duplication

Let us first define the *local gate duplication* problem.

1. Given a node $n$ and a set of fanouts that it drives
2. Given the required times at the input pins of all the fanouts
3. Given all gate parameters $\alpha_{i,g}, \beta_{i,g}, \gamma_i$
4. Only gate $n$ can be duplicated
5. Maximize the required time at the input of gate $n$ using gate duplication

In the worst case we will have to look at an exponential number of choices (exponential in the number of fanouts). Here a choice is defined as a partitioning of the fanouts between the gate $n$ and it's duplicate $n'$. The gate $n$ and it's duplicate $n'$ must have common fanins.

In this section we show the problem of partitioning the fanouts between the original and the replica for meeting the required time constraint at the input pin is NP-Complete. The decision problem (L-GD) can be stated as follows.

INSTANCE: Given gates $n$ and $n'$ where $n'$ is a replica of $n$ (with common fanins). Initially $n'$ does not drive any fanouts. Given the fanouts of $n$ and the required time at their inputs. Given the $\alpha_{i,n}$ $\beta_{i,n}$ and $\gamma_i$ for each input pin i of $n$ (or $n'$), $\gamma$ for each fanout and a number D.

QUESTION: Does there exist a partitioning of the fanouts FO between $n$ and $n'$ such that the required time at the input pins of $n$ (and $n'$) is at least D.

**Theorem 4.1** L-GD is NP-Complete.

Proof: Please refer to [2], [1]

The NP-Completeness of L-GD makes the local gate duplication problem (defined above) also NP-Complete. Note that local fanout optimization with fixed net topology has polynomial time algorithms. In the next section we present an algorithm for this problem.

# 5 Algorithm for Gate Duplication

In this section, we discuss an algorithm for solving the global problem. It is based on dynamic programming and is divided into three stages. The initial stage visits each node in topologically sorted order (from PO to PI) and calculates the required time at each input pin of a gate when that gate is duplicated and unduplicated. The second stage visits each node (in topological order from PI to PO) and decides if the gate should be duplicated. The final pass (in topological order from PO to PI) physically does the duplications. The next few subsections describe the algorithm in detail. We use the terms nodes and gates interchangeably.

## 5.1 Stage 1: Tuple Evaluation

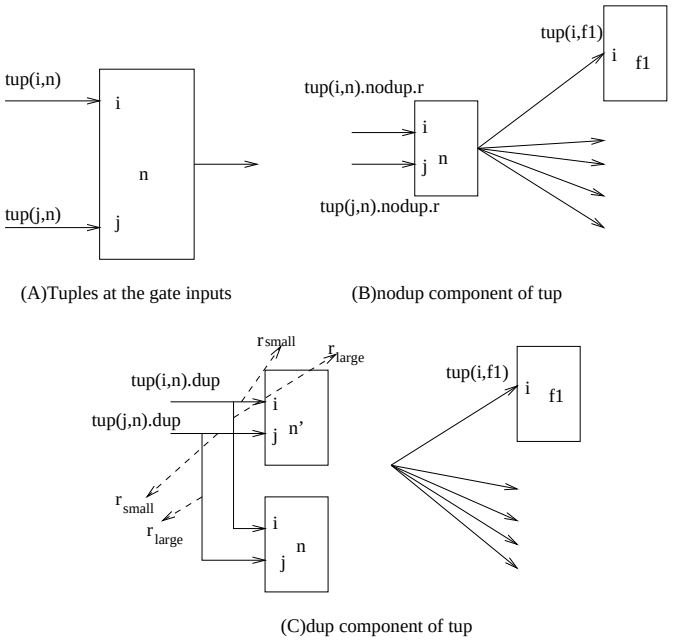


Figure 2: Tuple Evaluation

In this stage we traverse the network in topologically sorted order from POs to PIs. This ensures that when a node is being processed, all it's fanouts have already been processed. Before we proceed with the detailed explanation let us define some terminology.

1. A tuple $tup(i,g)$ is associated with all input pins i of all gates g of the network
2. $tup(i,g)$ has two components $tup(i,g).nodup.r$ and $tup(i,g).dup$. $tup(i,g).dup$ has two sub-components $tup(i,g).dup.r_{small}$ and $tup(i,g).dup.r_{small}$

