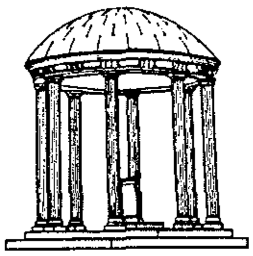
The Right Stuff-
Techniques for High Speed CMOS Circuit Synthesis

TR89-046

December, 1989

Michael S. Kotliar



The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

The Right Stuff — Techniques for High Speed CMOS Circuit Synthesis

by
Michael S. Kotliar

A dissertation submitted to the faculty of The University of North Carolina at Chapel Hill in partial fulfillment of the requirments for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill

1989

Approved by:

Advisor: Kye S. Hedlund

Akhilesh Tyagi

Reader: Akhilesh Tyagi

[illegible]

Reader: John Poulton

MICHAEL SETH KOTLIAR. The Right Stuff – Techniques for High Speed CMOS Circuit Synthesis (Under the direction of Kye S. Hedlund)

ABSTRACT

This dissertation presents a method for improving the speed of combinational CMOS circuits. The original circuit is replaced by a faster circuit with equivalent function by restructuring the gates and their connections in the circuit. This restructuring has more flexibility and greater potential for reducing delays than other methods such as transistor sizing. However, this increased flexibility comes at a cost of increased problem complexity, so to solve this problem in an efficient manner, we use greedy heuristics that have been augmented with knowledge of the problem.

The restructuring is accomplished by a set of transformations that are applied to the circuit in both a local and global manner. One of the transformations described, path resynthesis, is a new technique and its operation is described.

The amount of speed-up possible using restructuring is illustrated using a set of representative circuits from a standard benchmark set. Since the reduction in delay is not without cost, the cost associated with the increased speed, namely increased circuit area, is also studied using the set of representative circuits.

ACKNOWLEDGMENTS

As with any large project, this work benefitted from the help and wisdom of many individuals. First, my advisor Kye Hedlund provided advice, guidance, and encouragement which made this work possible. Also, I wish to thank the other members of my committee, Akhilesh Tyagi, Gershon Kedem, John Poulton, and Dean Brock for their help and suggestions.

I would also like to thank Yuki Watanabe and Jay Nievergelt for serving on my committee and for their interesting discussions with me. Bob Lisanke and Franc Brglez from MCNC also provided helpful discussions early on in this work, and they provided access to and help with their logic synthesis tool, DECAF. I am indebted to Dick Craddock for his work on AESOP, the team of programmers from the University of California at Berkeley that developed MISII, and the University of Colorado at Boulder group that developed BOLD. Thanks also goes to the faculty, students and staff of the Department of Computer Science for making my stay here enjoyable.

Finally, special thanks must go to my wife, Shou-Huey, for the endless encouragement, patience, support, and especially the sacrifice that allowed me to complete my dissertation. Last but not least, I thank Baby Ninja for behaving so well during the last few months of this work.

柯麦克

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AT^2	area times square of time, efficiency of circuit
β	MOS transistor gain
B^n	n-dimensional Boolean space
BLIF	Berkeley Logic Interchange Format
BSD	Berkeley Standard Distribution
C	capacitance
C_L	load capacitance
CPU	central processing unit
d	delay of a gate
DAG	directed acyclic graph
ϵ	permittivity of the gate insulator
η	Boolean network
F	Boolean function
f	frequency
GND	ground
GNU	GNU is not UNIX (recursive)
IV	intermediate variable
k	transistor scaling factor

L	length of transistor or channel
μ	effective mobility of electrons in channel
P	power dissipation
PI	primary input
PLA	programmable logic array
PO	primary output
R	resistance
s	slack time of a gate
τ	rise/fall time of a gate
T	reciprocal of frequency
t_i	ready time for the input to a gate
t_r	ready time of a gate
t_{ox}	thickness of the gate insulator
$\hat{t}$	required ready time of a gate
V	voltage
V_{dd}	supply voltage
V_T	threshold voltage of transistor
W	width of transistor or channel

Chapter I
Introduction

1.1 VLSI Design Process

Very Large Scale Integration (VLSI) circuit design is the process of building an integrated circuit, or chip, from a high level behavioral description of the circuit. This problem is typically split into three phases: circuit design, mask generation, and chip fabrication. The first phase, circuit design, determines the implementation of the circuit from the behavioral description, including deciding on modules such as register files, adders, and random logic, and determining the logic gates necessary for their implementation. The second phase, mask generation, produces the actual physical masks used for fabrication from the logic description. This phase includes floorplanning, cell generation, and placement and routing. In the final phase, chip fabrication, the masks are sent to a silicon foundry to produce wafers which are then cut into die and packaged into chips. These phases are illustrated in Figure 1.1. The work described here occurs in the circuit design phase.

One way of representing the different parts and levels of the design is the Gajski-Kuhn diagram[GK83]. This diagram, shown in Figure 1.2, illustrates the different representations of the circuit as axis on the diagram. Proceeding out from the center on an axis increases the level of abstraction for the particular representation. The functional axis represents the behavior or architecture of the system. This functional description is the input to the circuit design phase. The structural axis represents the components used, or the implementation, and is produced during the circuit design phase. The geometrical axis represents the physical design, or realization, and is produced by the mask generation phase.

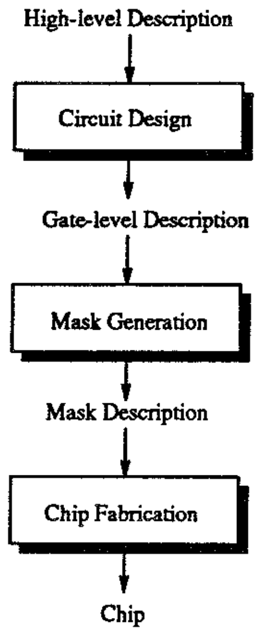


Figure 1.1: The VLSI Design Process

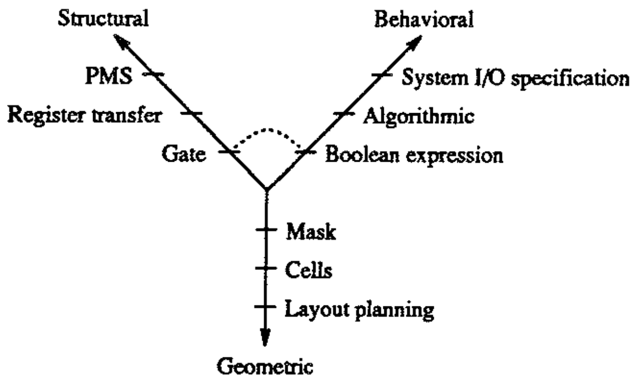


Figure 1.2: Representation of Design

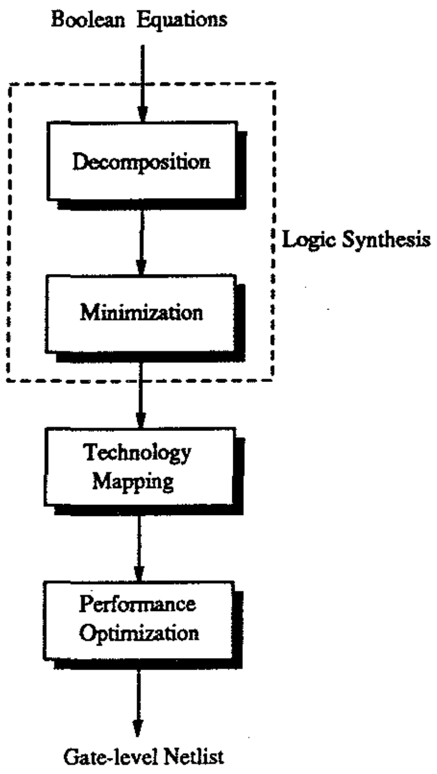


Figure 1.3: Circuit Design Process

1.2 Circuit Design

Circuit design is a difficult part of the design process since a designer must trade-off three resources: area, time, and power, and there are limits on each of these resources. Large chips have much lower yields than smaller chips, so a designer must keep the design as small as possible. The chip also must operate at a particular speed set by the design specifications. Finally, there is only a limited amount of power that a chip can safely dissipate. Unfortunately, it is impossible to minimize all three factors simultaneously since they are not independent. For example, faster circuits often require larger transistors, which take up more area and dissipate more power. Another examples of this problem is evident when using gates with large fan-in. These gates can produce a small circuit, but are slower than gates with less fan-in. However, using logic gates with less fan-in requires more gates in the circuit, and so increases the area of the circuit. Therefore, a designer must find the right balance of these three resources so that the circuit can operate at the required speed, while not exceeding the area limit or power budget.

The remainder of this dissertation is limited to discussing the phase of circuit design that determines the gate implementation from the Boolean level description, as shown by the dotted arc in Figure 1.2. The other aspects of the implementation, while important to the design of the chip, are not part of the work described here.

The design of gates from Boolean equations is typically broken into three steps: decomposition, minimization, and technology mapping. An optional fourth step, performance optimization, is also part of this process. These steps are illustrated in Figure 1.3. When this process is automated, the first two steps are usually combined into a single step, called logic synthesis. The final step, performance optimization, is the focus of this research, although each of these steps is described in detail, below.

1.3 Logic Synthesis

Logic synthesis consists of two steps: decomposition of a set of two-level Boolean equations into a set of multi-level Boolean equations, and the minimization of the set of multi-level Boolean equations. Decomposition is necessary because, when implemented as gates, multi-level Boolean equations require less space than two-level Boolean equations and can produce faster circuits. In the minimization step, the number of literals in the circuit are minimized, reducing the area of the circuit, since the number of literals is proportional to the area. The computational complexity of minimization is co–NP-hard[KR89], but some useful heuristics have been developed that return satisfactory results[BHMSV84, BHJ+87, BCDH86, BRSVW87, DeM87, LKB87, MB88, HJ88, Bra83, Sas86, BS89]. A more detailed overview may be found in [Tre87].

1.4 Technology Mapping

After minimizing the Boolean equations, they must be converted into gates from the implementation technology. This operation, called technology mapping, replaces each Boolean equation with one or more logic gates. The choice of gates is limited to what is available in the particular implementation technology. Finding the best mapping of the circuit is an NP-complete problem[KR89], so the most common technique is to break the circuit into trees of functions and then map the individual trees into gates.

1.5 Performance Optimization

1.5.1 Why is it needed?

Logic synthesis' primary goal is to minimize the number of literals in the set of Boolean equations, since this corresponds to minimizing the area of the circuit. However, logic synthesis tools emphasize area minimization often at the expense of circuit speed, and can produce circuits that are too slow for some applications. For example, when trying to increase sharing of gates, long chains of gates that can be very slow are produced. Technology mapping also often contributes to slow circuit because either delay is not considered during the mapping operation, or inaccurate delay models are used.

If the circuit produced by automatic tools is too slow, one alternative is for the designer to modify the circuit manually. Unfortunately this method is time consuming and error-prone and so is undesirable. Another alternative is to enlarge some of the transistors in the circuit. This technique, known as transistor sizing, matches the driving ability of a gate with the load that it must drive. A gate that must drive a large load has its transistors enlarged so it will be able to drive its large load. While transistor sizing is useful and has been successfully applied to many circuits[Hed84, FD85, Mat85, Hed87], the restructuring of the circuit by adding, removing, and modifying the gates and their connectivity is a more flexible approach. The latter method is the one described in this work.

As the size and complexity of VLSI designs increase and product life cycles decrease, there will be a greater reliance on automatic design and synthesis tools. The ability to automatically generate designs that meet certain speed requirements is therefore paramount. Automatic restructuring is a one technique for reducing delay and if applicable over a broad range of circuits, will allow automatically synthesized circuits to become more widely used, especially in the time critical parts of a chip. Therefore, this area is an important research topic.

1.5.2 Thesis Statement

A set of transformations, heuristically applied to a circuit, can produce significant reductions in delay with acceptable increases in the area of the circuit. These transformations provide greater reduction in delay than transistor sizing alone. A greedy approach combined with knowledge of the problem provides a useful heuristic to apply these transformations.

This work differs from earlier work in that it includes a larger set of transformations, including one not presented before, gives a more formal description of the different transformations, and provides improved heuristics for applying these transformations to the circuit.

1.5.3 Problem Statement

Given a combinational circuit described as a gate-level netlist, a set of timing constraints, and information about the implementation technology, such as intrinsic resistance and capacitance values, produce an equivalent gate-level netlist circuit that has the equivalent function of the input circuit but meets the timing constraints. It is desired that the size of the equivalent circuit is not excessively larger than the original circuit.

1.5.4 Why is this problem hard?

There are two factors that make this problem difficult. First, the number of transformations are large and there are many places in the circuit where they could be applied. Thus, there is a combinatorial explosion in the complexity of the problem. Second, the order of application can have a profound effect on the type of improvement possible. Different order of the place and type of transformation can preclude or allow other transformations being used later on.

1.5.5 Assumptions

Performance optimization converts a gate-level circuit description into an equivalent description that operates at a faster speed than the original circuit description. Two assumptions simplify this problem. First, the description is assumed to be purely combinational logic. This assumption simplifies the timing analysis of the circuit by preventing feedback paths. It is reasonable to make this assumption because it is consistent with the Mead and Conway design style[MC80] or the design for testability paradigm[McC86]. In this design style, shown in Figure 1.4, a register feeds into a combinational logic block that, in turn, feeds into another register. This second register may feed into another combinational logic block. Note that this design style is not limiting since most circuits can be represented in this form.

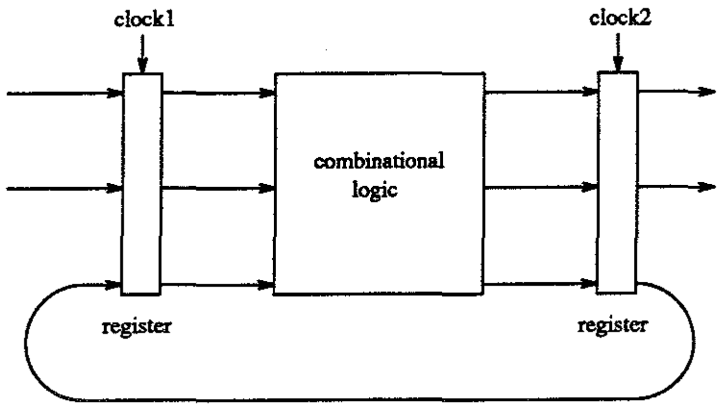


Figure 1.4: Mead and Conway Design Style

The second assumption is that there is no restriction on the type of gate available during the restructuring operations, as long as the gate is a complex, static, full-complementary CMOS gate. No nMOS or dynamic CMOS gates are allowed. Also, no bidirectional pass gates are permitted in the circuit. Unlike standard cells or gate arrays, however, there is no limitation to the size of the gate library. Since this technology is a superset of the common implementation technologies, this work can also be used when restricted to a particular and limited implementation technology.

1.5.6 Method

The general method to solve this problem is an iterative two-step approach. The first step is to find paths through the circuit that are too slow, called *critical paths.* The second step is to apply a transformation to these paths to reduce the delay. These two steps are repeated until either the circuit meets the timing constraints, or it has become clear that it is not possible to meet the timing constraints.

1.6 Dissertation Overview

In this dissertation, the theory and techniques for circuit optimization are described. Chapter 2 describes the previous work that has been done in this area. The models used to represent the problem are described in Chapter 3. In Chapter 4, the different transformations are described in detail. Then, in Chapter 5, the techniques used to apply the transformations to a circuit are presented. Chapter 6 contains the results of applying these techniques to a set of circuits. Finally, Chapter 7 concludes with a summary of this work and describes the direction of future research.

Chapter II
Previous Work

2.1 Introduction

Performance optimization of combinational circuits has been the subject of earlier research, and this chapter surveys the previous research that is closely related to the research presented here. There are two principal approaches to logic optimization. The first approach occurs in the context of technology mapping, and the second approach uses a separate optimization phase to reduce the delay in the circuit. This chapter begins with a brief discussion of technology mapping techniques, and then describes previously reported optimizations.

2.2 Technology Mapping

Technology mapping converts a set of Boolean equations into a set of logic gates. It is instructive to survey the most common methods used for technology mapping, since it is closely linked to performance optimization. The set of Boolean equations can be represented as a directed graph, with each equation represented by a node. A directed edge between two nodes occurs when the output of the Boolean equation associated with the first node is used as input in the Boolean equation of the second node. This graph is commonly called a Boolean network. If the Boolean equations describe a combinational circuit, then this graph is acyclic. A set of logic gates forming a circuit can also be represented as a graph. Each logic gate is represented by a node and a directed edge exists between two nodes if the output of one gate is used as the input of another gate. The problem of technology mapping can then be represented as a mapping between two graphs, with the goal of finding the optimal mapping that produces the best circuit, where best is usually defined as the least area. There are two primary techniques for technology mapping, tree-matching and graph-matching, both of which can be formulated as a problem of covering a DAG with a smaller library of DAGs. Since the problem of DAG covering is NP-complete, both of these techniques are approximate.

2.2.1 Tree Matching

To convert the graph of Boolean equations into a graph of logic gates, the first step in tree matching is to divide the Boolean network into trees by partitioning the graph. than one edge. This partitioning is done by removing all the edges emanating from all nodes that have more than one edge emanating from it, and this node forms the root of the tree. In other words, the outputs of all nodes that a fan-out greater than one are removed from the tree. For example, in Figure 2.1 a graph is partitioned into a forest of four trees. After the partitioning of the graph into trees is completed, each tree is first converted into a canonical representation of two-input nand gates and inverters, and then mapped into logic gates using a pattern match of the gates in the library. Splitting the graph into trees allows each tree to be optimally mapped into gates in linear time using a dynamic programming techniques derived from an algorithm used for code generation in a compiler. The algorithm works by recursively finding the best mapping of each subtree in the tree, where the area or the delay of the gate is used to determine the quality of the mapping. While this method is optimal for each tree, it cannot guarantee an optimal mapping for the entire circuit. Both DAGON[Keu87] and MIS[BRSVW87, SWBSV88] use this technique. MIS improves upon the basic technique used in DAGON by adding inverter pairs at each internal node of the tree, allowing more flexibility in the choice of gates. An alternative representation of the Boolean network is used in SKOL[Ber88]. Rather than representing the network in terms of two-input nand gates and inverters, it represents the network in terms a factors. These factors are allowed to contain no other factors, other than themselves. This representation allows arithmetic operations to be used rather than pattern matching.

2.2.2 Graph Matching

Instead of using the tree-matching method of finding an approximate solution by exactly solving a related problem, graph matching solves the original problem in an approximate way. MIS uses a greedy technique of selecting a subgraph that covers the largest portion of the circuit graph. Kahrs[Kah86] also uses a greedy strategy, but the choice of subgraph at a particular node is restricted so that it does not

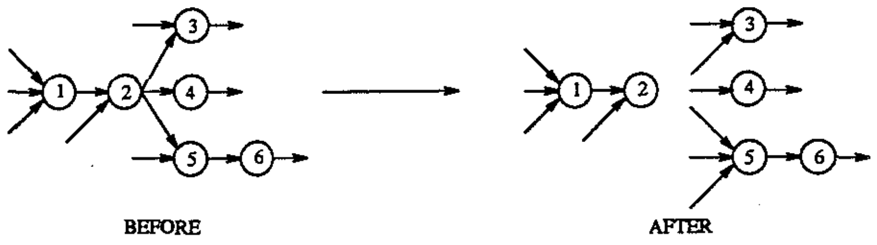


Figure 2.1: Partitioning a Graph into a Forest of Trees

overlap with other subgraphs already selected. Where comparisons of this technique to tree-matching are available[SWBSV88], tree-matching techniques produce smaller circuits than those produced by graph matching.

2.3 Optimizations

In previous work, only optimization techniques that reduce delay have been presented. Since the primary goal of logic synthesis has been area minimization, and technology mapping algorithms produce reasonably small circuits, separate techniques to reduce area have not been necessary. So, this section describes some of the delay optimizations that have been reported.

2.3.1 Fan-in Ordering

The selection of gates in technology mapping does not consider the arrival time of the inputs, even when optimizing for delay. MIS contains a timing optimization called fan-in ordering which modifies the structure of the Boolean network based on the input arrival time. Using a unit-delay model to measure delays and determine the critical path, this optimization collapses several gates on a critical path together and then decomposes the collapsed gate based on the arrival time of the input. This operation restructures the Boolean network so that slow signals travel through fewer stages than fast signals. The decision on which gates to select is guided by heuristics that look for groups of gates with slow inputs traveling along the longest paths using a least-squares fit of the arrival time versus the delay through the group of gates.

LSS[DJBT81, DBG+84, JTB+86] and Socrates[BCDH86, GBdAH86] also use fan-in ordering. Unlike MIS, however, LSS and Socrates use this technique after mapping the Boolean network into gates.

2.3.2 Gate Collapsing

Another optimization technique collapses together pairs of gates along the critical path[BJ88]. This technique tries to shorten the critical paths, thus reducing the delay of the paths. In LSS, gates are collapsed using a method called path correction. This method also tries to shorten paths, but it allows for duplication of logic which may occur when the first gate of the pair has a fan-out that is greater than one.

2.3.3 Gate Partitioning

Gate partitioning splits up large gates to decrease the delay on a critical path. MTA[HK87] describes two techniques for partitioning gates. In the first technique, signal grouping, the inputs to the gate are divided into a time critical group and a non-time critical group. Inputs arriving within ten percent of the slowest input are considered part of the time critical group. The inputs in the non-time critical group are removed and placed in a new gate which drives the original gate. The net result is that the original gate is now smaller and faster, reducing path delay at a cost of increasing the delay for the inputs from the non-time critical group, since they must pass through an extra stage of logic. However, since the non-time critical inputs arrive much earlier, the extra logic does not adversely affect the speed of the circuit.

Another technique of gate partitioning used in MTA is called binary partitioning. In this method, the gate splits into a multi-tiered network of gates, with either a two-input nand gate or nor gate on the top tier for improved load driving ability. Socrates also looks for large gates on the critical path that can be broken up to reduce delay.

McMAP[LBK88] combines gate partitioning and gate collapsing together, using a loop of partition-collapse operations to optimize the circuit. In McMAP, the partition-collapse operation is performed several times, each with a different random initial assignment. The fastest circuit produced during all the trials is then kept as the final circuit.

YLE[BCD+88, DeM87], which is part of the Yorktown Silicon Compiler system, also combines gate partitioning and collapsing. However, it adds an additional step, gate simplification, between the partition and collapse steps. This transformation is described below. YLE uses a greedy heuristic that selects the slowest gates on the critical path first.

2.3.4 Logic Dual Replacement

In logic dual replacement, one or more gates is replaced with it logic dual. For example, a nor gate is replaced by a nand gate, or an aoi gate is replaced by an oai gate. The motivation for this type of operation is to reduce the number of inverters in the circuit and along the critical path, reducing both area and delay. McMAP uses a greedy interchange algorithm with random initial assignment of the gates and multiple trials. The circuit with the shortest delay from all the trials is then selected. MIS also uses this technique, but in the context of area minimization. Two different heuristics are used. The first heuristic replaces each gate with its dual if it reduces the number of inverters in the circuit. Because the first heuristic is susceptible to getting caught in a local minimum, the second heuristic uses the first heuristic to find a local minimum and then inverts some gates, even if the number of inverters increase, so that later iterations may find a circuit with fewer inverters. Berkelaar uses heuristics that minimize inverters along the critical paths. SKOL separately replaces each gate on the critical path with its dual to see if there is any decrease in delay. MTA also uses a similar technique. LSS also tries to remove inverters on critical paths, but it searches for complements of the inverter's source. If such a signal is found and replacing the inverter with the complemented signal reduces the delay, then the inverter is removed.