This stage computes tuples for each input pin of each gate $g$. The topo-sorted order ensures that when a tuple is being evaluated for $g$, the tuples of all the fanouts of $g$ have already been evaluated. The first component $tup(i,g).nodup.r$ is the best(largest) required time at the input pin $i$ if that gate is not duplicated (Figure 2(B)). The second component $tup(i,g).dup$ is associated with duplication of the node. If the node $g$ is duplicated into $g$ and $g'$ (see Figure 2(C)), then the two sub-components $tup(i,g).dup.r_{small}$ and $tup(i,g).dup.r_{large}$ correspond to required times at the ith input of $g$ and $g'$ as shown in Figure 2(C). Here $r_{small}$ is smaller of the two and $r_{large}$ is larger of the two. The objective for maximization while computing $tup(i,g).dup$ is $r_{small}$. There might be cases in which a gate cannot be duplicated, (for instance if it has only single fanout). In such a situation $tup(i,g).dup$ is set to NULL. Let us define another term *fanout script* as follows

*Fanout script associated with an input pin of a node is a vector of boolean variables. Each boolean variable corresponds to a fanout.*

**Algorithm 1** $EVAL_{tup(p,n).nodup}$

$n$ - node under consideration
$p$ - input pin under consideration
Sort the fanouts f in increasing order of their $tup(n,f).nodup.r$ entries
/*$tup(n,f).nodup.r$ is the tuple of fanout gate f at input pin connected to n */
/*Call this array A*/
/*Let the ith entry A correspond to the ith fanout */
$maxreq$ = min (All entries in A)
$maxreq = maxreq - \delta(p,n)$
/* $\delta(p,n)$ is the delay in gate $n$ w.r.t pin $p$ */ /*maxreq is the required time at input pin p if none of the fanouts are duplicated*/
maxindex = 0
capload = 0
for i=0; i< $number\_of\_fanouts$; i++
    if ($tup(n,i).dup! = NULL$)
        Replace the ith entry in A by $tup(n,i).dup.r_{small}$
        /* $tup(n,i)$ is the tuple associated with fanout i at pin which connects to gate n */
        Insert $tup(n,i).dup.r_{large}$ at the end of A
        /*note that this amounts to the duplication of fanout i */
        $capload = capload + \gamma_{n,i}$
        /*$\gamma_{n,i}$ is input pin capacitance of gate i at pin connected to $n$ */
        $curr\_req$ = min (all entries currently in A)
        $curr\_req = curr\_req - \delta(p,n) - capload\beta_{p,n}$
        /*the third term in this equation corresponds to increase in the capacitive loading if i gates are duplicated */
        /* curr_req has the current required time at input pin p of node n if the first i fanout gates in A are duplicated */
        if ($curr\_req > maxreq$)
            $maxreq = curr\_req$
            $maxindex = i + 1$
$tup(p,g).nodup.r = maxreq$
/* maxindex contains the information on fanout script /*
/* The first maxindex terms in the original sorted array should be duplicated */
end

Figure 3: $tup(p,n).nodup$ evaluation

*If the boolean variable is TRUE then that fanout is designated for duplication else it is not duplicated.*

The process of evaluating the $tup(i,g).nodup.r$ corresponds to finding an optimal fanout script for each input pin $i$ such that the required time at that pin is maximized, (note that the gate under consideration cannot be duplicated). This fanout script may not be the same for different input pins of the gate. Figure 3 has a formal description of the algorithm used in computation of $tup(i,g).nodup.r$. Basically it sorts the fanouts f by their $tup(g,f).nodup.r$ value, where $tup(g,f)$ is the tuple associated with fanout f at the input pin connected to gate $g$. It then duplicates the first b fanouts with b varying from 0 to number_of_fanouts (total number_of_fanouts +1 possibilities). Each value of b corresponds to a distinct fanout script. Hence this algorithm enumerates number_of_fanouts + 1 fanout scripts and picks the one that gives the maximum required time at that input pin.

We state the following lemma; please refer to [2] for proof.

**Lemma 5.1:** Given a gate g and an input pin i of g, $tup(i,g).nodup.r \leq tup(i,g).dup.r_{small} \leq tup(i,g).dup.r_{large}$.

**Theorem 5.1**: Consider a gate g and input pin i of g, a set of fanouts FO of g and the tuples associated with all the FOs. Algorithm $EVAL_{tup(p,n).nodup}$ gives the optimal $tup(i,g).nodup.r$ value. Where optimality is defined as the largest value of $tup(i,g).nodup.r$.