Socrates uses this technique from the point of view of replacing nor-nor gate pairs with nand-nand gate pairs when the speed of the nand gates are faster than nor gates in the particular implementation technology, as in the case with CMOS.

2.3.5 Load Matching

Load matching consists of several techniques designed to help a gate drive a large load. In MTA, inverters that drive large load are duplicated, with the load divided between the two inverters. EPOXY[OK88] can either insert a buffer to drive a large load, or duplicate a gate so that the original gate drives only the critical part of the load and the duplicate drives the non-critical part of the load. LSS tries to choose the optimum buffer size based on the size of the load to be driven. Socrates looks for heavily loaded gates and inserts buffers between them and the load.

2.3.6 Gate Simplification

In gate simplification, which is used by YLE, a logic gate is replaced by another logic gate that has the same function but requires less area using Boolean minimization techniques from logic synthesis. The simplification is done using the Boolean function representation of the gate and tries to take advantage of don't care conditions that may exist for the function.

2.3.7 Transistor Ordering

Transistor ordering, a transformation available in both MTA and EPOXY, reorders the transistors within a gate in order to reduce the gate delay due to internal gate parasitics. By moving early arriving inputs closer to the power or ground nodes, the sources and drains of the transistors are set with the proper voltage, allowing the gate to switch faster when the slower inputs finally arrive. MTA and EPOXY use this transformation as a last resort to reduce the delay along a particular path.

Chapter III
Models

This chapter describes the different models that are used to represent the circuit. These models abstract away unimportant details from the problem. The models discussed include: the representation of the circuit, and the models for delay, area, critical path and power dissipation.

3.1 Representation of Circuit Structure

A circuit can be represented as a hypergraph with each node representing a Boolean function or gate, and each edge connecting the output of a node to all other nodes which use the function computed at the node as an input. To slightly simplify this representation, the hypergraph is converted to a normal graph by replacing all hyperedges that connect n nodes together with $n-1$ edges. If the circuit is only combinational logic, then the graph is acyclic. Furthermore, if the nodes represent something where the signal flow is uni-directional, as with a Boolean equation or logic gate, then the edges become directed edges.

The important issue for the representation of the circuit structure is the level of representation of the nodes in the graph used to represent the structure. The first choice is letting each node represent a Boolean equation. This representation, called a Boolean network[BBH+88], is commonly used to represent a circuit during logic decomposition and minimization. The second choice assigns a logic gate to each node in the graph. This second representation is called a net-list.

3.1.1 Boolean Network

The first representation, a Boolean network, is a technology-independent description of the circuit. Although it does not contain some important information that is needed for performance optimization, the description of the transformations are sometimes easier to understand using this representation. Before defining a Boolean network, though, a Boolean function must be defined.

Definition 3.1 (Boolean Function) A Boolean function is a function f: $B^t \rightarrow B$. An *incompletely* specified Boolean function is a set of three completely specified Boolean functions, (f, d, r), where f: $B^t \rightarrow B$, d: $B^t \rightarrow B$, and r: $B^t \rightarrow B$. The function f is called the ***on set*** , d is called the ***don't care set*** , and r is called the ***off set*** . □

When an incompletely specified Boolean function does not have a don't care set, *i.e.* $d = \emptyset$, it is sometimes called a completely specified Boolean function. Such a function can be specified using only the on set or only the off set. A Boolean network is defined in terms of Boolean functions:

Definition 3.2 (Boolean Network) A Boolean network, η, is defined as a pair $(\mathbf{F}, PO)$, where $\mathbf{F} = \{F_i, i = 1, 2, \ldots, m\}$ is a set of m *on sets*, f_i, of incompletely specified Boolean functions. Each f_i is associated with a variable, y_i, which represents the function in the network. The set of variables, y_i, are grouped together in the intermediate variable set, $IV = \{y_1, y_2, \ldots, y_m\}$. PO is the primary output set, $PO \subseteq \{1, 2, \ldots, m\}$. This primary output set contains those intermediate variables, y_i, that are observable as outputs of the Boolean network. Sometimes, the set PO is represented as a vector $\mathbf{z}$, such that $z_i = y_{PO(i)}$, where $i = 1, 2, \ldots, p$ and $p = \mid PO \mid$. □

A Boolean network is called technology independent because the Boolean functions at each node of the network need not correspond to a particular gate in any technology.

Three measures commonly used to evaluate Boolean networks are: primality, irredundancy, and R-minimality. The first two measures, primality and irredundancy, can also be applied to a single Boolean equation. However, here they are considered only in the context of a Boolean network. The two measures are defined as follows:

Definition 3.3 (Prime) A Boolean network, η, is *prime* if for any cube, c, in a function (node) in the network, F_i, no literal of c can be removed without causing the resulting network, η', to be *not* equivalent to η. □

Definition 3.4 (Irredundant) A Boolean network, η, is *irredundant* when no cube, c can be removed from any F_i in η without causing the resulting network η' to be *not* equivalent. □

A function must be considered prime or irredundant with respect to the *entire* Boolean network because a function may be prime or irredundant by itself, but not within a particular Boolean network. This situation is caused by the don't care conditions associated with the particular Boolean network. A function that is prime and irredundant does not have any unnecessary literals or terms, which is desirable for area minimization (fewer transistors), delay minimization (shorter paths in the gate), and power minimization (fewer transistors). Also, it has been shown[BBH+86] that a network that is prime and irredundant is also 100% testable for input and output single stuck-at faults. This result is important since a high degree of testability is an important requirement for circuits.

The final concept is R-minimality. Unlike the primality and irredundancy, R-minimality has meaning only in the context of a Boolean network. Before giving the definition, the reduction and expansion operations on a Boolean network must be defined.

Definition 3.5 (Reduction) Given the function $F_j \in \mathbf{F}$ in the Boolean network $\eta = (\mathbf{F}, PO)$, a cube $c \in F_j$ can be *reduced* to cube c' if $\eta = \eta'$, where η' is defined by replacing F_j with $F_j' = c' \cup (F_j - c)$, and for all $c'' \subset c', \eta \neq \eta''$, where $F_j'' = c'' \cup (F_j - c)$. □

In other words, the reduction operation replaces cube c with a cube c' that is a subset of the original cube and has no smaller subset that can replace the new cube and still preserve the function of the Boolean network.

Definition 3.6 (Expansion) Literal l in cube $c \in F_j$ may be *expanded* to cube c' where $c' = c/l$(cube c with literal l removed), if $c' \cap \overline{F_j} = \emptyset$. □

In other words, literal l in a cube c may be expanded if the expanded term and the complement of the original function have no terms in common. Expansion allows us to remove non-prime literals from the function. Now, R-minimal is defined in terms of these two operations.

Definition 3.7 (R-minimal) A Boolean network η is called *R-minimal* if there does not exist a cube $c_k \in F_j$ of η that when reduced and expanded into cube c_k^+ causes cubes $c_k \subset F_j$ and $c_l \subset F_j$ $(k \neq l)$ to become redundant[BBH+88]. □

In practice, R-minimality occurs when none of the cubes, c, of the functions, F_i, in the Boolean network can be reduced. A cube c of F_i, cannot be reduced when it does not intersect any cube in the union of the cover of F_i, $\mathcal{F}_i$, and the don't care set of F_i, d_i, except itself. Since several equivalent Boolean networks may be prime and irredundant, R-minimality implies a local minimum among equivalent networks with respect to the set of functions in the network, since in an R-minimal network none of the individual functions can be replaced by a combination of one or more other functions in the network. While insuring that a network is R-minimal is useful for minimizing the area of a circuit, it does not indicate an optimal Boolean network as far as delay is concerned. In chapter 4, we will describe transformations that decrease the delay in the circuit, but do not maintain R-minimality.

3.1.2 Gate-level Netlist

Another representation used in the description of the transformations is the gate-level netlist representation of a circuit. This representation corresponds to the actual implementation of the circuit, so unlike a Boolean network, the nodes of the gate-level netlist correspond to actual gates from a particular technology. In a standard cell or gate array implementation, the types of gates in the net-list are restricted to be in a particular set or library.

A gate-level netlist has a clear advantage over a Boolean network for modeling purposes in performance optimization because it contains gates rather than Boolean equations. The delay, area, and power requirements of the gates in the net-list can be more accurately modeled than Boolean equation, allowing more accurate representation of these values in the circuit. If there is some implicit assumption about the implementation of the Boolean functions, then the delay, area, and power can be accurately modeled, but this assumption is actually a trivial technology mapping and so the representation is actually a netlist. Despite the Boolean network's lack of accurate modeling for delay, area and power, it is still useful to describe the action of some transformations with this representation, so both models are used here.

3.2 Delay Model

The delay model allows for the estimate of the speed for the circuit and is used to make decisions on where to reduce the delay. One common model for delay is the unit-delay model. In this model, each gate is given the same delay, and so the delay for a particular path through the circuit is just the number of gates in the path.

Although this method can quickly analyze a circuit, it is too inaccurate. The delay for a gate is dependent upon the size of the gate and the size of the load it must drive and neither of these factors are accurately modeled by this technique. Therefore, a more accurate model is required.

In this work, delay in the circuit is estimated by the RC model[Elm48]. The delay of a gate, d, is the product of all of the resistances and capacitances associated with the gate and the load it must drive:

$$d = \sum R \cdot \sum C \tag{3.1}$$

The delay d_P through a path P in the circuit is given by:

$$d_P = \sum d_i \tag{3.2}$$

where d_i is the delay of gate i and gate $i \in P$. The resistances are the resistance of the transistor that is turning on as well as any parasitic resistances in the gate and load. The capacitances consist of parasitic capacitances within the gate and the load capacitance. A more detailed description of this model is given in Appendix A.

More accurate and complex models could be used, such as those used in device-level simulators, but the evaluation of these complex models requires significantly more CPU time. Since the improvement in accuracy does not outweigh the cost of additional CPU times, the RC model is used.

To simplify the modeling of the gate, a restriction is made so that only one input to the gate changes at any time. While this restriction is not always true, it simplifies the evaluation of gate delay and removes the possibility of function hazards. While it is possible that this restriction can cause large errors in delay for the gate, such as when all the inputs change at the same time, the likelihood of this situation occurring is highly unlikely.

Wires are used to connect the gates in the circuit. These wires, however, contribute parasitic capacitance and resistance to the circuit. Ignoring these parasitics causes some errors, especially at small geometry size because these parasitics become larger in comparison with gate resistances and capacitances. Unfortunately, these parasitics can only be determined from the mask-level description, and so are not usually available when performance optimization is done. One way to make up for this missing information is to estimate the wiring parasitics. However, estimation of these parasitics is complicated because the parasitics can vary widely based on the placement of the gates in the layout and on the implementation, be it standard cells, gate arrays, or custom layout. For this reason, wiring parasitics are ignored, unless they are specifically supplied with the circuit description. If these parasitics are supplied, then they are incorporated into the delay evaluation of the gates.

3.3 Area Model

The area model is used to measure the area required to construct a circuit. Since large integrated circuits are costly to produce, it is important to keep track of the total area used. There are two components to the area of the circuit. The first component is the transistors and the second component is the wires that connect the transistors together. The area required by the transistors depends upon the implementation technology and the relative sizes. For wires, the area depends not only on the implementation technology and relative sizes, but the design style as well. For example, gate array design style has a large wiring area requirement. Standard cells, while not as large as gate array still use a moderate wiring area. Full custom, on the other hand, has the smallest area requirement. There has been some work on estimating wiring area[HMD77, EG81, SP86], however these models assume a particular implementation, usually gate arrays. Rather than making an assumption on the particular design style, we opt for modeling the area just by using the transistors.

There are three principal methods for measuring the area of the circuit using only the transistors. The first method, gate count, is popular for modeling area in logic synthesis. However, this model is undesirable because it underestimates the area of the circuits because it treats all gates the same. Thus, large complex gates are considered to be the same size as inverters.

Another technique used to model circuit area is the sum of transistors sizes or the sum of the widths of transistors. This technique, which makes sense only when differing transistors sizes are allowed, tends to overestimate the relative area of the circuit because it implicitly assumes that the wiring area of the transistor is directly proportional to transistor sizes, which is not correct. There are significant wiring sections of a circuit that are independent of the transistor size.

The technique used here to measure area is the total number of transistors. It does not treat all gates with equal size and also does not overestimate relative area when the sizes of transistors change. Although the actual area is highly dependent on the mask implementation, this model provides a reasonable approximation.

3.4 Power Model

Only a limited amount of power may be safely dissipated from a chip. Thus, it is important to keep track of the total power released. Since the circuits analyzed are static CMOS circuits, there are three sources of power dissipation. The first source is static power dissipation caused by substrate leakage current, and the second source is dynamic power dissipation caused by switching states. The static power dissipation component is very small, however, and is ignored. The other two components are the dynamic power, used to charge or discharge a node when changing state, and the short-circuit power, which occurs when there is a path between V_{dd} and GND.

The dynamic power dissipation, P_d, is modeled for each stage using the following equation:

$$P_d = C_L V_{dd}^2 f_p \tag{3.3}$$

where C_L is the load capacitance of the gate, V_{dd} is the supply voltage, and f_p is the frequency of operation. The short-circuit power is given by:

$$P_s = \frac{\beta}{12}(V_{dd})^3 \frac{\tau}{T} \tag{3.4}$$

where τ is the rise or fall time of the gate, and T is the reciprocal of the frequency of operation[Vee84].

In limiting the scope of this research, the effects on power dissipation are not considered in the implementation and experiments. However, they are considered in the discussion of the transformations in Chapter 4.

3.5 Critical Path Model

The signals that pass through the circuit require different amounts of time to propagate through to the outputs. The amount of time required may be more than what the designer has intended. When there exists paths through the circuit that do not meet the design requirements, they are called *critical paths.* Before giving the formal definition, some terms need to be defined:

Definition 3.8 (Ready Time) The ready time of a gate g, $t(g)$, is the time when the output of gate g is available for input to other gates. It is the sum of the arrival time of the slowest input to gate g and the delay associated with gate g. □

Definition 3.9 (Required Time) The required time of a gate g, $\hat{t}(g)$, is the time when the output gate g needs to be available for input to other gates in order to meet the timing constraints for the circuit. □

These first two definitions form the basis for gate slack[DeM87]:

Definition 3.10 (Slack) The slack time of gate g, $s(g)$ is the maximum amount of leeway between the ready time and the required time for the gate. If gate g drives a primary output, then the slack is the difference between the required time and the ready time: $s(g) = \hat{t}(g) - t(g)$. If gate g does not drive a primary output, then $s(g) = min_{j \in J}[s(j) + max_{k \in K} t(k) - t(g)]$ where J is the set of all gates driven by gate g and K is the set of gates that drive gate $j \in J$.

Now, the definition a critical path in the circuit, in terms of slack, is as follows:

Definition 3.11 (Critical Path) A critical path is a path through the circuit from a primary input to a primary output that does not meet the timing constraints set by the designer. The gates on this critical path have a negative value for slack and are sometimes called *critical gates.* □

The critical paths are important, because these are the paths in the circuit that are transformed, using techniques in chapter 4, to reduce delays.

When determining critical paths, it is assumed that only one primary input to the critical path changes at a time. The critical paths then measure the delay from the primary input to the primary output assuming all other primary inputs do not change.

3.5.1 False Paths

The above definition for critical paths is called data-independent because the critical paths are found without checking every input pattern. This method has one drawback though. It is possible to identify paths in the circuit which appear to be critical paths but, in fact are not because the worst-case conditions responsible for the path wil never occur. This type of critical path is called a *false path.* Another problem that may occur is that paths associated with race conditions in reconvergent fan-out will not be detected.

To remove false paths and find undetected paths, a complete simulation using all possible input values and using a detailed circuit simulator is necessary. However, this approach is extremely expensive because of the large run times needed and is not feasible for very large circuits.

Some recent work by McGreer [MB89] has considered the problem of finding the true critical paths in the circuit and provides a more efficient solution to this problem. However, there is still a large cost in run time associated with this technique. Because of these costs, our approach here is to assume that the paths determined using static timing analysis are correct. Fortunately, McGreer found that all but one of the circuits in the MCNC Benchmark set [LBK88], which are used in the experiments in Chapter 6, do not contain false paths.

Chapter IV
Transformations

This chapter describes the different types of transformations that are used in performance optimization. The transformations are divided into two groups: local and global. If the circuit is thought of as a graph, then a tree is defined as a set of connected gates in the circuit that, except for the root, have fan-out of one. Then local transformations, with one exception, are applied only to individual trees. Their effect is only on a small number of gates. On the other hand, global transformations involve two or more trees in the network, and therefore affect a larger number of gates. Because of this difference, local transformations usually have smaller effects on the delay of the circuit, as well as the area of the circuit, than global transformations. This difference will become apparent in the following sections.

The description of each transformation uses some abbreviations which are divided into two groups: those for delay and those for power. The discussion of delay uses the following abbreviations:

R	=	the resistance for a minimum size transistor
C	=	the gate capacitance for a minimum size transistor
k_i	=	the transistor scaling factor for gate i, that is, each transistor in gate i has size k_i times a minimum size transistor
k_i'	=	the transistor scaling factor in the transformed circuit for gate i that is, each transistor in gate i has size k_i times a minimum size transistor
$t_r(v_i)$	=	the data ready time for gate v_i
$t_r(v_i)'$	=	the data ready time for gate v_i after it has been transformed
$t_i(v_i)$	=	the ready time for the input to gate v_i
$d(v_i)$	=	the delay due to gate v_i
$\Delta t_r(v_i)$	=	$t_r(v_i) - t_r(v_i)'$, the change in the data ready time caused by the transformation

Whenever C has a subscript, it refers to a particular capacitance value and not the gate capacitance of a minimum sized transistor. The equations for delay do not explicitly include parasitic capacitance and resistance values. However, if these values are available, they can be easily included in the equations.

The discussion of power uses the following abbreviations:

C_L	=	the load capacitance
V_{dd}	=	the supply voltage
f_p	=	the frequency of operation
β	=	MOS gain factor
V_T	=	the threshold voltage of a transistor
T	=	$\frac{1}{f_p}$
τ	=	the rise or fall time of a gate
P'	=	the power use after the transformation has been done
ΔP	=	$P - P'$, the change in power use caused by the transformation

In the discussion of dynamic power dissipation, the frequency of operation, f_p, is used. The transformations will speed up the section of the circuit that is changed by the transformation. However, in the context of the entire circuit, f_p may not increase, or the increase will be less than the improvement of the transformation because of other critical paths. Therefore, predicting the actual change in f_p and hence the dynamic power dissipation becomes very difficult. Using the change in frequency implied by the transformation would overestimate the actual change in dynamic power dissipation. Therefore, in the discussion below, f_p is assumed, for simplicity, to be a constant for some of the analysis. So, the equations in the discussion indicate a lower bound on the change in dynamic power dissipation.

4.1 Local Transformations

4.1.1 Buffering

Transistor sizing is a commonly-used technique for allowing gates to drive capacitive loads. This technique, however, is not sufficient if the capacitive load is too large. By inserting one or more buffers between the gate and the capacitive load, a reduction in the path delay can occur. This insertion process is called the buffering transformation.

If the buffering operation is considered at the Boolean network level, then it adds Boolean equations into the network. So applying the transformation to a circuit revokes the R-minimal property. This change occurs because the insertion of a buffer does not contribute to the computation of the function, that is, it can be removed without changing the functions of the Boolean network. However, as will be shown below, the delay of the circuit can be reduced when this transformation is applied.

Example

Consider the following simple example, where a nand gate drives a large capacitive load, C_L, shown on the left side of Figure 4.1. Based on the ratio of input to output capacitance of the gate, it is determined through a method described below, that in order to decrease the delay along the path, a buffer is needed between the gate and the load that it drives. The transformed circuit is shown on the right side of Figure 4.1.

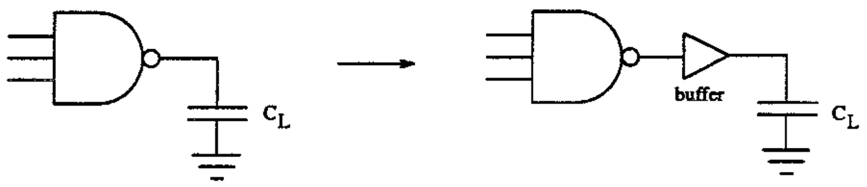


Figure 4.1: Buffering Transformation Example

This transformation takes advantage of the analog properties of the MOS transistor. Since there is a large capacitive load, the buffer is sized larger than the gate originally driving the load. The buffer acts as an amplifier by providing more current and thus charges the capacitive load faster than the original gate. The original gate now has a reduced load to drive, and so it operates faster. Although the buffer adds delay to the signal path, the increase is more than offset by the decrease in delay for the original gate, so the net result is a decrease in path delay.

Method

The first step of this transformation is to determine the optimal number of stages for the load that must be driven. The formula to determine the optimal number of stages, S, to drive a capacitive load comes from Lee[LS84] and is:

$$S = \ln\left(\frac{C_L}{C_i}\prod_{n=1}^{N} R_n\right) \tag{4.1}$$

where N is the number of stages in the current path, C_L is the capacitive load to be driven, C_i is the input capacitance of the first gate in the chain of gates that drive C_L, and R_n is the resistance of the longest series chain for gate n. Notice that adding inverters to the path does not change the value of $\prod_{n=1}^{N} R_n$. This formula arises from solving the partial derivative of the equation for path delay with respect to the number of stages, and assumes that the parasitic capacitances are small in comparison to the gate capacitances. If this assumption is not true, the optimal number of stages cannot be computed analytically. For the technology that is used here, however, the assumption is true.

The number of stages driving the load is determined by counting back along the critical path up to but not including the next gate with fan-out of greater than one. This is the number of stages in the worst case path of the current tree in the Boolean network representation. This number of stages is then compared with the optimal number from Equation 4.1 to determine whether or not additional stages are required.

If more stages are required than exist, additional stages are added between the driving gate and the capacitive load. Since inverters are used as the additional stages, a phase shift, that is the replacement of a gate by its dual, may be required in order to get the proper number of stages. The action taken is summarized below in Table 4.1. The use of phase shift requires that the inverted signal of the inputs to the tree is also available in the circuit.

Stages	Operation
1	no change (transistor sizing only)
2	"phase shift" and add inverter after driving gate
3	add two inverters after driving gate
4	"phase shift" and add two inverters after driving gate
5	add three inverters after driving gate

Table 4.1: Buffer Transformation Rules

Speed Changes

Consider the example in Figure 4.1. Generalizing that example to an r-input nand gate (and introducing an r factor into the equations), the effect of the transformation on the speed of the gate driving the large load and the data ready time for the final stage to drive the large load is described by the following equations.

$$\begin{aligned} t_r(g) &= t_i(g) + \frac{r}{k} RC_L & (4.2) \\ t_r(g_{\text{buffer}}) &= t_i(g) + \frac{2r}{k'} k_{\text{buffer}} RC + \frac{1}{k_{\text{buffer}}} RC_L & (4.3) \\ \Delta t_r(g) &= \frac{2r}{k} RC_L - (\frac{r}{k'} k_{\text{buffer}} RC + \frac{1}{k_{\text{buffer}}} RC_L) & (4.4) \end{aligned}$$

If we assume that the transistors are optimally sized, then we know that the ratio of transistor sizes for two consecutive stages is: $\frac{k_{i+1}}{k_i} = \alpha$. So if we add n stages to the circuit, the data ready time at the output of the last buffer, $t_r(g_{\text{buffer}})$, is described by:

$$t_r(g_{\text{buffer}}) = t_i(g) + 2r\alpha RC + \sum_{i=1}^{n-1} 2\alpha RC + \frac{1}{\alpha^n k_{\text{buffer}}} RC_L \qquad (4.5)$$

Area Changes

The change in area depends on the number of inverters that are required to buffer the capacitive load. Each inverter adds two transistors to the area of the circuit.

Power Dissipation Changes

The change in dynamic power dissipation depends on two factors. First, the number of additional stages added, and second, the change in transistor sizes for the gate. Generalizing to the case of an r-input gate, the equation for dynamic power dissipation is as follows:

$$\begin{aligned} P_d &= 2rkCV_{dd}^2 f_p + C_L V_{dd}^2 f_p & (4.6) \\ P_d' &= 2rkCV_{dd}^2 f_p + \sum_{i=1}^{n} 2\alpha^i kCV_{dd}^2 f_p + C_L V_{dd}^2 f_p & (4.7) \\ \Delta P_d &= 2r\Delta kCV_{dd}^2 f_p + \sum_{i=1}^{n} 2\alpha^i kCV_{dd}^2 f_p & (4.8) \end{aligned}$$

where α is the transistor scaling factor and n is the total number of additional buffer stages in the transformed circuit.

The change in the short circuit power dissipation is proportional to the decrease in the rise/fall time of the original gate plus the rise/fall time for each of the inverters in the buffer, assuming the frequency of operation remains the same. The decrease is then given by:

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3[(\sum_{i=1}^{n} \frac{\tau_{b_i}}{T}) - \frac{\Delta\tau}{T}] \tag{4.9}$$

where $\Delta\tau$ is the change in the rise/fall time for the original gate and τ_{b_i} are the rise/fall times for the additional stages added to the circuit.

4.1.2 Critical Signal Isolation

This transformation's goal is to reduce the delay by reducing the load that a gate must drive by isolating the gate's critical load from its non-critical load. An isolation buffer is inserted between the gate and the non-critical load, reducing the total load that the gate drives. As with the buffering transformation, this transformation does not maintain R-minimality, although it does preserve primality and irredundancy.

Example

To illustrate this transformation, consider a simple example. The circuit on the left side of Figure 4.2 contains a nand gate on the critical path driving a load on the critical path and a load that is not on the critical path. These loads represent other gates in the circuit. The transformation adds an isolation buffer, implemented as two minimum-sized inverters, between the first gate and the non-critical load, shown on the right side of Figure 4.2. The capacitive load that the gate must drive is reduced because the buffer is minimum sized. Also, the gate's transistor sizes may be reduced because of the smaller load, resulting in an increase in speed for the path and a decrease in power consumption for the gate.

Method

The first step in this transformation is to separate the non-critical signals from the critical signals. First, all the gates driven by the gate to be transformed that have slack time less than zero are placed into the critical group. In the next step, an isolation buffer is placed between the original gate and the non-critical load. Then the gates in the non-critical load are re-evaluated to make sure that they are still non-critical. If a gate has become critical, it is removed from the non-critical load

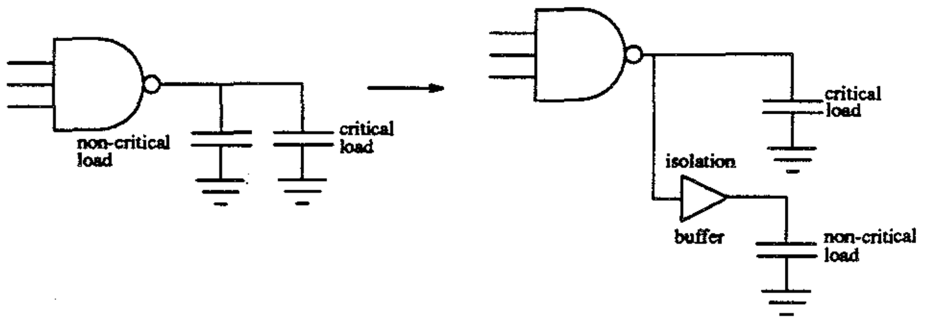


Figure 4.2: Circuit for Critical Signal Isolation

and placed in the critical load, and the remaining gates that are in the non-critical group are re-evaluated. This evaluation process continues until all the gates in the non-critical load have been verified as non-critical. This process is guaranteed to terminate because once a gate is place in the critical group it cannot be moved back into the non-critical group.

The algorithm, described in pseudo-code, for this transformation is shown in Figure 4.3. The threshold value in the pseudo-code description is set to be slightly greater than zero, so those gates that are nearly critical are also put in the critical group.

```
for each gate in load do
    if slack < threshold then
       put gate into critical group
    else
       put gate into non-critical group
end for
add isolation buffer between original gate and non-critical group
repeat
    compute new slack time for non-critical gates
    order non-critical gates by increasing slack time
    if slack time of slowest gate < threshold value then
       move slowest gate to critical group
    else
       exit loop
until all gates in critical group
```

Figure 4.3: Pseudo-code for Critical Signal Isolation

An alternate method used in earlier work duplicates the gate and then has the original gate drive the critical load and the duplicate gate drive the non-critical load. This technique is not as effective because it adds more area, unless the duplicate gate is a two-input nand or nor gate, in which case it adds the same area. The reduction in delay of the alternate method is slightly better, but only if the resistance of each gate driving the duplicated gate is less than the duplicate gate. Also, this method can slow down other critical paths that contains one of the gates that drives the duplicated gate. For this reason, this alternative technique is not used here.

Speed Changes

The effect of this transformation is to speed up the gate by reducing the capacitive load it must drive. Two equations describe the data ready time of the gate before and after the transformation, as shown in the above example. These equations assume that the gate has r inputs, rather than only two inputs. The remaining equations describe the change in speed as a result of applying the transformation:

$$t_r(g) = t_i(g) + \frac{r}{k}R(C_{\text{critical}} + C_{\text{non-critical}}) \tag{4.10}$$

$$t_r(g)' = t_i(g) + \frac{r}{k'}R(C_{\text{critical}} + C_{\text{inv}}) \tag{4.11}$$

$$\Delta t_r(g) = \frac{r}{\Delta k}RC_{\text{critical}} + rR(\frac{1}{k}C_{\text{non-critical}} - \frac{1}{k'}C_{\text{inv}}) \tag{4.12}$$

Assuming that the inverter in the buffer has minimum sized transistors, the overall load decreases and so the new sizes of the transistors in gate g, k', are smaller. If we ignore the change in the size of transistors in gate g (let $k = k'$), the equation simplifies to:

$$\Delta d(g) = \frac{r}{k}R(C_{\text{non-critical}} - C_{\text{inv}}) \tag{4.13}$$

Note that the data ready time for the signal that must pass through the isolation buffer to the non-critical load is now greater due to the additional stage delay of the buffer. However, since the load is not on the critical path, this slowdown does not adversely effect the circuit speed.

Area Changes

One driver, the isolation driver, is added to the circuit. If this driver is implemented by a pair of inverters, then there are two additional gates or four transistors added to the circuit description.

Power Dissipation Changes

Using the example shown in Figure 4.2, above, but generalizing to the case where the gate has r inputs, the following equations describe the change in power dissipation. Note that optimum transistor sizes are assumed. Because there is a reduced load on the original r-input gate, gate g, after the transformation takes place, its transistors can be smaller, and the resulting loads on the gates that drive the gate g are reduced.

The equations that describe the dynamic power dissipation are as follows:

$$\begin{aligned} P_d &= r(2kCV_{dd}^2 f_p) + C_{\text{critical}} V_{dd}^2 f_p + C_{\text{non-critical}} V_{dd}^2 f_p && (4.14) \\ P_d' &= r(2k'CV_{dd}^2 f_p) + C_{\text{critical}} V_{dd}^2 f_p + 2C_{\text{inv}} V_{dd}^2 f_p + C_{\text{non-critical}} V_{dd}^2 f_p && (4.15) \end{aligned}$$

If the sum of capacitive load of the two inverters in the isolation buffer and the reduced gate capacitance of gate g is less than the sum of the original non-critical load and the original gate capacitance of gate g, then there will be a decrease in the dynamic power dissipated by this part of the circuit. The equation is as follows:

$$\Delta P_d = 2r\Delta kCV_{dd}^2 f_p + C_{\text{non-critical}} V_{dd}^2 f_p - 2C_{\text{inv}} V_{dd}^2 f_p \tag{4.16}$$

If, for simplicity, the term $V_{dd}^2 f_p$ is treated as a constant, then the term $2(r\Delta kC + C_{\text{non-critical}} - 2C_{\text{inv}})$ shows, to a constant factor, the change in dynamic power dissipation.

The change in short circuit power dissipation is given by

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3 \left(\frac{\tau_{b1} + \tau_{b2} - \Delta\tau}{T}\right) \tag{4.17}$$

So, the term $\tau_{b1} + \tau_{b2} - \Delta\tau$ shows, to a constant factor, the change in short circuit power dissipation, where τ_{b1} and τ_{b2} are the rise/fall times of the inverter pair added to form the isolation buffer. Usually this value is positive, so there is a small increase in the short circuit power dissipation.

4.1.3 Gate Collapsing

The purpose of the gate collapsing transformation is to reduce the delay along a critical path by reducing the number of stages. This goal is accomplished by collapsing two gates together forming a larger, single gate.

Example

In the circuit, shown on the left side of Figure 4.4, a gate drives several gates, including one on the critical path. The two gates are collapsed together, shown on right side of Figure 4.4. Notice that since the original driving gate also drives gates elsewhere in the circuit, the intermediate node cannot be removed, so there can a slight increase in the area of the circuit. In this example, the transistor count remains the same because the increase in area from the collapse operation is offset by the removal of the inverter.

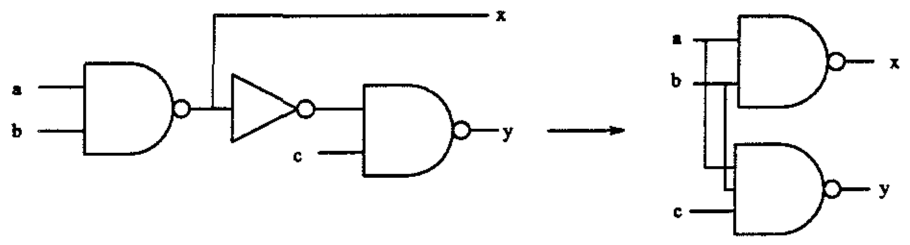


Figure 4.4: Gate Collapsing Without Node Elimination

Method

The first step in this transformation is to select two gates on the critical path to be collapsed together. If there is an intervening inverter, then the gates may be collapsed without worrying about phase of the inputs. However, if the gates are adjacent, then the polarity of the inputs of one of the gate must be inverted because negative logic is used. The gate which requires fewer inverters is chosen to be replaced by its dual. Then the gates can be combined in a straightforward manner. There is a limit on the size of the gates to be collapsed, otherwise the combined gate will not reduce the delay on the critical path. In most cases, replacing the pair of gates with the collapsed gate will actually increase the delay for the gate. However, when the load that the first gate drives is large enough because it drives a large fan-out, the critical path will decrease in delay because the transformation will by-pass the capacitive load. Care must be taken when applying this transformation because it may increase the delay along other, more critical paths because it increases the load on the gates driving the collapsed gate. To prevent this situation from occurring, the transformation is only applied when the gates to be collapsed are the most critical gates driven the by all of the collapsed gates' inputs. The pseudo-code for this transformation is shown in

Figure 4.5.

```
look for pairs of gates straddling tree boundaries where the first gate
drives a large fan-out
if there is no intervening inverter {
  determine which gate's dual requires fewest inverters
  replace that gate with its dual
}
combine gates together
make sure path delay decreases
```

Figure 4.5: Pseudo-code for Gate Collapsing Without Node Elimination

The implementation technology may limit the gates that may be combined. With full complex gate CMOS, there is no restriction, but with standard cells, the gate that is formed by collapsing the two gates together must be part of the standard cell library. Otherwise, the collapse may not take place.

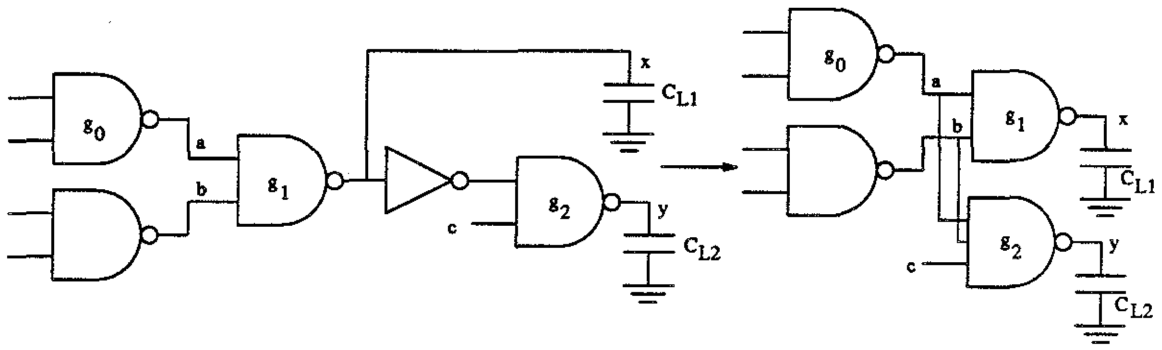


Figure 4.6: Gate Collapsing Example for Delay Analysis

Speed Changes

Let us consider the circuit shown in Figure 4.6. This is the same example as shown in Figure 4.4, but now the gates that drive the first nand gate are shown and the load at the output nodes are explicitly shown (they are needed to determine the change in delay). In this example, the path through the nand gates and inverter is the critical path. Assuming the first nand gate to be collapsed, g_1, has m inputs and the second nand gate to be collapsed, g_2 has $n+1$ inputs, and letting k_i be the gate scaling factor for gate g_i, the change in delay can be seen from the equations for the data ready time:

$$t_r(g_1) = t_i(g_0) + \frac{2k_1}{k_0}RC + \frac{m}{k_1}R(C_{L1} + C_{inv}) \tag{4.18}$$

$$t_i(g_2) = t_i(g_0) + \frac{2k_1}{k_0}RC + \frac{m}{k_1}R(C_{L1} + C_{inv}) + \frac{2k_2}{k_{inv}}RC \tag{4.19}$$

$$t_r(g_2) = t_i(g_2) + \frac{n+1}{k_2}RC_{L2} \tag{4.20}$$

Assuming, $t_i(g_2)' = t_i(g_1)$, that is, the critical path is still the path from gate g_0 through gate g_2 after the transformation, then

$$t_r(g_2)' = t_i(g_2)' + \frac{m+n}{k_2}RC_{L2} \tag{4.21}$$

$$\Delta t_r(g_2) = \frac{m}{k_1}R(C_{L1} + C_{inv}) + \frac{2k_2}{k_{inv}}RC - \frac{2k_2}{k_0}RC - \frac{m-1}{k_2}RC_{L2} \tag{4.22}$$

Assuming $k_{inv} = 1$ and $C_{inv} = 2C$, since the inverter drives only a single nand gate, the equation for $\Delta t_r(g_2)$ simplifies to:

$$\Delta t_r(g_2) = \frac{m}{k_1}RC_{L1} + (\frac{2m}{k_1} + 2k_2 - \frac{2k_2}{k_0})RC - \frac{m-1}{k_2}RC_{L2} \tag{4.23}$$

So the larger the value of C_{L1}, the greater the speed-up possible. Solving for C_{L1} results in an equation that shows when the delay of the path is reduced, in terms of the other factors:

$$C_{L1} > \frac{k_1}{2mR}((\frac{2m}{k_1} + 2k_2 - \frac{2k_2}{k_0})RC - \frac{m-1}{k_2}RC_{L2}) \tag{4.24}$$

Area Changes

The area increase depends upon whether or not there is an intervening inverter that can be removed. If there is no intervening inverter, and there are m inputs to the first gate of the pair, then there is an increase of at least $2(m-1)$ transistors in the circuit description, assuming no inverters have to be added. If additional inverters are needed, then two transistors are added for each additional inverter. If there is an intervening inverter, and it can be removed, then only $2(m-2)$ transistors are added to the circuit description.

Power Dissipation Changes

The change in dynamic power dissipation is illustrated using the general case shown in Figure 4.6. The dynamic power dissipation is described in the following two equations:

$$P_d = 2mk_1CV_{dd}^2f_p + 2k_{inv}CV_{dd}^2f_p + 2(n+1)k_2CV_{dd}^2f_p \tag{4.25}$$
$$P'_d = 2mk'_1CV_{dd}^2f_p + 2(m+n)k'_2CV_{dd}^2f_p \tag{4.26}$$

where P is the dynamic power dissipation on the left side of Figure 4.6 and P' is the dynamic power dissipation on the right side of Figure 4.6. The capacitive loads on the outputs are not included since they do not change with the application of the transformation. Notice that $k'_2 = k_2$ since the gate drives the same load. Depending on the change in load driven by the m-input gate, k'_1 may equal k_1. Assuming that $k'_1 = k_1$, then the change in power dissipation is described by the following:

$$\Delta P_d = P'_d - P_d \tag{4.27}$$
$$= 2((m-1)k_2 - k_{inv}))CV_{dd}^2f_p \tag{4.28}$$

so the term $(m-1)k_2 - k_1$ will give the dynamic power dissipation within a constant factor. If $k'_1 \neq k_1$, then

$$\Delta P = 2(m(\Delta k_1) + (m-1)k_2 - k_{inv})CV_{dd}^2f_p \tag{4.29}$$

and the term $m\Delta k_1 + (m-1)k_2 - k_{inv}$ will give the dynamic power dissipation within a constant factor. The change in short circuit power dissipation is given by:

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3(\frac{\Delta\tau_{g0}}{T} + \frac{\Delta\tau_{g1}}{T} + \frac{\Delta\tau_{g2}}{T} - \frac{\tau_{inv}}{T}) \tag{4.30}$$

where $\Delta\tau_{g_i}$ is the change in the rise/fall times for gate g_i. If there is no intervening inverter, the the last term, $\frac{\tau_{inv}}{T}$ is zero.

4.1.4 Gate Collapsing With Node Elimination

This transformation's goal is to reduce the length of a path through the circuit and the area of the circuit. By combining two gates together, the length of the path can be decreased. This transformation is usually applied *off* of the critical path to save area in the circuit without effecting the overall speed of the circuit.

Example

Consider the circuit shown on the left side of Figure 4.7, below. In this circuit, a two-input nand gate is a predecessor gate that drives another two-input nand gate, and both gates are *not* on the critical path. Since both gates are small, the transformation collapses the two gates together and reduces the delay, as shown in the left side of Figure 4.7. If the two gates were larger, then collapsing them together would not reduce the delay, but it would still save area.

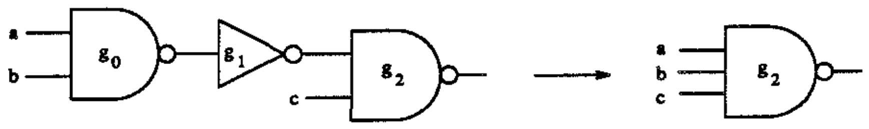


Figure 4.7: Gate Collapsing With Node Elimination

Method

The first step in this transformation is to identify gates with large slack time. Since one effect of this transformation is to slow down part of the circuit, it is necessary to choose gates that will not become critical after applying the transformation. Once these gates have been identified, they are further reduced by selecting those gates that have at least one direct predecessor gate that drives the selected gate. Then for each gate to be collapsed, its predecessors that only drives the gate to be collapsed are sorted by increasing size. Then a predecessor gate is selected and combined with the gate. If the new gate still has a large slack time, then the combination step may be repeated. Finally, the predecessor gates that were merged into the selected gate are removed from the circuit.

There are two alternative methods for the ordering used to attempt to add the predecessor gates. The first method, used above, starts with the smallest predecessor and proceeds to the larger ones in order of increasing size, while the second method starts with the largest predecessor and proceeds in order of decreasing size. Although the best method depends on the relative sizes, the number of predecessors, and the slack time of the gate, the first method is used because it is more likely to be successful. The pseudo-code describing the operation of this transformation is shown in Figure 4.8.

The implementation technology may limit the gates that may be combined. With

```
select gates with large slack time that have at least one direct predecessor gate
that drives only the selected gate
for each gate do
    sort the direct predecessors that drive only the selected gate in order of increasing
    (decreasing) size
end for
for each gate do
    for $i = 1$ to number of sorted direct predecessor gates do
        combine direct predecessor gate $i$ with selected gate
        compute slack time for new combined gate
        if slack time < threshold value then
            remove direct predecessor gate
            replace selected gate with new gate
        else
            exit loop
        end if
    end for
end for
```

Figure 4.8: Pseudo-code for Gate Collapsing With Node Elimination

full complex gate CMOS, there is no restriction, but with standard cells, the gate that is formed by collapsing the two gates together must be part of the standard cell library. Otherwise, the collapse may not take place.

Speed Changes

Consider the circuit shown in Figure 4.7. Assuming the critical path is through the chain of gates and is not input c, the equations for delay in the two circuits, and the change in delay are as follows:

$$t_i(g_2) = t_i(g_0) + \frac{2mk_1}{k_0}RC + \frac{2k_2}{k_1}RC \quad (4.31)$$

$$t_r(g_2) = t_i(g_2) + \frac{n+1}{k_2}RC_L \quad (4.32)$$

$$t_i(g_2') = \max(t_i(g_0), t_{i\ other}(g_2)) \quad (4.33)$$

$$t_r(g_2') = t_i(g_2') + \frac{n+m}{k_2}RC_L \quad (4.34)$$

$$\Delta t_r(g_2) = \Delta t_i(g_2) + (1-m)RC_L \quad (4.35)$$

where m is the number of inputs to predecessor gate, n is the number of additional inputs to selected gate (for a total of $n+1$ inputs to the selected gate), C_{L1} is the capacitive load for predecessor gate, C_L is the capacitive load for selected gate, and $t_{i\ other}$ is the arrival time of the other inputs to the second gate (c in Figure 4.7). Notice that k_2 does not change because the load C_L remains unchanged.

The value of $\Delta t_i(g_2)$ can take on three possible values, depending on the values of $t_i(g_0)$ and $t_{i\ other}(g_2)$:

1. When $t_i(g_0) + \frac{2mk_1}{k_0}RC_L + \frac{2k_2}{k_1}RC > t_{i\ other}(g_2)$ but $t_{i\ other}(g_2) > t_i(g_0)$, then $t_i(g_0)$ limits the speed of the original gates, but not the new gate. The change in data ready time then simplifies to:

$$\Delta t_r(g_2) = t_i(g_0) + \frac{2mk_0}{k_1}RC + \frac{2k_2}{k_1}RC - t_{i\ other}(g_2) + \frac{(1-m)}{k_2}RC_L \tag{4.36}$$

2. When $t_i(g_0) > t_{i\ other}(g_2)$, then $t_i(g_0)$ limits the speed of the gates before and after the transformation, so the change in data ready time becomes:

$$\Delta t_r(g_2) = \frac{2mk_0}{k_1}RC + \frac{2k_1}{k_2}RC + \frac{(1-m)}{k_2}RC_L \tag{4.37}$$

3. When $t_{i\ other}(g_2) > t_i(g_0)$ but $t_i(g_0) + mR(C_{L1} + \frac{2}{k_1}C) + \frac{2k_2}{k_1}RC > t_{i\ other}(g_2)$ then $t_{i\ other}(g_2)$ limits the circuit speed only after the transformation has been applied. Here, the data ready time for the circuit is:

$$\Delta t_r(g_2) = \frac{(1-m)}{k_2}RC_L \tag{4.38}$$

Area Changes

The area savings depends upon whether or not there is an intervening inverter. If there is no intervening inverter between the two gates, then there is a reduction of two transistors from the circuit description. If an inverter is also removed, then two additional transistors are removed from the circuit description.