**Algorithm 2** $EVAL\_PARTITION$

$n$ - node under consideration
$p$ - input pin under consideration
$A$ - Array of fanouts and their required times at input pins
Sort the fanouts f in increasing order of their required time entries
If two fanouts have same required time then insert the fanout with higher input pin capacitance first in the sorted list
Call this List L /*Let the ith entry L correspond to the ith fanout */
$req\_part0 = L[0] - \alpha_{p,n} - \beta_{p,n}\gamma_{n,0}$
$req\_part1$ = min(all entries in L starting from 1) - $\alpha_{p,n} - \beta_{p,n}\sum_{i:1..number\_of\_fanouts-1}(\gamma_{n,i})$
$tup(p,n).dup.r_{small} = min(req\_part0, req\_part1)$
$tup(p,n).dup.r_{large} = max(req\_part0, req\_part1)$
partindex = 0
for i=1; i< $number\_of\_fanouts - 1$; i++
    $req\_part0$ =
    min(all entries in L upto i) $-\alpha_{p,n} - \beta_{p,n}\sum_{j:0toi}(\gamma_{n,j})$
    $req\_part1$ = min(all entries in L from i+1 upto end) $-\alpha_{p,n} - \beta_{p,n}\sum_{j:i+1..number\_of\_fanouts-1}(\gamma_{n,j})$
    if ($min(req\_part0, req\_part1) > tup(p,n).dup.r_{small}$
        $tup(p,n).dup.r_{small} = min(req\_part0, req\_part1)$
        $tup(p,n).dup.r_{large} = max(req\_part0, req\_part1)$
        partindex = i
partindex - contains the partitioning location in the sorted list L
end

Figure 4: Partitioning the fanout set into two

**Proof:** Please refer to [2].

The next step is the evaluation of $tup(i,g).dup$ for the ith input pin of gate g. It involves two steps. The first is the selection the fanout script and second is partitioning the fanouts between the node and it's duplicate. In each fanout script there will be some fanout gates that would be set for duplication. Hence the number of fanouts of $g$ will change with the fanout script. The objective as mentioned before is to maximize $tup(i,g).dup.r_{small}$. Theorem 4.1 showed that the problem of partitioning a set of fanouts between a node and it's duplicate is NP-Complete. Hence we use suboptimal heuristics to obtain a good $tup(i,g).dup$ value. In this case also we enumerate the $number\_of\_fanouts + 1$ fanouts scripts considered by $EVAL_{tup(p,n).nodup}$. For each fanout script, we generate the new fanout set (we will have more fanouts to drive as some gates might have been duplicated). An array is populated with this new fanout set. If a fanout f is duplicated then $tup(g,f).dup.r_{small}$ and $tup(g,f).dup.r_{large}$ are inserted in this array. These correspond to the required times at the input pin of the fanout f and it's duplicate. If the fanout f is not duplicated then $tup(g,f).nodup.r$ is inserted in this array. Hence we get an array fanouts and their required time at the input pins that the node and it's replica will have to drive. The next step is partitioning of this fanout set into two and computation of $tup(i,g).dup.r_{small}$ and $tup(i,g).dup.r_{large}$ such that $tup(i,g).dup.r_{small}$ can be maximized. The partitioning strategy is formally described in Figure 4. This algorithm basically sorts the array which contains the required time information of the fanouts (fanouts given by the fanout script under consideration). It then generates partitions in this sorted list and picks up the best partition. Note that it gives the good $tup(i,g).dup$ for a particular fanout script. The best $tup(i,g).dup$ is chosen from the $number\_of\_fanouts + 1$ fanout scripts that we consider.

**Theorem 5.2:** Consider an array of fanouts, the required time at their input pins, a node n and it's duplicate n'(with common fanins) and an input pin p of n. If the input pin capacitances of all the fanouts are the same then algorithm $EVAL\_PARTITION$ generates an optimal partition for the given fanout set, where optimality is defined as $max(min(r(p,n), r(p,n')))$ (see equation 3).

**Proof:** Please refer to [2].

## 5.2 Stage 2 & 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 it's 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.

**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. This is a polynomial time algorithm.

# 6 Results

We integrated our algorithm in SIS [4] and obtained results for ten MCNC benchmarks. We used gate duplication as a post mapping step. The results were obtained in two categories. The first shows the effectiveness of the algorithm on a general network and the second category improves upon the best result of SIS. Circuit optimized by script.rugged was further optimized by speed_up for minimum delay. For the first category this was followed by technology mapping for minimum delay (map -n 1 option). The results obtained by this category are illustrated in Table 1. The large improvements in delay show the effectiveness of the algorithm.