Power Dissipation Changes

The savings in dynamic power dissipation is as follows. Using the general case for the transformation, the equations to describe the two figures are:

$$\begin{aligned} P_d &= 2mk_0CV_{dd}^2f_p + 2k_1CV_{dd}^2f_p + 2(n+1)k_2CV_{dd}^2f_p && (4.39) \\ P'_d &= 2(m+n)k'_2CV_{dd}^2f_p && (4.40) \end{aligned}$$

where P_d is the dynamic power dissipation for the circuit on the left side of Figure 4.7 and P_d' is the dynamic power dissipation for the circuit on right side of Figure 4.7. Assuming that each group of gates must drive the same capacitive load, then $k_2 = k_2'$. Thus, the change in dynamic power dissipation would be

$$\Delta P_d = P_d - P_d' \tag{4.41}$$

$$\Delta P_d = (2mk_0 + 2k_1 + (2n+2)k_2 - 2(m+n)k_2)CV_{dd}^2 f_p \tag{4.42}$$

$$= 2(mk_0 + k_1 - (m-1)k_2)CV_{dd}^2 f_p \tag{4.43}$$

If, for simplicity, we assume that the term $CV_{dd}^2 f_p$ is a constant, then the term $mk_0 + k_1 - (m-1)k_2$ will show to a constant factor the change in dynamic power dissipation. The change in short circuit power dissipation is given by:

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3[\frac{\tau_{new}}{T} - (\frac{\tau_1}{T} + \frac{\tau_2}{T} + \frac{\tau_{inv}}{T})] \tag{4.44}$$

where τ_{new} is the rise/fall time of the collapsed gate, and τ_1 and τ_2 are the rise/fall times for the original pair of gates. If there is no intervening inverter, then the term, $\frac{\tau_{inv}}{T}$ is zero.

4.1.5 Gate Decomposition (Non-Uniform Arrival Time)

In the two previous sections, gate collapsing was discussed. The next two sections discuss the inverse operation, that is, the splitting of large, slow gates to reduce delays along the critical path. This section describes the case when the arrival time of the inputs are not the same, while the next section describes the case when the inputs to the gate arrive at nearly the same time.

Example

First, consider the case where one of the inputs to the gate is much slower than the remaining inputs. On the left side of Figure 4.9, below, a nand gate has one slow input, marked "critical input", and the remaining inputs are fast in comparison. To speed up this gate, it is split into two gates, with the fast inputs removed from the original gate, and an inverter inserted between the two nand gates to keep the signal polarity correct. The transformed circuit is shown on the right side of Figure 4.9. The fast inputs now must go through additional stages which slow them down, but the inputs are still fast enough that they are available before the critical input and so the overall delay of the gate decreases because the gate's resistance is now much smaller than before.

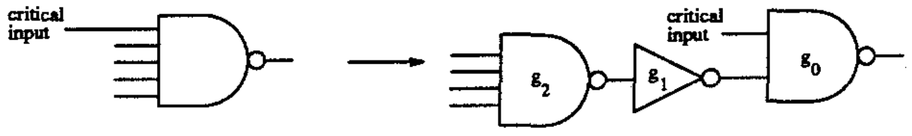


Figure 4.9: Gate Decomposition (Non-uniform Input Arrival Time) Example

If there is more than one slow input, transformation works in the same way. As long as there are at least two fast inputs in the original gate, all fast inputs are removed from the original gate and placed in a new gate. If only one fast input exists, a new gate cannot be formed.

When the gate with non-uniform input arrival times is a complex CMOS gate, such as the gate shown on the left side of Figure 4.10, it may not be possible to remove some or all of the fast inputs of the gate in some circumstances. For example, let inputs *a* and *b* be slow inputs in Figure 4.10. Then, the fast inputs *d* and *e* are removed, but *c* cannot be removed. In fact, with some complex gate configurations it may not be possible to remove any of the fast inputs. In such a case, the gate may need to be factored in a different fashion. Techniques for factoring are discussed further in Section 4.1.8.

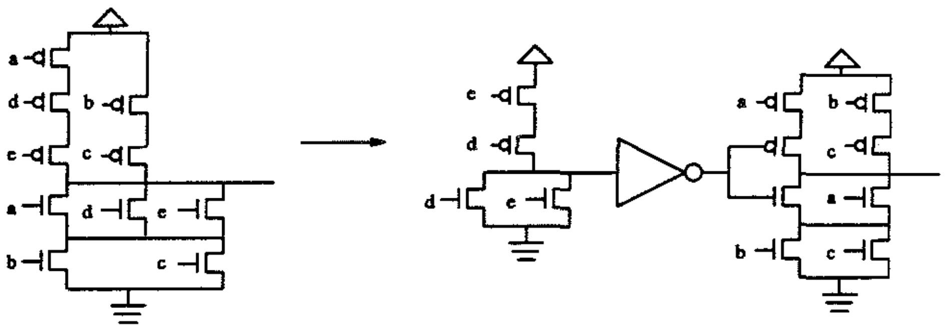


Figure 4.10: Complex Gate for Gate Decomposition

Method

Gate decomposition uses general factoring methods, such as those available in logic synthesis tools such as MisII[BRSVW87]. However, the heuristics are slightly modified to take into account the difference in arrival times so that the critical inputs are separated from the non-critical inputs. The pseudo-code for the method is shown in Figure 4.11.

```
gate_decomposition(gate g, arrival_time_list times) {
  nand-not_network = gate_decomposition(g, times)
  for each nand gate in nand-not_network do
      if can collapse adjacent gates then
        if both gates are (non-)critical then
          collapse gates together
  end for
  if delay reduced then
    replace original gate
}
```

Figure 4.11: Pseudo-code for Gate Decomposition

The gate_decomposition routine produces a network of two-input nand gates and inverters. This network in then collapsed into a gate containing all critical inputs and gate(s) containing no critical inputs. The final group of gates is then compared with the original to insure the transformation reduced the delay. In most cases there are two gates added, the non-critical input gate and an inverter. However, sometimes the non-critical input gate is too slow, so two or more gates are formed from the non-critical inputs. Also, when splitting a complex gate, the structure may not allow the non-critical inputs to be placed in a single gate, so more than one non-critical input gate must be formed. These cases, however, are not as common. The actual implementation uses MisII to produce the nand-not network.

The ability to decompose gates is limited by the implementation technology. Full complex gate CMOS has no restriction, but standard cell CMOS requires that the gates formed by the decomposition be part of the standard cell library. Otherwise, the transformation may not take place. However, because of the way standard cell libraries are designed, it is unusual for gates that are the decomposition of another gate in the library not to be in the library too. Thus, this limitation is usually not a problem.

Speed Changes

First consider breaking up a nand gate, shown in Figure 4.9. Generalizing so that the nand gate has $n+1$ inputs, the data ready time for the original gate and the transformed gate are given by:

$$
\begin{aligned}
t_r(g) &= t_i(g) + \frac{n+1}{k_0} RC_L && (4.45)\\
t_r(g') &= \max(t_{i\ c}(g), t_r(g_2)) + \frac{2}{k_0'} RC_L && (4.46)\\
t_r(g_2) &= t_i(g_2) + \frac{nk_2'}{k_1'} RC + \frac{2k_0'}{k_2'} RC && (4.47)
\end{aligned}
$$

with $t_{i\ c}(g_0)$ = data ready time for input the critical input. Usually $t_r(g_2) < t_{i\ c}(g_0)$, and since $t_i(g_0) = t_{i\ c}(g_0)$, then the data ready time for the transformed gate, and the speedup are given by:

$$
\begin{aligned}
t_r(g_0') &= t_{i\ c}(g_0') + \frac{1}{k_0'} RC_L && (4.48)\\
\Delta t_r(g_0) &= \left(\frac{n+1}{k_0} - \frac{1}{k_0'}\right) RC_L && (4.49)
\end{aligned}
$$

If $t_r(g_2) > t_{i\ c}(g_0)$, implying that the data ready time for the gate split off from the original gate is slower than the critical input, c, then the change in data ready time is given by:

$$
\Delta t_r(g_0) = (t_{i\ c}(g_0) - t_r(g_2)) + \left(\frac{n+1}{k_0} - \frac{1}{k_0'}\right) RC_L - \left(\frac{nk_2'}{k_1'} + \frac{2k_0'}{k_2'}\right) RC \qquad (4.50)
$$

Area Changes

The change in area depends on the number of gates formed from the original when it is split, and if negative logic is used (requiring inverters to be inserted between the two gates). Since negative logic is used in CMOS, the transformation adds one inverter and one gate for each group of inputs split from the original gate. Thus, there are $n + 2m$ additional transistors when n inputs are removed from the original gate and placed into m gates.

Power Dissipation Changes

The equations for the dynamic power dissipation caused by driving the gates before and after the transformation shown in Figure 4.9 are as follows:

$$\begin{aligned} P_d &= 2(n+1)k_0CV_{dd}^2f_p & (4.51) \\ P_d' &= 2nk_1CV_{dd}^2f_p + 2k_2CV_{dd}^2f_p + 4k_0CV_{dd}^2f_p & (4.52) \\ \Delta P_d &= (2(n-1)k_0 - 2nk_1 - 2k_2)CV_{dd}^2f_p & (4.53) \end{aligned}$$

Thus the term $2(n-1)k_0 - 2nk_1 - 2k_2$ gives to a constant factor the change in the dynamic power dissipation by applying this transformation. A decrease in dynamic power consumption will result if the transistors in the gate that contains the fast inputs have size less than the gate that contains the slow inputs ($k_1 < k_0$) and offsets the increase in dynamic power consumption due to the added inverters. The change in short circuit power consumption is given by:

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3[\frac{\Delta\tau_{orig}}{T} + \sum_{i=1}^{m}(\frac{\tau_{inv}}{T} + \frac{\tau_i}{T})] \quad (4.54)$$

4.1.6 Gate Decomposition (Uniform Arrival Time)

When all the inputs to a large gate are critical, the transformation described in the previous section does not work well. In this case, splitting the gates into two equal sized parts, forming a tree of gates, reduces the delay for all the inputs of the large gate.

Example

Consider the gate on the left side of Figure 4.12, where a m-input nand gate is on the critical path. When the decomposition method is applied to this gate, the original gate has been split into a tree of five gates shown on the right side of Figure 4.12, below. The intervening inverters are required to maintain the correct phase. An alternative method replaces the pair of inverters and the two-input nand gate with a nor gate and a single inverter, reducing the area increase.

When transforming nand gates or nor gates, the gate will always be split into a tree of gates, with an extra inverter between the split gates and the top of the tree, in order to balance the delay through the different parts of the tree. For the case of a complex CMOS gate, shown in Figure 4.13, the decomposition becomes more

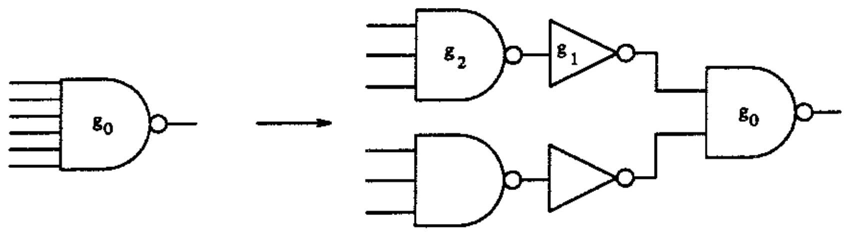


Figure 4.12: Example Circuit for Uniform Decomposition

complicated. With a uniform arrival time for the inputs, the gate may split into a group of gates that depends upon the structure of the gate. However, it is better to split the gate into two parts, even if they result in unbalanced delay, because a two-input gate at the top of the tree is better able to drive large loads than a many-input gate. The right side of Figure 4.13 shows that gate has been split into two parts.

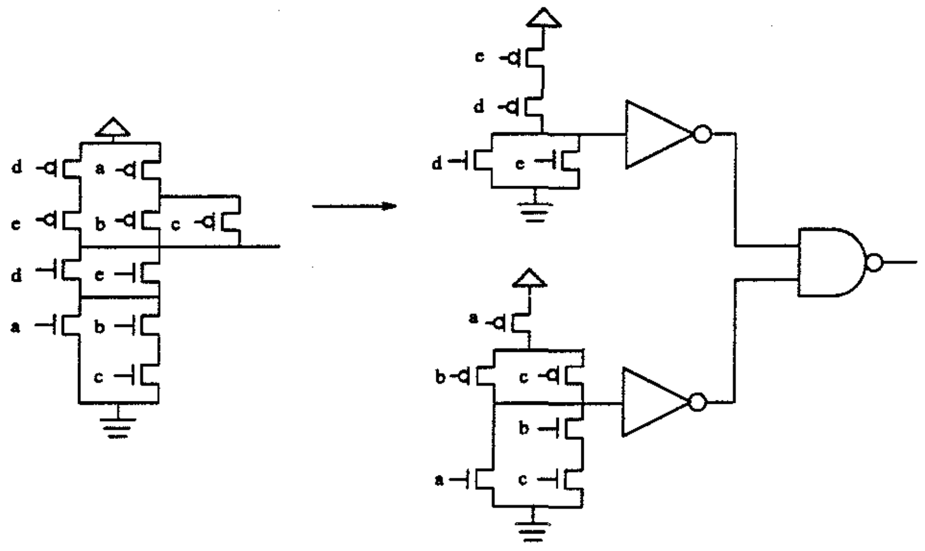


Figure 4.13: Complex Gate for Uniform Decomposition

Method

Gate decomposition for uniform input arrival times uses factoring methods that are derived from MisII[BRSVW87]. When there is a complex gate, the factoring method should split the gate so that the number of literals and the depth of literals is nearly even between the two parts. The pseudo-code for the generic factoring method is shown in Figure 4.14.

```
generic_factor(function $f$) {
  if ($|f| \neq 1$)
    return $f$
  $k$ = choose_divisor( $f$ )
  ($h$,$r$) = divide($f$, $k$)
  if ($|h| > 1$) {
    $k$ = cube_free( $h$ )
  }
  else {
    $k$ = one_literal_of( $h$ )
  }
  ($h$, $r$) = divide($f$, $k$)
  return generic_factor($k$) generic_factor($h$) + generic_factor($r$)
}
```

Figure 4.14: Pseudo-code for Gate Decomposition

The ability to decompose gates is limited by the implementation technology. Full complex gate CMOS has no restriction, but standard cell CMOS requires that the gates formed by the decomposition be part of the standard cell library. Otherwise, the transformation may not take place. However, these gates are usually available in the standard cell library, so this limitation is usually not a problem.

When working with standard cell libraries, this transformation can be implemented using table-lookup to find the best way to decompose the gate. This method is considerably faster than performing the decomposition on the Boolean equation for the gate. However, it can only be used when a fixed set of gates is available so that all possible decomposition rules can be determined beforehand.

Speed Changes

Consider breaking up a nand gate, shown in Figure 4.12. Assuming that the input arrival time is exactly the same for all the inputs, the data ready time for the original gate and the transformed gates are given by:

$$t(g_0) = t_i(g_0) + \frac{m}{k_0} RC_L \tag{4.55}$$

$$t(g_0)' = t_i(g_0) + 2\frac{m}{2k_2} k_1 RC + \frac{2}{k_1} k_0' RC + \frac{2}{k_0'} RC_L \tag{4.56}$$

Since the load driven by gate g_0 does not change, $k_0 = k_0'$, and $k_2 = 1$ (minimum size) since gate g_2 only drives an inverter. Then the change in the data ready time can be expressed as:

$$\Delta t(g_0) = \frac{(m-2)}{k_0} RC_L - \frac{m}{k_2} RC - \frac{2k_0}{k_1} RC \tag{4.57}$$

So, as long as $\frac{(m-2)}{k_0} C_L > \frac{m}{k_2} RC + \frac{2k_0}{k_1}$, the decomposition of the gate will decrease the delay in the circuit.

Area Changes

The change in area depends two factors. First, the increase depends on the number of gates produced by splitting the original gate. Second, the increase depends on whether or not negative logic is used, which requires additional inverters to be inserted. Since CMOS uses negative logic, the transformation adds one inverter and one gate for each division of the gate. Thus, there will be $2n+4$ additional transistors for each group of n inputs removed from the original gate.

Power Dissipation Changes

Assume a circuit shown in Figure 4.12. This gate is then split into five gates, so that the top of the tree is an two-input gate. The equations for the dynamic power dissipation to drive the gate before and after the transformation are as follows:

$$\begin{aligned} P_d &= 2mk_0 C V_{dd}^2 f_p & (4.58) \\ P_d' &= 2(2\frac{m}{2} k_2 C V_{dd}^2 f_p + 2k_1 C V_{dd}^2 f_p) + 2k_0 C V_{dd}^2 f_p & (4.59) \\ \Delta P_d &= (2(m-1)k_0 - 2mk_2 + 4k_1) C V_{dd}^2 f_p & (4.60) \end{aligned}$$

Thus the term $2(m-1)k_0 - 2mk_2 + 4k_1$ gives to a constant factor the change in the dynamic power dissipation by applying this transformation. A decrease in dynamic power consumption will result if the transistors in the gates that form the leaves of the tree have size less than the root of the tree ($k_2 < k_0$) and offsets the increase in power consumption due to the added inverters. The change in the short circuit power consumption is given by:

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3 [(\sum_{i=1}^{m} \frac{\tau_i}{T}) - \frac{\tau_{orig}}{T}] \tag{4.61}$$

where there are m gates after the decomposition. This equation shows that the short circuit power will increase.

4.1.7 Gate Simplification

Unlike some of the previous transformations, this transformation does not change the structure of the circuit. Rather, it changes the structure of a gate by attempting to replace it with a smaller, yet equivalent gate by removing non-prime literals and redundant terms. This technique is used in logic minimization, but it is included here because not all circuits to be optimizied have been generated using logic synthesis; some manual designs may be able to take advantage of this transformation.

Example

Consider the gate shown in Figure 4.15, below. This gate has the function $f = x \oplus y$ (exclusive-or). If there exists the input don't care conditions, $\mathcal{D} = xy$, then the gate can be simplified, using ESPRESSO[RSVD85], into $f = x + y$, shown on the right side of Figure 4.15.

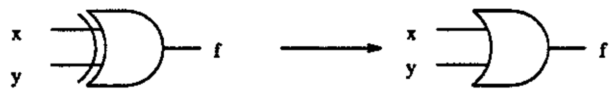


Figure 4.15: Simple Gate Simplification Example

A slightly more complex example, shown in Figure 4.16, contains a gate with the function, $f = \overline{(a\bar{b} + bc + a\bar{c}d)}$, and the don't care condition, $\mathcal{D} = ab\overline{cd}$. When a simplification method is applied to this gate, the gate shown on the right side of Figure 4.16, below, is the result. The function of the transformed gate is $f = \overline{(a\bar{b} + bc + ad)}$.

Method

In gate simplification, the goal is to replace a logic gate with an equivalent one that uses less area and is faster. There are many published methods for removing non-prime literals and irredundant terms. One of the most widely used is ESPRESSO. This transformation applies ESPRESSO to the selected gate. Speed of the simplification process can be controlled by how much time is spent determining the don't care set.

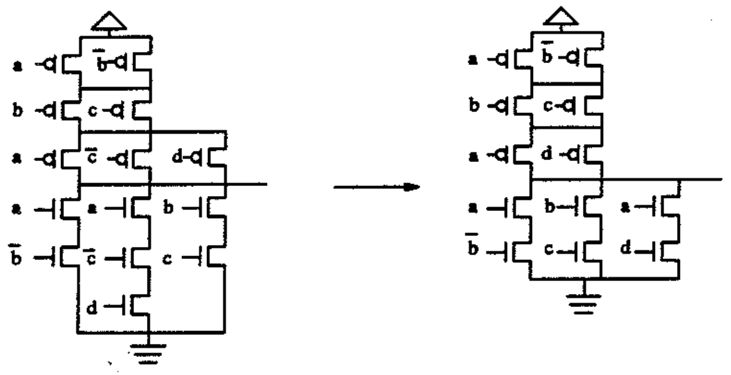


Figure 4.16: Circuit For Applying Gate Simplification

The cost of determining the complete don't care set is prohibitive for all but the smallest of circuits, so approximations that are subsets of the don't care set are used.

The gain in speed from this transformation can only be evaluated after it is performed since its effects are hard to predict beforehand. However, the evaluation is typically fast, provided the don't care set is not too large. Using a large don't care set, though, requires additional time to compute and use in minimization. Also, after minimization, the result must be factored before being evaluated to see if it improves the circuit.

Speed Changes

As mentioned earlier, the change in speed is difficult to characterize before the transformation takes place because it depends on several factors:

1. The number of literals removed.
2. Whether there is a decrease in the maximum number of series transistors in the gate.
3. Whether any of the critical literals were removed.

Unless slow inputs are completely eliminated from the gate, the expected speed gains from this transformation will be small.

Area Changes

The change in area depends on the number of literals removed. For each literal removed from the gate, two transistors are removed from the circuit.

Power Dissipation Changes

The dynamic power dissipation changes depends on the number of literals that are removed from the gate. This reduction will reduce the load of the previous stage to drive this gate, thus reducing the dynamic power consumption. If $2k_0C$ is the gate capacitance for each literal in the transformed gate, then reducing the appearance of a literal by n means a reduction of $2nk_0C$ in the gate capacitance that the previous stage must drive. This reduction allows the size of the previous stage's transistors to be reduced. If we assume that the previous stage's transistors do *not* change size, and if the literal i has n instances removed, then the change in dynamic power dissipation is:

$$\Delta P_d = 2nk_0CV_{dd}^2f_p \tag{4.62}$$

Since the transistors in the previous stage will never increase in size due to this transformation, the above formula represents a lower bound on the change in dynamic power dissipation. The change in short circuit power dissipation is given by:

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3(\frac{\Delta\tau}{T} + \sum_{i=1}^{m}\frac{\Delta\tau_i}{T}) \tag{4.63}$$

where $\Delta\tau_i$ is the change in rise/fall time for each gate driving the original gate. Since removing literals reduce the load on some of the inputs, this equation shows a drop in the short circuit power.

4.1.8 Gate Factoring

As in the previous transformation, this transformation, gate factoring, attempts to simplify a single gate, rather than restructuring part of the circuit by replacing the current factored form of the gate with another form that has fewer literals and/or transistors in series. As with gate simplification, this technique is also used in logic minimization, but since the input circuits are not assumed to have already had this technique applied, it is included in the set of transformation.

Example

The gate on the left side of Figure 4.17 has the function $x = \overline{((d + ag) + (d + g)af)}$. When a factoring method is applied to this gate, the gate shown on the right side of Figure 4.17, is the result. The new function of the gate is $x = \overline{((af + b)(ag + d))}$.

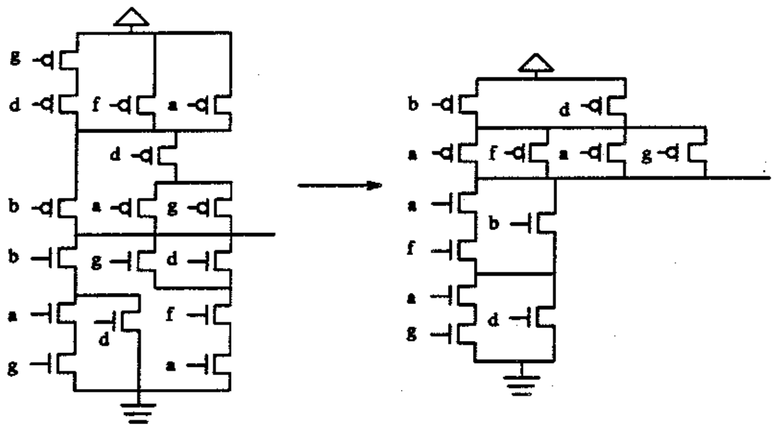


Figure 4.17: Circuit For Applying Gate Factoring

Method

Gate factoring applies the factoring methods used in gate decomposition transformation. There are two ways of varying the factoring method: changing the routine for determining the divisor and changing the routine for dividing the function by the divisor. Here, the method for determining the divisor computes all kernels and chooses the kernel that contains the largest number of literals. Algebraic division is used to divide the function by the divisor.

Speed Changes

The reduction in delay will come from decreasing the worst case series transistor path in the circuit. If the length of the worst case path before the transformation is x, and the path length is x' after the transformation, then the change in data ready time is:

$$\Delta t_r(g_0) = \frac{x}{k_0}RC_L - \frac{x'}{k_0}RC_L \tag{4.64}$$

$$= \frac{\Delta x}{k_0}RC_L \tag{4.65}$$

So the change in the data ready time is proportional to the change in the number of transistors in the worst case path. The expected speed gain from this transformation is small.

Area Changes

The change in area depends on the number of literals removed, which is difficult to predict beforehand.

Power Dissipation Changes

As in the previous transformation, gate simplification, the change in dynamic power dissipation depends on the number of literals that are removed from the gate. This reduction will reduce the load of the previous stage to drive this gate, thus reducing the power consumption. If $2k_0C$ is the gate capacitance for each literal in the transformed gate, then reducing the appearance of a literal by n means a reduction of $2nk_0C$ in the gate capacitance that the previous stage must drive. This reduction allows the size of the previous stage's transistors to be reduced. If we assume that the previous stage's transistors do *not* change size, and if the literal i has n instances removed, then the change in power dissipation is:

$$\Delta P_d = 2nk_0CV_{dd}^2f_p \tag{4.66}$$

Since the transistors in the previous stage will never increase in size due to this transformation, the above formula represents a lower bound on the change in dynamic power dissipation. The change in short circuit power dissipation is given by:

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3(\frac{\Delta\tau}{T} + \sum_{i=1}^{m}\frac{\Delta\tau_i}{T}) \tag{4.67}$$

where $\Delta\tau_i$ is the change in rise/fall time for each gate driving the original gate. Since removing literals reduces the load on some of the inputs, this equation shows a drop in the short circuit power.

4.2 Global Transformations

4.2.1 Fan-In Reordering

The purpose of this transformation is to identify chains of gates that have interchangeable inputs, and then reorder the inputs so that the slower inputs travel through a smallest number of gates and the faster inputs travel through a larger number of gates in the chain. This transformation can either be applied locally, that is, restricted to gates in a tree between two fan-out points, or globally, where the transformation is applied across tree boundaries. Since it can be used globally, we include it in the global transformations section.

Example

This transformation is best illustrated with an example. First, let us look at the transformation being used as a local transformation. Consider the circuit in Figure 4.18, where a group of four *and* gates form a string of gates. If the order of arrival time of the inputs from slowest to fastest are b, c, a, e, d, then the circuit may be restructured as shown on the right side of Figure 4.18, so that the slower inputs are nearer to the output of the string of gates.

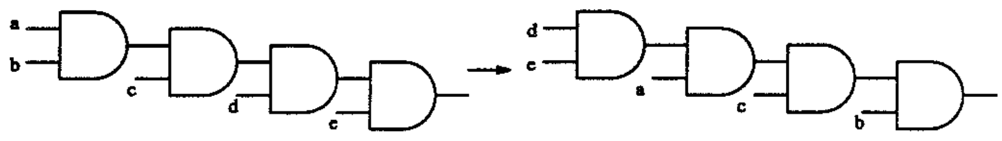


Figure 4.18: Fan-in Reordering Transformation

The primary requirement for changing the order of the inputs is that Boolean associativity must hold for the inputs. In the previous example, Boolean associativity holds for the inputs to the chain of *and* gates. When working with negative logic, as in CMOS, Boolean associativity occurs in several combinations of gates. For example, chains alternating nand gates and nor gates, chains of nor-inverter pairs of gates, or chains of nand-inverter pairs of gates all are Boolean associative. If Boolean associativity does not hold for all the inputs to be ordered, the inputs cannot be interchanged.

Complex gates complicate the way we handle a string of similar type gates. In order for the complex gate to be included in the chain, the chain feeding into the part of the complex gate must be the same type as the rest of the chain. For example, if a chain of *nand-inverter* gate pairs feeds into an *and* part of the complex gate, the *and* part can be included in the chain of gates. If the chain had fed into an *or* part of the gate then the complex gate could not be part of the chain. Unless the input to the complex gate is at the top of its structure (i.e. close to the output), the chain cannot include gates following the complex gate. The transformation breaks up the complex gate when they can be included in the fashion described.

So far only chains of gates have been discussed. This transformation also works on a tree of gates. Slow inputs still move up the tree to the root, and fast inputs move back away from the root.

When this transformation includes gates with fan-out greater than one that are not the last gate in the chain, logic must be duplicated to preserve the function of the circuit. This situation occurs when the transformation is used globally. An example of this situation is illustrated in Figure 4.19. Here, the first gate in the chain must be duplicated so that the other gates not on the chain that use its output as an input still have the correct input value.

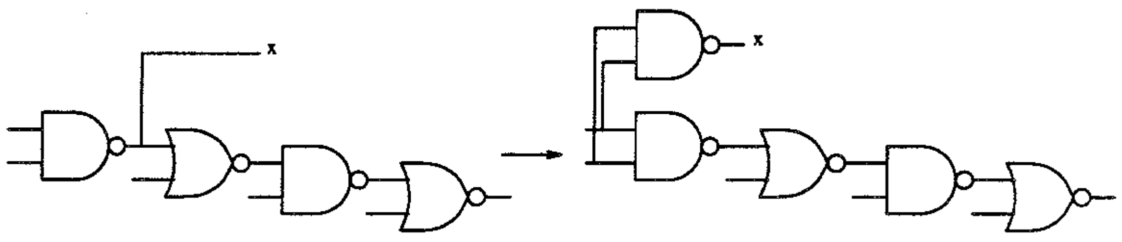


Figure 4.19: Logic Duplication During Fan-in Reordering

Method

The first step of this transformation is to identify a fan-in tree of gates where fan-in ordering can be applied. The first requirement, which is applicable only when applying this transformation locally, is that each gate in the fan-in tree, except the final gate, must drive only the next gate in the fan-in tree. Instead of trying to find chains of gates that have correct nand-nor, nand-invert, or nor-invert pairs, a simple analysis of the fan-in tree is made. The arrival times of all inputs to the tree and their delay through the tree is determined. If one input is very slow and has a long path delay through the tree, then the tree is a candidate for fan-in ordering. The tree is then decomposed into a simple and-inverter tree and the inputs ordered by arrival time. Then, the tree is converted to CMOS gates and compared with the delay of the original circuit. If the delay has decreased, then the new group of gates replaces the old fan-in tree. The actual algorithm is shown in Figure 4.20.

```
fan_in_ordering( gate_list $g$, arrival_times $times$) {
  $and\text{-}not\_network$ = gate_decomposition($g$, $times$)
  for each nand gate in $nand\text{-}not\_network$ do
      collapse together adjacent gates when possible
  for each gate in network do
      apply gate_decomposition (non-uniform arrival time)
}
```

Figure 4.20: Pseudo-code for Fan-in Ordering

Speed Changes

The reduction in delay will come from decreasing the number of stages that the slowest inputs must travel through. If the initial number of stages the slowest input s must travel through is k, and the delay through each stage is d, and the number of stages after the transformation has been applied is l, then the data ready times for the original circuit and the new circuit are:

$$t_r(g_k) = kdt_i(s) \tag{4.68}$$

$$t_r(g_k) = ldt_i(s) \tag{4.69}$$

$$\Delta t_r(g_k) = (k-l)dt_i(s) \tag{4.70}$$

So the change in the data ready time is proportional to the change in the number of stages due to the fan-in reordering.

If we have to duplicate logic because a gate on the reordering path has fan-out off of the path, then there will be a change in the delay due to different loads on the stages.

Area Changes

Since the transformation attempts to keep the number of gates constant, and no literals are removed from the circuit, the change in area caused by this transformation is zero. If gates are duplicated because of fan-out off of the reordering path, then there will be an area increase in the circuit.

Power Dissipation Changes

If this transformation only reassigns the inputs to a string of gates and does not add additional gates, there should be no change in power dissipation, unless the gates are of unequal size. It is expected that any increase in dynamic power dissipation for one input should be canceled by a decrease in dynamic power for another input, so the overall dynamic power dissipation should remain constant. If gates are added to the circuit description, then the dynamic power dissipation will increase. As long as the rise/fall times of the gates do not change, the short circuit power remains unchanged. Otherwise the change is given by:

$$\Delta P_s = \frac{\beta}{12}(V_{dd} - 2V_T)^3(\sum_i^m \frac{\Delta\tau_i}{T}) \tag{4.71}$$

where m is the number of gates in the chain, and τ_i is the change in rise/fall time for gate i.

4.2.2 Logic Dual Replacement

Occasionally inverters may be eliminated from a circuit description by replacing a set of connected gates with their dual and inverting the inputs and outputs. Also, the speed of a gate and its dual may not be equal; thus, replacing a gate with its dual may also reduce the delay as well.

Example

Consider the circuit shown on the left side of Figure 4.21, below, where a two-input nor gate with inverted inputs drives another two-input nor gate with an inverted output. The transformation reduces the total gate count and number of stages by replacing the nor gates with nand gates and inverting the inputs and outputs. Since in CMOS technology, nand gates are faster than nor gates, the circuit gains additional speed by the substitution of the dual. The resulting circuit is shown on the right side of Figure 4.21. Note that the intermediate output requires an inverter to maintain the correct polarity.

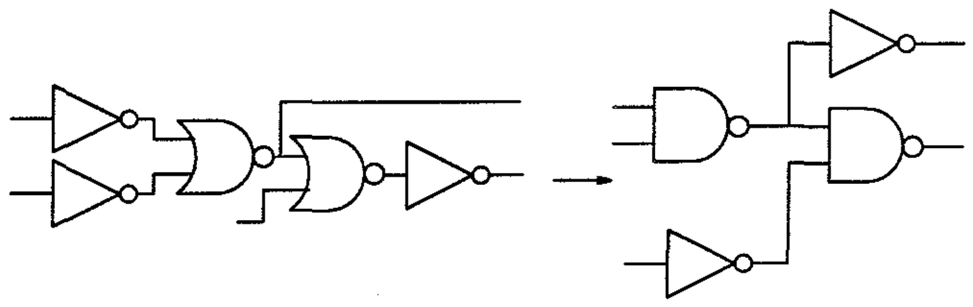


Figure 4.21: Circuit For Applying Logic Dual Replacement

Method

The first step is to identify a chain of gates that can be replaced by its dual. Since current technology mapping procedures do a good job at minimizing inverters in a single tree of the Boolean network, several trees on the path must be considered by this transformation. Simply replacing a single gate with its dual, which is used in previous published methods, will not usually work well unless that gate is a single tree on the critical path. The first step is to identify a tree on the critical path that has some of its critical inputs inverted in a previous tree and has the output inverted in a tree occurring further down the critical path. When such a group of trees is identified, the gates on the chain are replaced by their dual, and the inputs and outputs of the gates that are not on the chain are inverted. An inverter is added to the inputs and outputs (not including the output on the chain of gates) only if the non-inverted signal is not available elsewhere in the circuit. These inverters preserve the correct polarity of the intermediate signals. If one of the inputs is inverted, then the input to that gate is used rather than inserting an additional inverter. Then the new circuit is compared with the old circuit to see if there is an improvement in the area and delay. If the gates were on the critical path and the delay is improved, then the new circuit is accepted. If the gates were not on the critical path and the area improved, then the new circuit is accepted. After accepting the new circuit, any inverters that are no longer used are removed from the circuit. This transformation must be used carefully so that the insertion of inverters to preserve correct signal polarity does not slow down paths that are more critical then the path being transformed. The pseudo-code for this transformation is as shown in Figure 4.22.

The implementation technology may limit the gates that may be replaced by their dual. With full complex gate CMOS, there is no restriction, but with standard cells,

```
identify consecutive trees on critical path for candidate chain
for each gate in chain do
    replace gate with its dual
    for each input (output) not from chain do
        if input (output) not inverted then
            invert input (output)
        else
            connect to non-inverted input (output)
    end for
end for
if dual of chain is faster than original chain then
    replace chain
else
    circuit is unchanged
```

Figure 4.22: Pseudo-code for Logic Dual Replacement

the gate's dual must be part of the standard cell library. Otherwise, the transformation may not take place. Most standard cell libraries are symmetrical, so the availability of a gate's dual is not a problem.

Speed Changes

There will be an increase in speed if inverters can be removed from the critical path, or if faster gates replace the original gates. Consider the example in Figure 4.21. If the resistance for nands and nors are the same, then the change in delay is given by the equation for the change in ready time for the output of the chain:

$$\Delta t_r(g) = \frac{2R}{k_0}k_1 C + \frac{R}{k_3}C_L - \frac{R}{k_2'}C_L \tag{4.72}$$

Since the resistance of the nor is worse than the nand, this equation represents a lower bound on the improvement possible.

Area Changes

The change in area depends on change in the number of inverters, since changing the logic family of the gates will not change the area. If the number of inverters increases, then there will be an area increase of two transistors per inverter. Similarly, if the number of inverters decreases, then there will be an area decrease of two transistors per inverter.

Power Dissipation Changes

The transformed area of the circuit will still operate at the same frequency, since all inputs and outputs are inverted. Thus the change in dynamic power depends on the number of inverters removed. The change in short circuit power also depends on the change in rise/fall time for the gates in the chain that are replaced by their dual.

4.2.3 Logic Path Resynthesis

Technology-independent logic synthesis techniques reduce the amount of area required in the circuit by increasing the amount of shared circuitry. This approach may result in some long, slow paths, that contain non-critical fan-out. The reduction in delay of the path using local optimizations is limited by this non-critical fan-out. Each subexpression becomes a fan-out on the critical path. With a large number of common subexpressions for the function, the path becomes long. An alternative optimization technique synthesizes replacement circuitry for the critical path that is optimized to compute only the function on the critical path. This technique, called logic path resynthesis, is described in this section.

Example

This transformation is best illustrated with an example. Consider the circuit in Figure 4.23, where several single bit adders are combined to form an adder. The critical path through the circuit is the carry-ripple chain. Replacing this chain with some carry-lookahead circuitry that is optimized to compute the carry-out signal improves the speed of the adder.

Method

Once a critical path is chosen, a cut point is selected on the critical path. The circuit is broken at the cut point and all the gates in the transitive fan-in of the cut point (gates on and off of the critical path) are used in the resynthesis operation. These gates are run through MISII for logic minimization, decomposition, and technology mapping. The resulting set of gates is added to the circuit and the output of the new set of gates is connected to drive the critical path starting at the cut point. Finally,

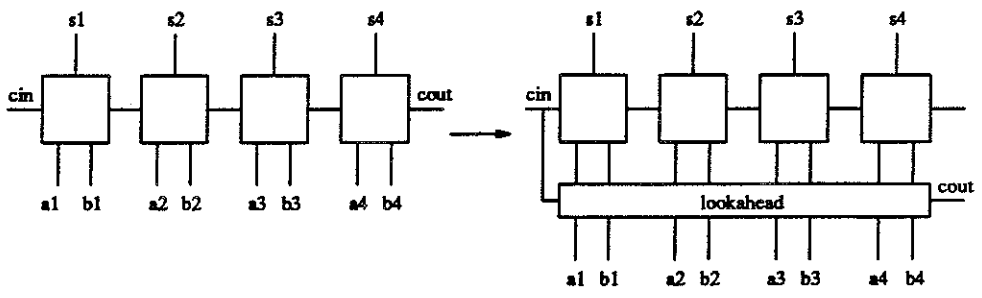


Figure 4.23: Adder Example

the original set of gates in the transitive fan-in of the cut point are checked to see if they still drive other gates in the circuit. Those gates which are no longer used in the circuit are then removed.

The remainder of the discussion on the method is split into two sections. The first section describes the theoretical basis for this transformation, and the second section describes the method for choosing the cut point.

Theoretical Basis

Before describing this transformation, the following definitions are needed[HL87, BBH+88].

Definition 4.1 (External Don't Care Set) The external don't care set is a don't care set for a Boolean network that is derived from conditions *outside* of the Boolean network. There are two components: DXP and DXO. DXP is the set of all primary input minterms that will not occur as input to the Boolean network. DXO_i, which is defined separately for each primary output i, is the set of primary input minterms for which primary output i will not be used. Then, the total external don't care set for primary output i, DX_i, is: $DX_i = DXP + DXO_i$. □

In addition to the external don't cares, there are two sets of internal don't cares, the satisfiability and observability don't cares. Both of these sets arise from the structure of the Boolean network.

Definition 4.2 (Satisfiability Don't Care Set) For node j in the Boolean network, its functions F_j (expressed in terms of the primary input and intermediate variables), and its corresponding intermediate variable y_j, the satisfiability don't care set for j is defined as follows: $DSAT_j = F_j \oplus y_j$. □

Definition 4.3 (Observability Don't Care Set) For node j in the Boolean network, its functions f_j, and its corresponding intermediate variable y_j, the observability don't care set for j is defined as: $DOBJS_j = \prod_{i \in PO \cap TFO(j)} (DX_i + D_{ij})$, where $D_{ij} = \{x \in B^{n+m} \mid (y_i)_{y_j}(x) = (y_i)_{\overline{y_j}}(x)\}$, where $(y_i)_{y_j}(x) and (y_i)_{\overline{y_j}}(x)\}$ are the logic variables that represent the cofactored functions $(F_i)_{y_j}(x) and (F_i)_{\overline{y_j}}(x)\}$[1]. PO is the set of primary outputs, and $TFO(j)$ is the set of nodes in the Boolean network that are in the transitive fan-out of j. In other words, a node n is in the $TFO(j)$ if there is a path from the output of j to n. □

The total don't care set for each node, j, in the Boolean network is defined as:

$$D_j = DOBS_j + DSAT_j \tag{4.73}$$

In path resynthesis, a subnetwork of the Boolean network will be resynthesized. Considering just the subnetwork requires the sets DXP and DX_i to change to take into account the structure of the remainder of the network:

$$DXP = DSAT_{y_s} \cup DXP_n[y_i \in FI(y_s)] \tag{4.74}$$

$$DX_i = (\overline{DSAT_{y_s}}) \cap (DX_i + DOBS_{y_s})[y_i \in FI(y_s)] \tag{4.75}$$

The expression $[y_i \in FI(y_s)]$ means that all literals not in this set are dropped, since they are not in the subnetwork. Now if the primary outputs of the subnetwork are removed, except for the output on the critical path, only $DOBS_j$ changes. In fact, the observability don't care set will not change only if the following occurs:

1. there was only one primary output for the subnetwork, or

2. $DX_i + D_{ij}$ is the exact same set for every primary output i in the transitive fan-out of j.

So in removing all but one output, the size of the don't care set *increases*, which improves the chances of reducing the size of the Boolean subnetwork. Even if the don't care set does not increase in size, the removal of all but one of the primary outputs allows the subnetwork to be split into fewer trees, so that the tree mapping phase of the technology mapping procedure can do a better job.

[1]The cofactor of function F_i with respect to literal v_j, $(F_i)_{v_j}$, is the function F_j with all terms containing literal $\overline{v_j}$ removed and all literals v_j removed.

This method will never produce a slower path, assuming a "perfect" logic minimization tool. In practice, where the logic minimization tool is based on heuristics, all cases have produced an improvement in the speed of the circuit because the heuristics work better on smaller circuits and the larger don't care sets.

Cut Point

To select the subnetwork to resynthesize, the first step is to identify a critical path. Then, a cut point is selected. There are three issues involved in choosing a cut point. First, the cut point must include a sufficient section of the circuit so that resynthesis will reduce the delay of the path to meet the timing constraints. This requirement suggests selecting a cut point close to the primary output. Second, the cut point should allow as many critical fan-out paths as possible that occur on the path after the cut point to share the speed-up from this transformation. If fewer paths share the speed up, then this transformations will need to be applied many times, which will cause a large area increase. This requirement suggests selecting a cut point close to the primary inputs. Third, the cut point should be selected so that there is a small number of gates in the fan-in tree. A small fan-in tree limits the area increase for each application of the transformation. Also, since the performance of logic synthesis tools is not even close to linear with respect to the problem size, a large fan-in tree may take a considerable time to resynthesize. This case is especially true when the fan-in tree is very broad. This requirement to limit the size of the fan-in tree also suggests selecting a cut point close to the primary inputs. Since this requirement is related to sharing, both are considered together.

The graph in Figure 4.24 shows how the speed-up and sharing are effected by the number of stages before the cut point. The cross-over point represents a reasonable trade-off between the two, and is used as the cut point. While the amount of sharing can be easily evaluated, the speed-up for a particular cut point cannot be determined except by performing the resynthesis operation. Therefore, the actual speed-up is approximated by the value of the contribution to the path delay for the path up to the cut point, which represents an upper bound on the speed-up possible through resynthesis.

Once the cut point is selected and the subnetwork consisting of the transitive fan-in of the cut point is built, determining if there exists an equivalent network with fewer literals is a co-NP hard problem[KR89]. So, even after selecting the cut

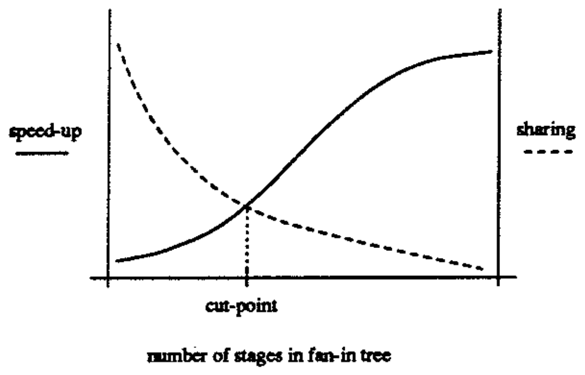


Figure 4.24: Speed up versus sharing

point, determining whether or not it will reduce the delay in the circuit is a difficult problem. Therefore, some heuristics are used to select the cut point.

Before describing the heuristics, notice that many paths in the circuit share the same gates at the beginning of the path. The critical paths that share the same gates at the beginning of the path with other critical paths are called *mops*. The set of gates at the beginning of the path that are shared in common between the critical paths in the mop is called the *handle*. The number of stages in the handle is called the *handle length*. Circuits that have mops that contain many critical paths can get a greater benefit from path resynthesis because more paths are effected. Analysis of the circuits described in Chapter 6 show that all the circuits contain mops with multiple paths even when the handle length approaches fifty percent of the total path delay. This result is encouraging because it indicates that a small number of applications of this transformation can effect a large number of paths. It also indicates that consideration should be made to select a cut point in the mop handle, so that paths in the mop all share the speed up.

One fact helps simplify the selection of a cut point. If there are critical fan-outs from the critical path, then is the critical path is part of a mop. If the critical path is the slowest path in the mop, then a cut point that provides a sufficient decrease in delay for the critical path, will also provide sufficient speed up for all the other paths in the mop that fan-out after the cut point. This results allows the speed up requirements of the critical paths that fan-out from the most critical path in the mop to be ignored.