In the second category we use the map -n 1 -AFG option of the mapper which produces highly optimized results by doing better fanout optimization and area recovery. Table 2 shows the delay improvements and area penalty by our algorithm. In order to constrain the area penalty, we duplicated only $\epsilon$-critical gates. An $\epsilon$-critical gate has slack $s$ such that $\text{min\_circuit\_slack} \leq s \leq (1-\epsilon)\text{min\_circuit\_slack}$. Increasing $\epsilon$ will increase the candidate gates for duplication, hence increasing the area penalty and potentially increasing the delay improvement. The parameter $\epsilon$ was set as an input parameter and it could be varied for getting a range of area, delay values for each benchmark. Cases $x4$ and $frg1$ were specifically interesting. They gave delay improvements of 8% and 6.9% for small area penalties. 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. 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. We propose that the algorithm should be applied only to certain critical sections of the circuit. A large area penalty in these sections will not amount to a large increase in the overall circuit area. Whereas any improvement in the delay of this critical section will be reflected by decrease in the overall circuit delay.

# 7 Conclusion and Future Directions

We presented an algorithm for gate duplication and showed it's effectiveness in improving the circuit delay. In section 4 we showed that the problem of partitioning the fanouts between a node and it's replica is NP-Complete. This makes local optimization by gate duplication NP-Complete. We suggested the heuristic EVAL_PARTITION for solving the fanout partitioning problem. Our future research will focus on strategies of merging duplication with gate sizing and buffer insertion. We will also develop layout driven algorithms for gate duplication.

| Bench | orig ar | orig delay | area inc | delay dec |
|---|---|---|---|---|
| frg1 | 154976 | 9.31 | 12.8% | 18.8% |
| x4 | 516432 | 16.15 | 3.2% | 26.6% |
| pair | 2197504 | 24.94 | 14% | 26.5% |
| i10 | 3194640 | 50.78 | 8.1% | 26.4% |
| rot | 1023584 | 16.18 | 5.5% | 9.3% |
| dalu | 1906576 | 41.67 | 4.4% | 16.4% |
| pcler8 | 148016 | 10.19 | 8.2% | 33.76% |
| des | 4173216 | 77.67 | 5.6% | 75.8% |
| too_large | 436624 | 11.49 | 10.8% | 7.2% |
| C432 | 463536 | 34.4 | 14% | 21.7% |
| average | | | **8.7%** | **26.2%** |

Table 1: Delay improvements and area penalty over map -n 1

| Bench | orig ar | orig delay(ns) | area inc | delay dec |
|---|---|---|---|---|
| frg1 | 160544 | 8.23 | 6% | 6.9% |
| x4 | 512256 | 9.62 | 12% | 8.1% |
| pair | 2155280 | 17.46 | 12% | 6.5% |
| i10 | 2997440 | 31.82 | 18% | 5% |
| rot | 983216 | 13.18 | 13% | 6.14% |
| dalu | 1830976 | 26.04 | 11% | 2.6% |
| pcler8 | 136880 | 6.44 | 14% | 4.2% |
| des | 3941680 | 15.41 | 13% | 3.6% |
| too_large | 431056 | 10.05 | 8.8% | 3.2% |
| C432 | 453792 | 22.84 | 10% | 3.0 |
| average | | | **11.8%** | **4.9%** |

Table 2: Delay improvements and area penalty over map -n 1 -AFG

# References


[1] A. Srivastava, R Kastner and M. Sarrafzadeh. "Complexity Issues in Gate Duplication". In *Workshop Handouts, International Workshop on Logic Synthesis*, May 2000.

[2] A. Srivastava, R Kastner and M. Sarrafzadeh. "Gate Duplication for Performance Optimization". In *Internal Memo, Northwestern University*, June 2000.

[3] C. Chen and C. Tsui. "Timing Optimization of Logic Network using Gate Duplication". In *Proc. Asia and South Pacific Design Automation Conference*, pages 233–236, January 1999.

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

[5] I. Neumann, D. Stoffel, H. Hartje and W. Kunz. "Cell Replication and Redundancy Elimination During Placement for Cycle Time Optmization". In *Proc. International Conference on Computer Aided Design*, pages 25–30, November 1999.

[6] C. Kring and A.R.Newton. "A Cell replication Approach to Mincut-Based Circuit Partitioning". In *Proc. International Conference on Computer Aided Design*, pages 2–5, November 1991.

[7] M. Enos, S. Hauck and M. Sarrafzadeh. "Evaluation and Optimization of Replication Algorithms for Logic Bipartitioning ". In *IEEE Transactions on Computer Aided Design*, pages 1237–1248, September 1999.

[8] R. Murgai. "On the Complexity of Minimum-delay Gate Resizing/Technology Mapping under Load-dependent Delay Model". In *Workshop Handouts, International Workshop on Logic Synthesis*, pages 209–211, June 1999.

[9] R. Murgai. "On the Global Fanout Optimization Problem". In *Proc. International Conference on Computer Aided Design*, pages 511–515, November 1999.