There are several measures that can be used to evaluate a cut point. The first measure is the percentage of the total path delay that occurs along the section of the path from the primary inputs to the cut point. This value gives an upper-bound on the reduction in delay using the cut point. A second measure is the number of stages on the critical path from the primary input to the cut point. The final measure is the number of paths in the transitive fan-in of the cut point that do not reconverge into the transitive fan-in. These paths represent intermediate functions computed by the transitive fan-in that may not be used to compute the function at the cut point and may be eliminated or simplified.

After some experimentation, it turns out that a simple method is effective. To find the cut point, the output of a gate on the path is found where the percentage of the path delay up to that point is equal to α times the required speed-up for the path, where the value of α is between one and two. If the output has fan-out greater than one, then this point is the cut point. Otherwise, the cut point is the next gate output occurring along the path that has fan-out greater than one. This method allows for sharing and limits the size of the transitive fan-in tree.

Unfortunately, this method of selecting a cut point does not always produce enough of a delay decrease. The method is therefore modified so that several cut points along a critical path may be used. The first cut point is still selected based on the required speed-up. However, if the path does not get a sufficient speed-up, then a new cut point is found on remainder of the path, and the stages up to that cut point, but not including the previously resynthesized part, are resynthesized. This process is repeated until either the path is fast enough, or the path has been completely resynthesized. While this method will not produce as much speed up as resynthesizing the entire path, the gain from increased sharing of the speed-up and smaller increase in area help offset that drawback.

A parallel of this technique can be seen in the design of carry chains for adders. If a 32-bit adder is built, the lookahead chain is not a single unit for all 32 bits, but rather is made up of several units. Each of these units can be thought of as a group of gates resynthesized using a different cut point.

Speed Changes

The reduction in delay will come from two sources. First, the replacement circuitry is optimized for the computation of just one function. Therefore, the size of the subnetwork is reduced, translating into fewer stages and reduced delay. Second, the gates have been chosen for smallest delay.

Area Changes

This transformation can have a significant cost in area because it is duplicating logic already present in the circuit. The size of the area increase depends on how much logic was selected to be resynthesized, how much of a reduction in logic was possible during the resynthesis, and how the resulting logic was mapped. A worst-case upper bound is the size of the original fan-in tree up to the cut point.

Power Dissipation Changes

As with area changes, the increase in dynamic and short circuit power dissipation can also be significant because of the possibly large number of gates that can be added to the circuit. A worst-case upper bound is the power dissipation of the original fan-in tree up to the cut point.

Chapter V
Application of Transformations

The technique described here determines a set of critical paths through the circuit and then applies a set of transformations to these critical paths. A greedy method is used: the slowest critical path is transformed first, with the transformation that gives the biggest gain applied to it first. This greedy approach is used because the complexity of the problem is large. However, knowledge about the problem is incorporated into the greedy heuristics to improve the results.

Since the task of speeding up the circuit can be broken into a two-step problem of identifying the critical paths and applying transformations to these paths to speed them up, the discussion of the heuristics are broken along the same lines. The next section in this chapter describes the method used to select gates on the critical paths to be transformed. Then, the second section of this chapter discusses the selection of a transformation once the gates to be transformed are determined.

5.1 Selecting Paths

5.1.1 Critical Path Analysis

The critical paths are found by a depth-first delay analysis of the circuit using a data-independent, static timing analysis technique. Beginning at a primary input, each path in the circuit is traced, as long as it is the slowest signal arriving at each node so far. This method assumes that there are no cycles in the circuit, which is true since only combinational logic is allowed. No clocking is allowed as input to the circuit, since the circuit is assumed to be combinational. After the timing analysis is complete, the slowest N paths that do not meet the timing requirements are kept. This technique is similar to the approach used in Crystal[Ous83, Ous84].

An alternative method is to use a breadth-first approach rather than a depth-first approach. The breadth-first approach is faster than the depth-first approach because of only one delay analysis step occurs at each gate in the breadth-first method, while several may occur using the depth-first approach. Also, the breadth-first method can handle, without changes, circuits with feedback. However, while the breadth-first method will determine which primary outputs are critical, extra mechanisms are required in order to extract the actual critical paths through the circuit. The depth-first method, produces this information at no additional cost.

This method for determining critical paths can identify false paths, that is, paths that do not actually occur because the model takes a worst-case view of the delay of the gates. As mentioned in Chapter 3, we assume that they do not occur because of the large amount of additional run-time needed to remove false paths from the circuit.

5.1.2 Path Grouping

Many critical paths produced by this method are not disjoint; the paths may share fan-in, fan-out, or converge back together. Using the static timing analysis techniques described above, these paths would be considered distinct. However, if a transformation is applied to one of the paths and the delay on the path is reduced, then the delay is also reduced for all the paths that share the same transformed gates. Thus, from the point of view of logic optimization, these paths are not distinct.

The timing analysis tool TV[Jou83a, Jou83b, Jou87] addresses the problem of providing user-informative paths, that is, only reporting critical paths that represent a unique timing problem. This problem is similar to the problem of grouping paths for logic optimization. TV groups paths into equivalence classes, and then only reports the slowest path in each class. A similar approach is used here. Two paths are considered equivalent if the slowest stages of each path that contribute at least 60 percent of the total path delay are the same. The stages do not all need to be adjacent in the path, so that paths that differ only by sections between reconvergent fan-out can be considered equivalent paths, depending on which gates are shared. Unlike TV, though, two paths that are corresponding paths of duplicate bit-slices are considered separate paths because transforming one path will not effect the other path.

The implementation of path grouping is defined by the pseudo-code in Figure 5.1.

```
while set of paths $\neq \emptyset$ do
    remove most critical path and place in new group
    for each gate on critical path do
        add paths attached to gate to list of candidate paths
    end for
    remove paths in candidate set from path set that share majority delay
    with the most critical path
end while
```

Figure 5.1: Path Grouping Algorithm

5.1.3 Gate Selection on Critical Path

Once the paths have been grouped, the gates on the paths must be selected for transformation. The selection depends upon which type of transformation, local or global, that is to be applied.

Local transformations only affect the gates in a single tree of gates. Recall, from chapter 4, a tree of gates in the circuit is defined as a set of connected gates, each of which have fan-out equal to one, except for the root. These trees are used in technology mapping and are useful in deciding where to apply the transformations. Since local transformations only effect a tree, the critical path can be subdivided into trees, with a division occurring at each point where the fan-out is greater than one. Local transformations are then applied to some or all the gates in the tree, where appropriate.

The scope of global transformations, on the other hand, crosses over tree boundaries and so considering a tree at a time for the global transformations is not appropriate. In this case, their several adjacent trees are considered, as with logic dual replacement and the global version of fan-in ordering, or the entire path is considered, as with path resynthesis.

5.2 Order of Application

There are three issues to be considered before deciding on the order of application for the transformations. The first issue is the method of choosing the next transformation: whether it is fixed or flexible. The second issue is interaction between applying a transformation several times to the circuit. The third issue is the interaction between different types of transformations. All three issues guide the determination of the order of application for the transformations.

5.2.1 Fixed Order Versus Variable Order

The method used is a fixed order approach, that is, each transformation is applied in a particular order to the circuit. So, for example, buffering is applied everywhere possible on the critical paths in the circuit before gate collapsing is tried. This choice is based on the goal to change the circuit as little as possible in order to meet the timing requirements of the circuit. If we assume that the transformations are independent, which is not true, and if we order the transformations in terms of the gain in speed they produce and assume the circuit can be sped up, then the fewest number of transformations will be applied. This will also decrease the execution time of an automatic tool implementing these transformations. Since the transformations are not independent, the ordering is more complex, but we still use this heuristic as the basis for determining the order and use some knowledge of the transformations to reduce interactions between the transformation. Also, a fixed order for the transformations simplifies some of the analysis of interactions between the transformations. These interactions are described below.

An alternative method is to allow the order of application to be unrestricted. Instead, any of transformation can be applied to each path, based on the causes of the delay for the path. There are two principal problems with this method. First, in many cases, the same order of application seems to occur because there are certain problems, such as load mismatches, which cause large delays. Second, the causes of the path delay can be complex and interrelated. Often, there is no clear choice as to which problem is the primary cause for the delay. For example, a gate may have a very slow input. The cause of the slow input may be equally shared by a large gate occurring earlier in the path and a large load that another gate occurring earlier in the path must drive. So it is not always clear which of the three gates should be transformed. For these reasons, a fixed order is used in the implementation, with a variably ordered technique to be a question of future research.

5.2.2 Common Framework

Before describing the interactions, it is useful to group and classify each transformation by the way it affects the circuit as a way of grouping the dissimilar transformations together. For example, gate collapsing and gate decomposition remove and add gates to the network, respectively. They can be grouped together because one is the inverse of the other. On the other hand, fan-in ordering (the local version) rearranges the inputs between gates. However, fan-in ordering can really be considered as a combination of several gate collapsing and decomposition operations combined together. The only difference is that the arrival times of the inputs are guiding the operations rather than the size of the gates. Looking at the Boolean network representation, none of these operations change the Boolean network; they just change the mapping into gates. At this level logic dual replacement can be added, because it does not change the Boolean network although it does change the mapping. If any of these operations are applied across tree boundaries, the Boolean network will change, but only in small amounts. What this similarity implies is that these operations are really different versions of the same underlying operation, namely the mapping of a node in a Boolean network into a set of gates. This similarity also implies that in terms of ordering these transformations, the transformations in this group should be applied one after the other.

Buffering and critical signal isolation also form their own group. They are clearly the same operation, differing only in how the circuit is connected after the operation is complete. They are different from the first group because they are not purely mapping operations, but consider loading at particular points in the circuit.

The two other local transformations, gate simplification and gate factoring, do not fit together into either group, but this is not surprising since these operations are mainly used in logic synthesis, rather than in delay optimization. Gate simplification changes the underlying Boolean network so it does not belong in either group. While gate factoring does not change the Boolean network, its use in logic synthesis places it in this group.

The remaining transformation, logic path resynthesis, also does not fit in with the others, but for different reasons. This operation makes large changes to the structure of the Boolean network, reversing the sharing that was done in logic synthesis. Therefore it is placed in its own group.

5.2.3 Interactions Between Successive Applications of a Transformation

When applying a particular transformation, it may interact with previous or successive applications of the same transformation. This section describes these interactions.

Several transformations will not interact with themselves. This non-interaction is a desirable property because it simplifies the heuristics needed to apply the particular transformation. For example, both buffering does not interact with itself and critical signal isolation does not interact with itself. They can only be applied at fan-out points and only once at a particular fan-out point. Furthermore, the load at any other fan-out point does not influence whether the transformation needs to be applied at a particular fan-out point. Gate decomposition does not interact with itself either. Splitting any gate does not prevent adjacent gates from being split. Gate simplification and gate refactoring also do not interact with themselves for precisely the same reason.

The global transformations and gate collapsing do interact with themselves. Applying gate collapsing between a pair of gates will prevent other pairs that include one of the collapsed pair from being collapsed. When optimizing for delay, gate collapsing is restricted to being applied across tree boundaries which restricts some of the interactions, but not all of them. One problem that can occur is the slow down of critical paths. For example, when gate collapsing is applied across tree boundaries the first gate is in one tree and the second gate is in another tree driven by the first gate. Since the first gate must drive more than one gate, there is a choice for the second gate, namely each gate driven by the first gate. Several of these driven gates may be critical, so application of the transformation using any of the critical gates may speed up one of several critical paths. However, only the most critical gate driven by the first gate may be used, because duplication of logic caused by the collapse of the two gates slows down the gates that drive the first gate of the pair. This situation is illustrated in Figure 5.2.

When gate collapsing is used for area savings, it is usually not applied across tree boundaries, and so while the previous problem may not occur, there is still another problem, namely the order of application when several gates can be collapsed. The question of order is currently resolved by selecting the gates with the highest slack first. While it does not guarantee an optimal choice of gates to collapse, it does insure

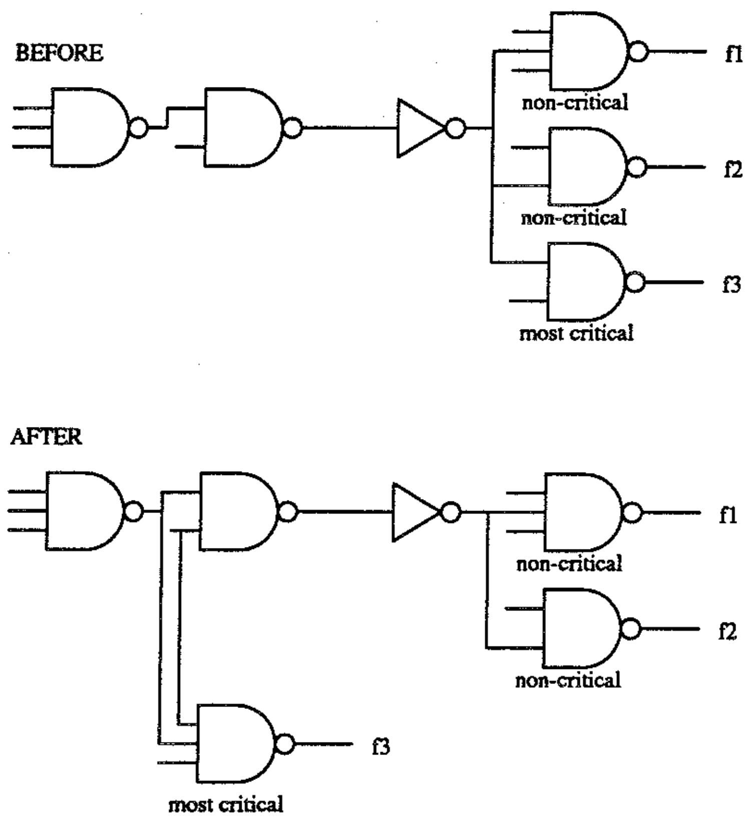


Figure 5.2: Interactions in Gate Collapsing

that the paths containing the gates will not become critical.

The application of fan-in reordering also has some pitfalls. Even when limited to just re-ordering inputs on a tree, it can still interact with subsequent applications. This interaction is apparent when deciding how much of the tree to use when applying transformation. Using only a small part of the tree may prevent a better result when using more of the tree, while in other cases, there is no additional benefit using the entire tree because only a small part of the tree can be reordered. A simple solution to this problem is just to always use the entire tree. This method will always do at least as well as using just part of the tree. When applying this transformation across tree boundaries, there is much stronger interaction, similar to the problem with gate collapsing across tree boundaries. For example, consider one tree in the Boolean network that drives several other trees. If two of these trees can reorder inputs using some of the gates in the first tree, there is competition between them. If extra logic is added in, it will slow down some of the paths. Because of this duplication of logic and its effect of slowing paths, only the most critical pair of trees will be used for the transformation.

Interactions also occur between successive applications of logic dual replacement. There may be several overlapping places on a single path, or application of the transformation on one path may prevent its application on a second path that shares gates with the first path. To prevent slower paths from slowing down further, care must be used so that slower paths do not get additional stages. Therefore each tree considered during this operation must have the current path as the most critical if additional stages are to be inserted.

Finally, path resynthesis also has interaction between successive applications of the transformation. Part of this problem is related to the choice of cut point, which was discussed in chapter 4. Once path resynthesis has been applied to a particular path, the section of the path replaced is not considered in any subsequent operation.

5.2.4 Interactions Between the Application of Different Transformations

Local transformations can be divided into three groups, as described above:

- buffering and critical signal isolation
- gate collapsing, gate decomposition, and fan-in ordering

- gate simplification and gate factoring

The last important step in deciding the order of application is to identify interactions between transformations of the different groups.

The first group, buffering and critical signal isolation, is independent of the other two groups of transformations. These two transformations are only applied at the root of a tree, and only if the load is large. They are only applied at the root of a tree because this location is the only place where the fan-out will be greater than one, and therefore the only place where the load will be large enough to warrant the use of these transformations. The other transformations will not appreciably effect the size of the load and so are independent of buffering and critical signal isolation. Similarly these two transformation do not affect the other groups. However, buffering and critical signal isolation do interact with each other, since both are applied to the same place in the tree and for the same reasons. This problem is resolved by always applying critical signal isolation before buffering. This order prevents resources being spent to reduce delay on non-critical paths. Whenever buffering is applied, it is only to speed-up critical paths.

The second group, gate collapsing, gate decomposition, and fan-in ordering, when applied locally, is independent of the first group. This is clear because the transformations in this group are used to find a better mapping of the tree, while the first group is concerned with the loading between trees. When gate collapsing or fan-in ordering are applied across tree boundaries, they do interact with the first group. But since these transformations are concerned with a single path rather than all the paths at the fan-out point as in critical signal isolation and buffering, these transformations are applied after the first group. As with the first group, the transformations within the second group are not independent of each other. To resolve this problem, the order of application is fan-in ordering, followed by gate collapsing, and gate decomposition. Fan-in ordering is performed first because either it was not done during the original mapping of the tree, or the delay model used was too crude. The accurate arrival times of the inputs are not available during technology mapping. Therefore, applying it to the tree is most likely to decrease the delay. Then gate collapsing is applied, followed by gate decomposition. Gate decomposition is applied last because most of the gates in the circuit are not large enough to get a large gain from this transformation. The gates are small because of the small size of the gates available in the gate library used to construct the circuit. If large gates are used, this transformation would still be used last because the gains from applying it would still be small.

The last group, gate simplification and gate factoring, are really Boolean network operations. If they can be applied to the circuit, they reduce both area and delay. The two transformations are considered Boolean network operations because they operate at the Boolean equation level, independent of the particular gate. In fact, most gates are too small for either of these transformations to improve them. Therefore, these transformations are done before any other transformations are applied to the circuit, but only when using gate libraries containing gates large enough for the transformations to be useful and only if the example has been generated by hand. Although these transformations interact with all the other local transformations, this strategy is reasonable because the resulting circuit then represents a local minimum for the particular Boolean network structure and so is a reasonable starting point for the application of the other transformations. Since gate simplification and gate factoring are not independent of each other, an ordering of their application is also needed. Gate simplification is applied first since it reduces the Boolean function. Gate factoring only finds the best representation of a Boolean function, so it works better when starting with a minimized function.

Global transformations, by their very nature, interact with each other and with the local transformations. They are applied after the local transformations for two reasons. First, if the circuit can meet the timing constraints using only the local transformations, then the overall size of the circuit will not increase much. Second, the global transformations require more run time and increase circuit area.

Logic dual replacement as a global transformation interacts with the local transformations of buffering and critical signal isolation, but since it is a global transformation, it is applied only after the local transformations. While this restriction limit the number of times this transformation can be applied, better results occur with critical signal isolation and buffering. If the technology mapper did a reasonable job on phase assignment, the number of possible places to apply this transformation will be small anyway.

Finally, path resynthesis obviously interacts with any transformation applied to the path being considered for resynthesis when it occurs before the cut point. This interaction is of no consequence because applying path resynthesis means that the other transformation did not provide sufficient decrease in delay and therefore did not prevent path resynthesis from being applied. Since path resynthesis has a much greater penalty in area, it will always be tried last.

So, the order of application will be:

- gate simplification
- gate factoring
- critical signal isolation
- buffering
- fan-in ordering
- gate collapsing
- gate decomposition
- logic dual replacement
- path resynthesis

Chapter VI
Results

In this chapter, the results of some experiments are presented and analyzed. The presentation begins with a description of the test cases used in the experiment and includes an explanation of the method used to generate them. The second section contains a description of the experimental method and types of experiments. Two sets of experiments were run using POLO (Performance Oriented Logic Optimizer) [KH89], which is the implementation of the techniques described in the previous chapters. In the first experiment, all the transformations are used to minimize the delay of the circuit, allowing an analysis of the amount of improvement and cost expected with this technique. In the second experiment, the first experiment was repeated, but unlike the first experiment, the transistor sizes for all gates were fixed to the default value. In the final section, the results of each experiment are described.

6.1 Test Cases

The test cases used in the experiment are derived from examples in the benchmark set from the 1989 MCNC International Logic Synthesis Workshop[Lis88]. These circuits have been assembled from both academic and industrial sources and are considered representative combinational logic circuits. The examples in the benchmark set are in either PLA (two-level Boolean equation) or BLIF (multi-level Boolean network) form. Since the experiments require gate-level netlist circuits as input, these descriptions were processed to form netlist circuits, as illustrated in Figure 6.1. Each benchmark example was run through one of the available logic synthesis tools, DECAF[LKB87], MISII[BRSVW87], or BOLD[BHJ+87], to simplify the logic descriptions and map them to gate-level netlists. In several examples, the logic decomposition and minimization operations of BOLD were combined with the technology mapping routines of MISII to produce the netlist. This step is illustrated by the box in Figure 6.1. All the tools were used in their default configuration. The target library set was the

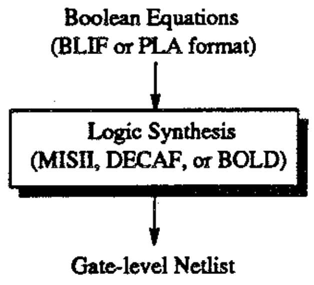


Figure 6.1: Technique to Generate Examples

MCNC International Logic Synthesis Workshop Lib2 library, which contains twenty-six gates. Three logic synthesis tools were used rather than one, so the results of the experiment would not be biased by any shortcomings or idiosyncrasies of any particular logic synthesis tool. Since all three tools use different heuristics, it is unlikely that they all have the same limitations. For example, the BOLD technology mapper only used nand or nor gates and inverters. DECAF used both simple nand and nor gates, as well as some complex gates. MISII, on the average, used the most complex gates of any of these tools. All tools use somewhat different techniques to do the minimization as well. These differences represent the use of different heuristics by the three tools.

The PLA examples were run through all three logic synthesis tools. All the BLIF examples were run using MISII. However, DECAF was not used on any of the BLIF examples because it cannot read BLIF format descriptions. BOLD was run on only the smallest of the BLIF examples because it often crashes and the running time of even medium-sized examples is prohibitively high.

Table 6.1 contains a description of each example circuit. Each entry names the original MCNC benchmark example that was the source of the circuit and the logic synthesis tool used to generate the circuit. It also shows the size, as measured by the number of transistors, the number of stages in the slowest path, and the delay in nanoseconds for the slowest path for each of the examples in the test case set before and after transistor sizing has been done. Thirty circuits were produced by this method, ranging in size from 140 to 2130 transistors, with an average size of 350 transistors. The worst case path delay of these circuits ranged from 5.2 to 25.2 nanoseconds, with an average worst case path delay of 11.2 nanoseconds. The delay values assume a 3 micron CMOS process from MOSIS[Tom88]. However, since no mask layout for these circuits was done, the parasitic wiring capacitances and resistances are not available and so were not included in the delay values. If these values had been available, they could have been used in the delay analysis, though. More detailed information on each example can be found in Appendix B.

6.2 Experiments

6.2.1 Experiment One

The first experiment runs each example using all the implemented transformations and the standard heuristic to minimize delay. The objective of the experiment is to evaluate the type of improvement possible and its costs on a set of representative circuits. The set of test cases are broad so that the results from this experiment should give a good indication of how well the techniques work. Several measurements are made to evaluate the effectiveness of the transformation techniques. Two measures, the change in the number of transistors and the change in delay of the slowest path, give an absolute measurement of the improvement to the circuit. A third measure, the change in the product AT^2, where A is the number of transistors and T is the worst case path delay through the circuit, gives a measure of the efficiency of the circuit. This product is used as a measure of efficiency because theoretical models for VLSI show a trade-off between the area and the square of the time[Ull84]. Therefore, a decrease in this product represents a better trade-off in area and delay than in the original circuit. In this experiment, the transistors sizes of each gate are allowed to vary as needed to improve its ability to drive large loads. The experiment is illustrated at the top of Figure 6.2.

In the first step of the experiment, the transistors in each gate were sized for minimum delay using AESOP. Recall that transistor sizing matches the size of the transistors within the gate with the load it must drive; gates with large loads are made larger to produce more current and thus drive the load faster. Circuit restructuring and transistor sizing are not orthogonal methods for reducing circuit delay. So, without an explicit transistor sizing step, some changes from applying a transformation may gain additional speed-ups because the fixed-size transistors are better suited for the particular load than before the transformation was applied. To remove this variability, transistor sizing is applied before and after circuit restructuring, and the *sized* netlist are compared to measure the effects of POLO on the circuit. While this

example name	generated by	number of transistors	number of stages	max. path delay(ns) unsized	max. path delay(ns) sized
5xpl	DECAF	360	7	8.0	6.0
9symml	MISII	518	9	11.2	9.2
9symml_b	BOLD	556	28	22.8	19.7
C880	MISII	1136	19	28.2	23.0
apex6	MISII	2130	5	17.6	14.5
apex7	MISII	658	7	12.8	12.0
apex7_b	BOLD	734	18	12.6	11.5
b9	MISII	388	7	7.8	7.7
b9_b	BOLD	364	11	6.0	5.4
duke2b	BOLD	1498	25	15.7	13.7
duke2bm	BOLD+MISII	1064	15	13.6	12.7
duke2d	DECAF	1198	11	14.5	14.0
duke2m	MISII	1138	13	16.8	14.3
f51m	MISII	370	21	25.2	25.2
f51m_b	BOLD	278	12	9.6	7.9
misex1d	DECAF	180	5	5.9	6.1
misex1m	MISII	282	7	8.3	7.2
misex2b	BOLD	416	11	7.3	6.8
misex2bm	BOLD+MISII	300	5	6.9	6.0
misex2d	DECAF	294	4	6.7	5.9
misex2m	MISII	282	7	9.2	7.7
rd53	DECAF	132	9	6.5	6.2
rd53m	MISII	140	7	6.0	5.2
rd53bm	BOLD+MISII	252	8	7.7	6.8
rd84b	BOLD	510	16	16.9	13.7
rd84bm	BOLD+MISII	417	15	16.5	13.2
rd84d	DECAF	561	10	11.9	11.3
rd84m	MISII	594	9	12.3	10.3
rot	MISII	1896	16	25.1	18.4
z4ml	MISII	162	7	8.0	6.8

Table 6.1: Experiment Test Cases

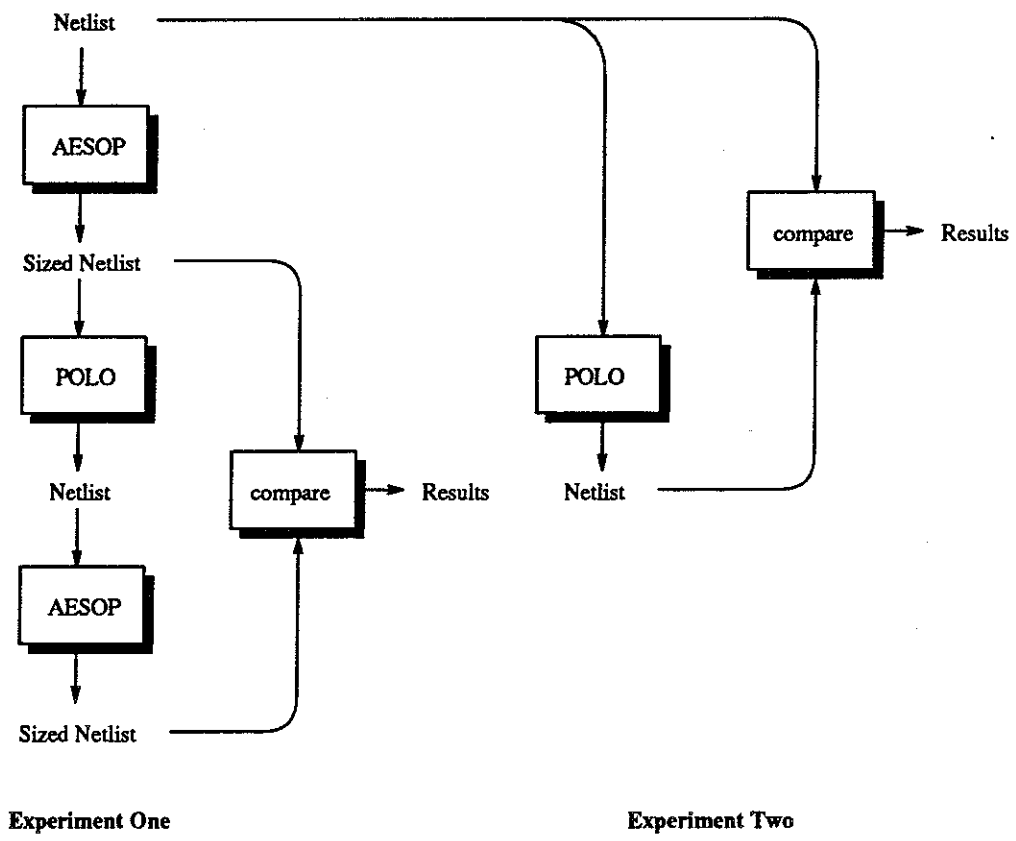


Figure 6.2: Experiments

method does not completely remove the effects of transistor sizing, it does minimize these effects. Also, some of the heuristics assume that transistor sizing is done.

AESOP uses an RC model to measure the delay, and associates with each gate a scale factor, S. Aesop then uses a quasi-Newton method that uses an approximate second order model of the delay function to iteratively compute the optimal transistor sizes to meet the desired timing constraints. In the experiment, AESOP was run subject to a maximum width limit of 30 λ and a minimum width limit of 3 λ for each transistor in the circuit. Also, some of the heuristics in POLO assume that transistor sizing will be applied to the circuits.

This type of use for POLO, where transistor sizing is used, is its expected use in a silicon compiler system that utilizes module generators to build the layout. These module generators would have control over the sizes of transistors used in the layout.

6.2.2 Experiment Two

A second experiment was run that differs from the first experiment only in that the transistor sizes of all the gates were fixed to a default size. Rather than using the sized circuit from AESOP, the unsized circuit produced by one of the logic synthesis tools is used directly as input. This experiment is consistent with POLO's use in a standard cell based synthesis system, where only fixed sized cells are available. The objective of this experiment is the same as in the first experiment, namely to evaluate the type of improvement possible and its costs. The same type of measurements were made in this experiment: change in number of transistors, change in slowest path delay, and change in the product AT^2. This experiment is illustrated at the bottom of Figure 6.2.

6.3 Experimental Results

6.3.1 Implementation Restrictions

The implementation of POLO differs from the techniques described in a small, but notable way. Two transformations, factoring and simplification, were not implemented because of the small size of the gates in the gate library used to map the gates. Most of the gates are so small that there exists only one reasonable way to factor the gate. For simplification, the size of the gates combined with the fact that the operation is already performed by the logic synthesis tools, prevents any gain from this technique. One other transformation, logic dual replacement, was not implemented. The technology mappers of MisII and Decaf already do a good job at minimizing the number of inverters, so further improvement is not likely using this transformation. These restrictions are due only to the set of input circuits.

Gate decomposition, while implemented, has only limited use in the experimental test cases. Its use is limited because the particular target gate library, MCNC Lib2, does not contain very many large gates.

6.3.2 Results of Experiment One

The results of the first experiment are shown in Figures 6.3 through 6.5. The effect of the transformations on the delay of the circuits are shown in Figure 6.3. This bar graph shows the normalized change in delay between the original circuit and the circuit after it has been optimized by Polo. The values shown for each example are the worst case path delay expressed as a percentage of the worst case path delay of the original circuit. Both the original and optimized circuit have been run through Aesop for transistor sizing, so this speed-up is in addition to the speed-up due to transistor sizing. The bottom entry represents the average for all the examples and shows an average decrease in the delay of the slowest path of 24%. In all cases these was a decrease in the maximum path delay, with two examples with about a fifty percent speed-up. Analyzing the four examples that had less than a ten percent speed-up shows that the slowest path in the final circuit had undergone path resynthesis. However, in these cases, the resulting speed-up from the transformation was not large, so the overall reduction in circuit speed was less than ten percent. The numerical data for all the examples is also shown in Table 6.2.

The bar graph in Figure 6.4 shows the new transistor count as a percentage of the original transistor count, as determined by dividing the new transistor count by the original transistor count. The average increase in transistor count was 20%. However, one example shows an area increase of about 100%. In this example, misex1m, the large area increase was due to a large percentage of critical paths relative to the total number of overall paths. When the average of all the examples except the one with a very large increase is taken, the average area increase is only 20%. In sharp contrast to the previous example, six other examples actually showed slight decreases in area. In these cases, this decrease was a result of the path resynthesis operation. When the transformation is applied, the old circuitry that was duplicated during resynthesis

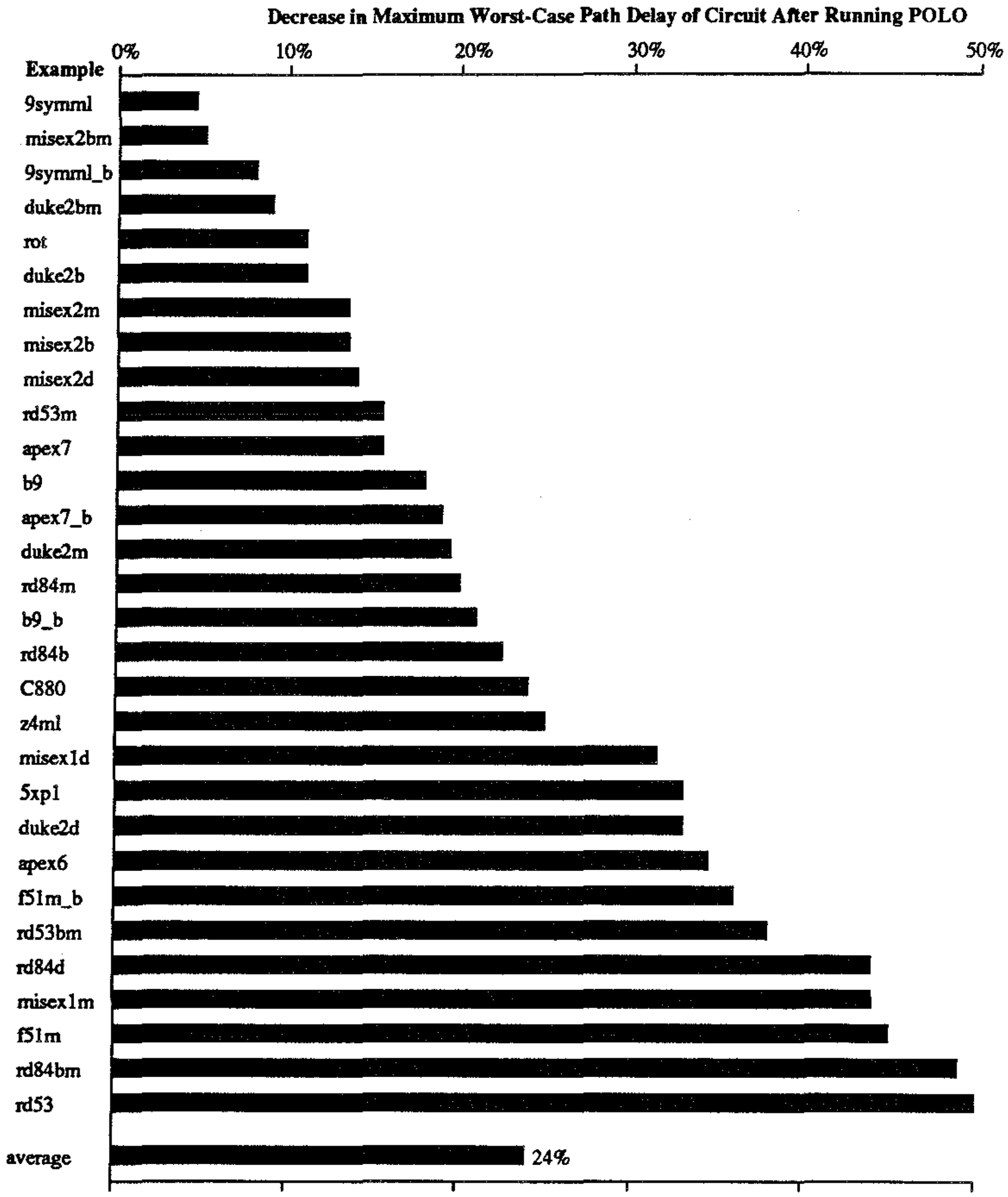


Figure 6.3: Summary of Experiment 1, Change in Delay of Slowest Path

is checked to make sure it is still driving some part of the circuit, otherwise the parts that are no longer used are removed from the circuit. If many of the gates are only used to compute the function at the cut-point, they will be deleted after the replacement circuitry is added. This occurrence is exactly what happened in the six examples that showed an area decrease. The numerical data for all the examples is also shown in Table 6.2.

The bar graph in Figure 6.5 shows the optimized circuits' AT^2 product, expressed as a percentage of the original circuits' AT^2 product. The average decrease was 28%. Only five examples showed an increase at all, and only one example had an increase of greater than 15%. For these examples, this data indicates that the speed-up was gained at a much greater cost of area. So, while the circuits were faster, they were not as efficient as before. However, most of the examples showed a reduction in AT^2, with six examples showing a reduction of at least 50%. The numerical data for all the examples is shown in Table 6.3. Complete data for the experiment can also be found in Appendix B.

6.3.3 Results of Experiment Two

Since POLO assumed the availability of transistor sizing in the way it implemented the transformations, the results produced by this experiment are not as good as in experiment one. However, there still was an average decrease in path delay of 20%, and an average decrease in AT^2 of 9%. There was a much larger area increase, however, averaging 35%.

The bar graph in Figure 6.6 shows the speed-up for each of the examples. In general, the speed-ups were somewhat less than in the sized case, although in a few examples, the percentage of decrease in delay was greater in experiment one. However, none of the examples are as fast as the examples in experiment one. Two examples were not able to speed up the most critical path at all. Although other critical paths were sped up, there was no change in the slowest path. This situation occurred because there was no way to compensate for the additional loads placed on some gates caused by some of the transformations, such as path resynthesis. Usually transistor sizing is used, but in this experiment it was not allowed.

The change in transistor count is shown in the bar graph in Figure 6.7. The area increase is greater than in experiment one because the same timing constraints were used. Since the original circuits in this experiment were slower than the original

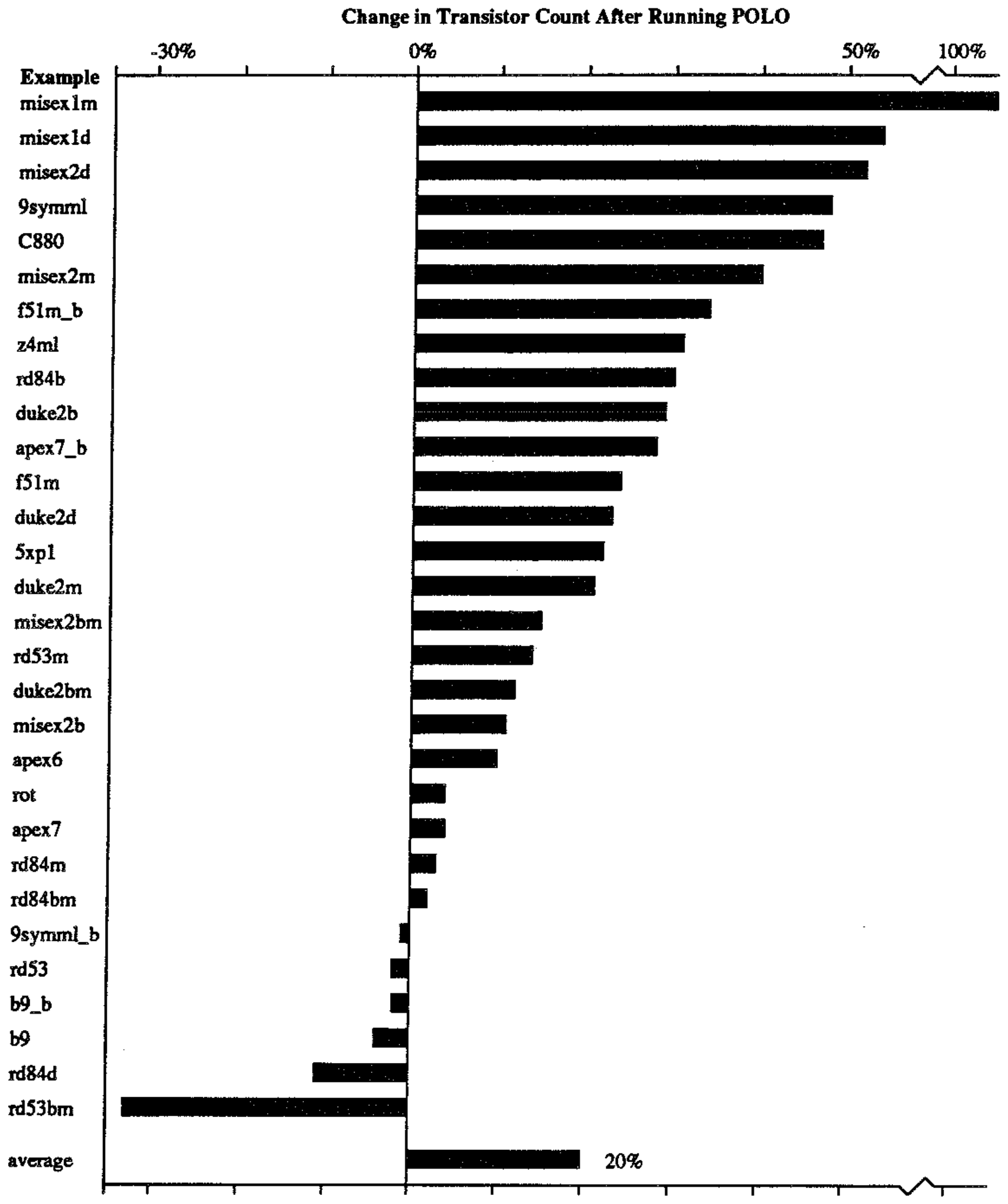


Figure 6.4: Summary of Experiment 1, Change in the Transistor Count

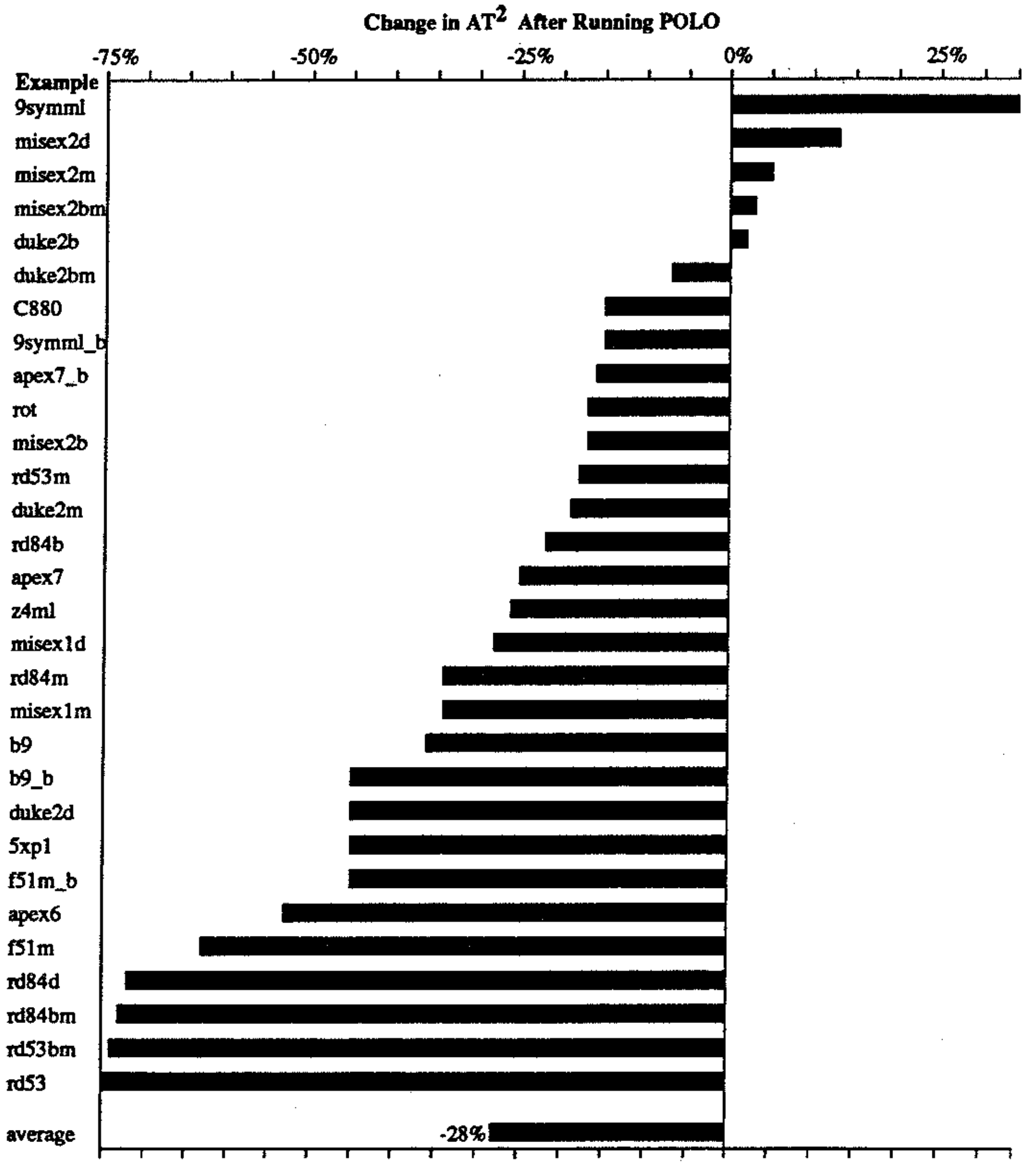


Figure 6.5: Summary of Experiment 1, Change in AT^2

example	number of transistors			max. path delay (ns.)		
name	before	after	% change	before	after	% change
5xp1	360	438	22	6.0	4.0	–33
9symml	518	764	47	9.2	8.9	–4
9symml_b	556	554	–0.36	19.7	18.2	–8
C880	1136	1666	47	23.0	17.5	–24
apex6	2130	2334	9.6	14.5	9.5	–34
apex7	658	682	3.6	12.0	10.2	–15
apex7_b	734	938	28	11.5	9.3	–19
b9	388	374	–3.6	7.7	6.3	–18
b9_b	364	356	–2.2	5.4	4.2	–22
duke2b	1498	1938	29	13.7	12.2	–11
duke2bm	1064	1196	12	12.7	11.5	–9
duke2d	1198	1480	24	14.0	9.4	–33
duke2m	1138	1328	21	14.3	11.7	–19
f51m	372	462	24	25.2	13.8	–45
f51m_b	278	372	34	7.9	5.1	–36
misex1d	180	278	54	6.1	4.1	–32
misex1m	178	368	107	7.2	4.1	–44
misex2b	416	462	11	6.8	5.8	–14
misex2bm	300	344	15	6.0	5.6	–5
misex2d	294	448	52	5.9	5.1	–14
misex2m	282	396	40	7.7	6.6	–13
rd53	132	130	–1.5	6.2	3.1	–50
rd53m	140	160	14	5.2	4.4	–15
rd53bm	252	168	–33	6.8	4.2	–38
rd84b	510	664	30	13.7	10.5	–23
rd84bm	416	426	2.4	13.2	6.7	–49
rd84d	560	500	–11	11.3	6.3	–44
rd84m	594	612	3.0	10.3	8.2	–20
rot	1896	1976	4.2	18.4	16.4	–11
z4ml	162	212	31	6.8	5.1	–25

Table 6.2: Experiment1 Results: Area, Delay

example name	number of stages			AT^2		
	before	after	% change	before	after	% change
5xp1	7	5	–29	1.3×10^4	7.1×10^3	–45
9symml	7	10	43	4.4×10^4	5.9×10^4	35
9symml_b	7	10	43	2.2×10^5	1.8×10^5	–15
C880	23	15	–35	6.0×10^5	5.1×10^5	–15
apex6	11	7	–36	4.5×10^5	2.1×10^5	–53
apex7	9	11	22	9.5×10^4	7.1×10^4	-25
apex7_b	18	11	–39	9.6×10^4	8.1×10^4	–16
b9	5	7	40	2.3×10^4	1.5×10^4	–36
b9_b	7	7	0	1.1×10^4	9.3×10^3	–12
duke2b	22	21	–5.5	2.8×10^5	2.9×10^5	1.5
duke2bm	12	8	–33	1.7×10^5	1.6×10^5	-6.9
duke2d	15	13	–13	2.4×10^5	1.3×10^5	–45
duke2m	13	7	–23	2.3×10^5	1.9×10^5	-19
f51m	5	16	180	2.4×10^5	8.9×10^4	–63
f51m_b	13	6	–54	1.8×10^4	9.6×10^3	–45
misex1d	6	4	–33	6.6×10^3	4.8×10^3	–28
misex1m	7	7	0	9.3×10^3	6.1×10^3	–34
misex2b	10	5	–50	1.9×10^4	1.6×10^4	–17
misex2bm	8	5	–38	1.1×10^4	1.1×10^4	3.4
misex2d	7	5	–29	1.0×10^4	1.2×10^4	13
misex2m	6	4	–33	1.7×10^4	1.8×10^4	5.1
rd53	9	3	–67	5.1×10^3	1.3×10^3	–75
rd53m	7	5	–29	3.8×10^3	3.1×10^3	–18
rd53bm	7	8	14	1.2×10^4	3.0×10^3	–74
rd84b	12	11	8.3	9.5×10^4	7.4×10^4	–22
rd84bm	9	9	0	7.3×10^4	1.9×10^4	–73
rd84d	9	10	11	7.1×10^4	2.0×10^4	–72
rd84m	11	7	–36	6.2×10^4	4.1×10^4	–34
rot	15	17	13	6.4×10^5	5.3×10^5	–17
z4ml	8	9	13	7.5×10^3	5.5×10^3	–26

Table 6.3: Experiment1 Results: Number of Stages on Critical Path, AT^2

circuits in experiment one, more paths had to be transformed, requiring greater area. One example, 9symml_b, grew particularly large, by over two hundred percent. This large area increase resulted from a combination of several applications of path resynthesis and because there was only a single output in the circuit, so the gates replaced by the resynthesis operation were still needed by other parts of the circuit.

Finally, the efficiency, as measured by AT^2, is shown in Figure 6.8. The average decrease in efficiency is much smaller, and as seen in the Figure, there are several examples that showed a large decrease in efficiency.

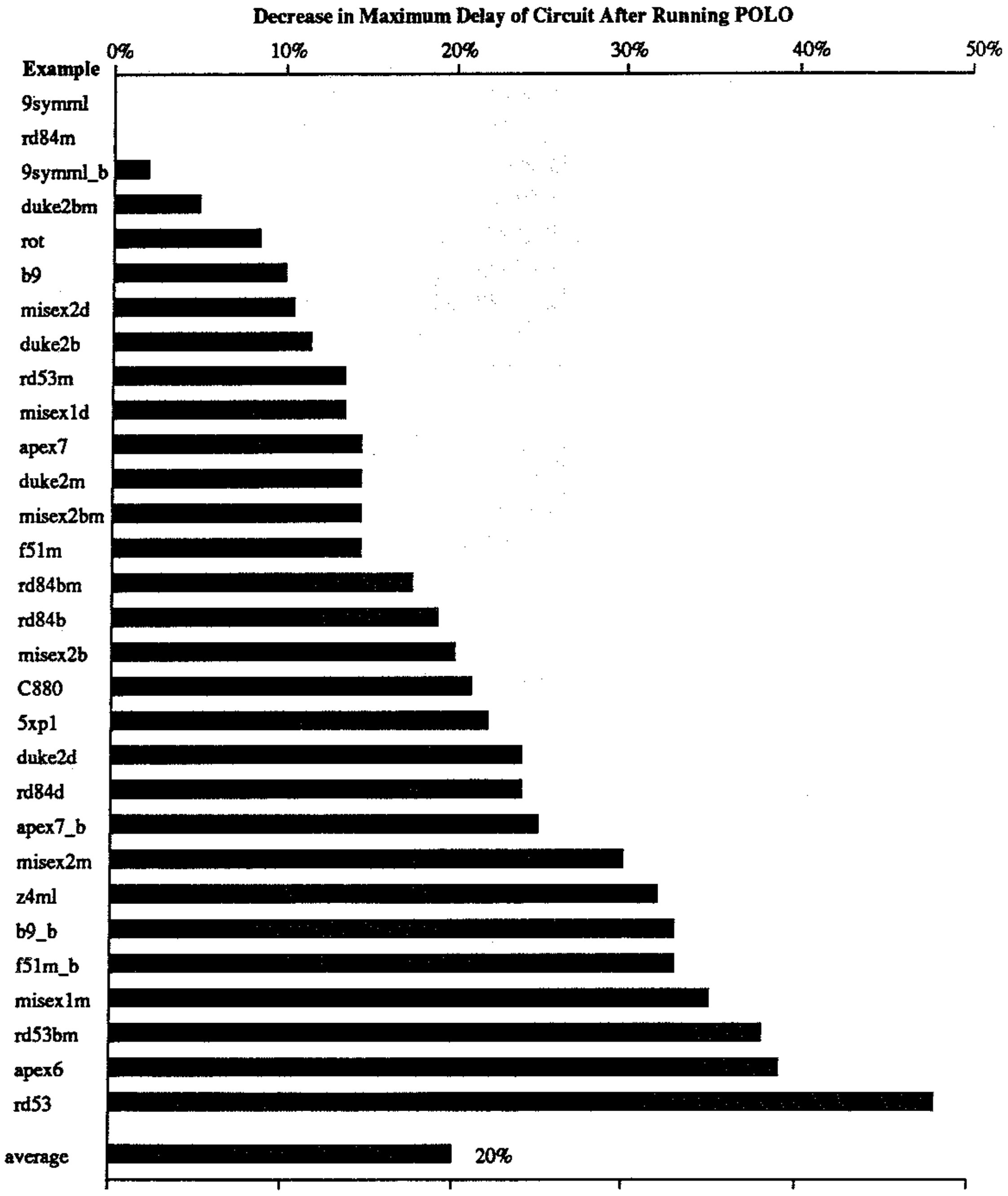


Figure 6.6: Summary of Experiment 2, Change in Delay of Slowest Path

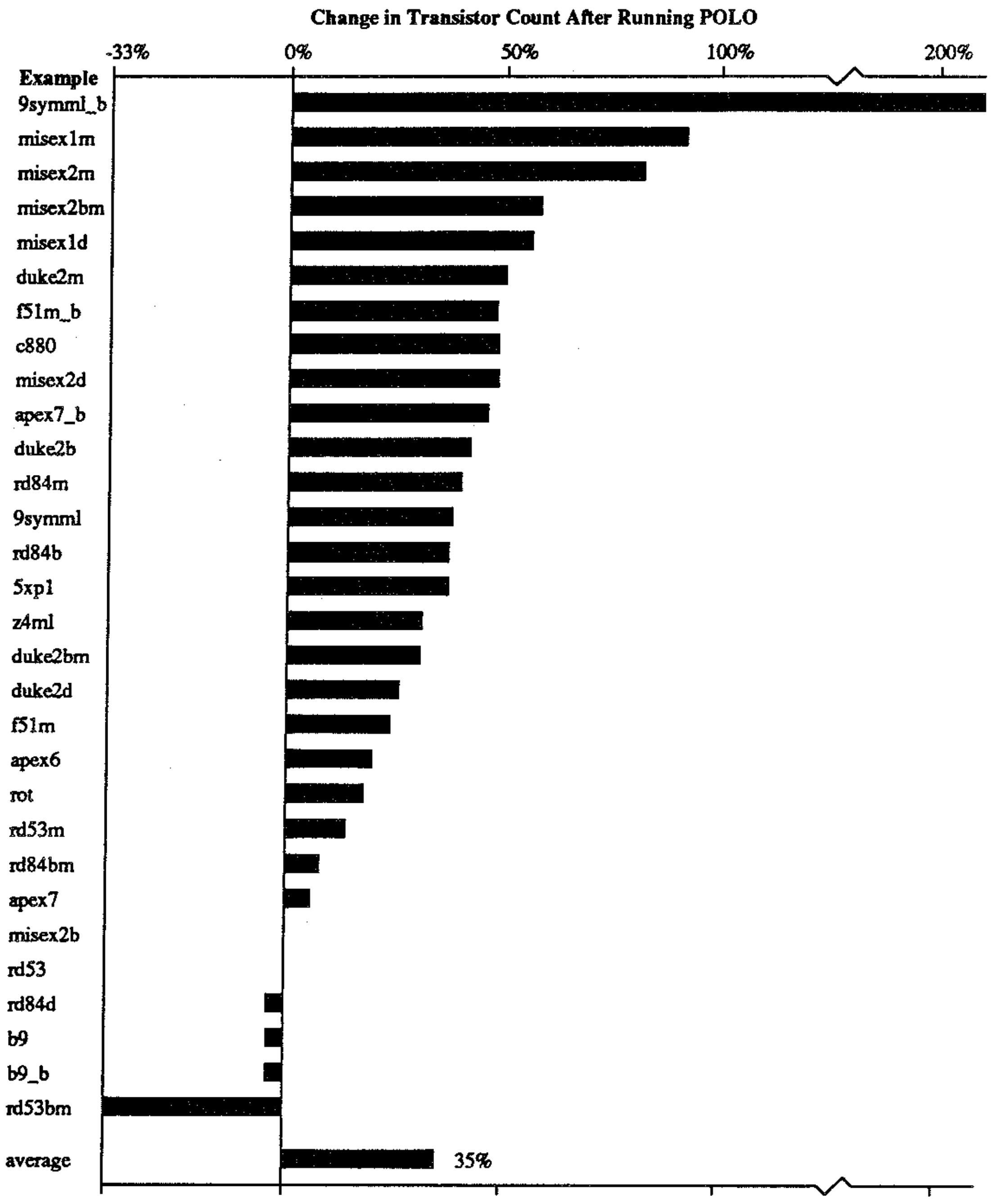


Figure 6.7: Summary of Experiment 2, Change in Transistor Count

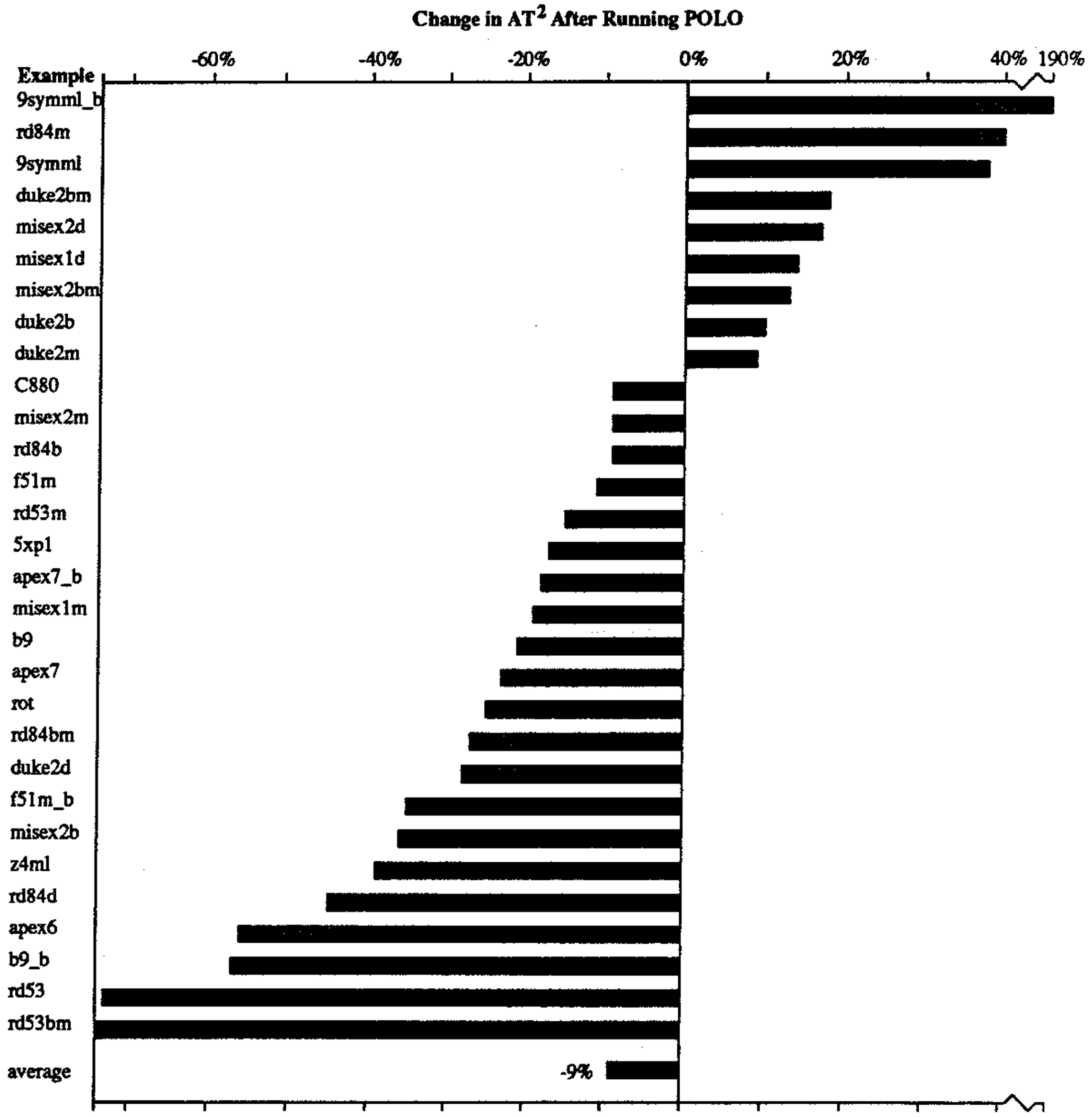


Figure 6.8: Summary of Experiment 2, Change in AT^2

example name	number of transistors			max. path delay (ns.)		
	before	after	% change	before	after	% change
5xpl	360	492	37	8.0	6.2	–22
9symml	518	714	38	11.2	11.8	0
9symml_b	556	1706	210	22.8	22.3	–2
C880	1136	1666	47	28.2	22.2	–21
apex6	2130	2560	20	17.6	10.7	–39
apex7	658	700	6	12.8	10.9	–15
apex7_b	734	1074	46	12.6	9.4	–25
b9	388	374	–4	7.8	7.0	–10
b9_b	364	350	–4	6.0	4.1	–33
duke2b	1498	2120	42	15.7	13.8	–12
duke2bm	1064	1398	31	13.6	12.9	–5
duke2d	1198	1500	25	14.5	11.0	–24
duke2m	1138	1716	51	16.8	14.3	–15
f51m	372	462	24	25.2	21.4	–15
f51m_b	278	408	47	9.6	6.4	–33
misex1d	180	278	54	5.9	5.1	–14
misex1m	178	340	91	8.3	5.4	–35
misex2b	416	416	0	7.3	5.9	–20
misex2bm	300	468	56	6.9	5.8	–15
misex2d	294	432	47	6.7	6.0	–11
misex2m	282	522	85	9.2	6.4	–30
rd53	132	132	0	6.5	3.3	–48
rd53m	140	160	14	6.0	5.2	–14
rd53bm	252	168	–33	7.7	4.8	–38
rd84b	510	698	37	16.9	13.7	–19
rd84bm	416	450	8	16.5	13.6	–18
rd84d	560	542	–3	11.9	9.0	–24
rd84m	594	834	40	12.3	12.3	0
rot	1896	2252	19	25.1	23.0	–8
z4ml	162	212	31	8.1	5.5	–32

Table 6.4: Experiment2 Results: Area, Delay

example	number of stages			AT^2		
name	before	after	% change	before	after	% change
5xpl	7	7	0	2.3×10^4	1.9×10^4	-17
9symml	8	12	50	6.5×10^4	8.9×10^4	38
9symml_b	28	22	–14	2.9×10^5	8.5×10^5	190
C880	23	15	–35	9.0×10^5	8.2×10^5	-9
apex6	11	5	-55	6.6×10^5	2.9×10^5	–55
apex7	9	9	0	1.1×10^5	8.3×10^4	–23
apex7_b	16	11	–31	1.2×10^5	9.6×10^4	–18
b9	9	5	–44	2.3×10^4	1.8×10^4	–21
b9_b	8	3	–63	1.3×10^4	5.8×10^3	–56
duke2b	22	13	–41	3.7×10^5	4.0×10^5	10
duke2bm	15	15	0	2.0×10^5	2.3×10^5	18
duke2d	9	9	0	2.5×10^5	1.8×10^5	–28
duke2m	13	11	–15	3.2×10^5	3.5×10^5	9
f51m	5	16	180	2.4×10^5	2.1×10^5	–11
f51m_b	12	7	–42	2.6×10^4	1.7×10^4	–35
misex1d	5	5	0	6.3×10^3	7.2×10^3	14
misex1m	7	5	–29	1.2×10^4	9.8×10^3	–19
misex2b	10	5	–50	2.2×10^4	1.4×10^4	–36
misex2bm	7	6	–14	1.4×10^4	1.6×10^4	13
misex2d	7	4	–43	1.3×10^4	1.6×10^4	17
misex2m	7	5	–29	2.4×10^4	2.2×10^4	-9
rd53	9	4	–56	5.5×10^3	1.5×10^3	–73
rd53m	7	5	–29	5.0×10^3	4.2×10^3	–15
rd53bm	7	5	–29	1.5×10^4	3.8×10^3	–74
rd84b	12	13	8	1.5×10^5	1.3×10^5	-9
rd84bm	7	10	43	1.1×10^5	8.3×10^4	–27
rd84d	14	10	–29	7.9×10^4	4.4×10^4	–45
rd84m	9	10	11	9.0×10^4	1.3×10^5	40
rot	15	14	7	1.2×10^6	1.2×10^6	–0
z4ml	8	8	0	1.1×10^4	6.4×10^3	39

Table 6.5: Experiment2 Results: Number of Stages on Critical Path, AT^2

Chapter VII
Conclusions

7.1 Summary and Conclusions

We have investigated automatic restructuring as a method to reduce delays in the circuit. The results from the first experiment are very promising. Although further improvement by tuning the heuristics is possible, the results already show significant improvements in speed and efficiency of the circuits by restructuring, averaging 24% and 28% respectively, with a somewhat modest increase in the overall area of the circuits, averaging 20%. Not counting the two circuits that had unusually large area increases, the average area increase was only 18%.

Some circuits, even after having all the transformations applied to the critical paths, showed less than a ten percent speed-up. So, although large speed-ups are possible, there are some examples that do not gain much from restructuring. One of the causes of this problem is the function being computed. For example, two of the examples showing little speed-up had a high fan-in tree structure with only a single output, which limited the amount of change possible in the circuit. Another cause of limited speed-up is the limits of the logic minimization and technology mapping used in the path resynthesis operation. The path produced was as good as possible using the current minimization and mapping heuristics so that further improvement is possible only if logic minimization and technology mapping methods are improved.

The greedy approach for applying the transformations worked well in almost all test cases. However, it tends not to work as well when there is a large number of paths that have about the same delay. The two examples from experiment one where this case occurred showed large area increases. However, it is not clear if other techniques would work well in this situation either.

The area decrease that occurred in several of the examples from experiment one was a surprising result. Since this effect occurred only in the smaller circuits, one cause of this effect is the high percentage of the total paths that are critical. Nonetheless, it does indicate that some of the current area minimization heuristics in logic synthesis may not always produce the smallest circuit, and the transformation responsible for the area savings, path resynthesis, may be useful in some circumstances for area optimization. Also, since this effect occurred in examples produced by each logic synthesis tool, the effect is not limited to any particular one of them.

It is clear from the results of the experiments that the heuristics in POLO cannot be the same when using POLO with transistor sizing and using POLO without transistor sizing. The heuristics were tuned for working with circuit that had undergone transistor sizing, and did very well in decreasing delay and increasing efficiency in the first experiment. However, the results from the second experiment where the circuits had not undergone transistor sizing were disappointing. Although there were decreases in delay, the cost in area was much greater than in the first experiment and the efficiency of the circuits did not improve as much on the average. The results of experiment two do not indicate that the basic idea of a greedy application of transformations will not improve the delay and efficiency of the circuit, since many of the examples did show improvement. However, since several example circuits showed reduced efficiency or no improvement in delay, the heuristics need to be tuned differently for working with unsized gates.

The new transformation, path resynthesis, proved to work well when used with multiple cut points and resynthesis operations per path. As mentioned above, in several cases, it actually reduced the area of the circuit. In general, it provided a good trade-off between area increase and delay decrease as long as the number of times it is applied is not too large. The two examples with large area increase show the area penalty for a large number of applications. Even though most of the circuits did not have very long paths through the circuit, there was a difference between using a single cut point and multiple cut points per path.

7.2 Future Research

Some interesting ideas have developed as an outgrowth of this work and bear further investigation. These topics are described below.

7.2.1 Further Tuning of Heuristics

The current heuristics produce good speed-ups. However, it should be possible to reduce delay even further. In addition, gate collapsing can be applied as an area saving transformations off of the critical path so that the overall area increase will be smaller or the area will decrease, and so that all examples will show smaller AT^2 products. As mentioned above, the heuristics also need to be tuned to work better when transistor sizing is not available.

An interesting experiment would be to run the tool on circuits created using a larger set of gates than in MCNC Lib2. This experiment would also allow further tuning of the heuristics by studying the effects of using much larger gates in the circuits and will determine whether or nor the same heuristics can be used.

The effects of the transformations on the power usage in the circuits are not measured. Another interesting experiment is to measure the effects of the transformations on the power use of the circuit and use this information to guide the heuristics.

7.2.2 Improved Technology Mapping

One of the results of this research is the improvement in speed using fan-in reordering within a tree of gates in the circuit. This transformation reduces delay because during the technology mapping phase the correct delay information is not available. Therefore, when deciding how to break apart a gate, the arrival times are not considered, and it is done arbitrarily and many times incorrectly. In the performance optimization step, this information is available, so trees on the critical path can be remapped using this delay information. It should be possible to move this operation into technology mapping. Moving this operation requires two changes in the basic technology mapping method. First, the order of mapping of the trees now becomes important. Mapping must begin at the trees that use the primary inputs and proceeds in a breadth-first fashion through the circuit. This order insures that correct delay information will be available when determining the best way to split a tree. Second, a more accurate delay model, such as the RC model, must be incorporated into the mapping operation. Since these changes are a modification to technology mapping, they were not included in the main research of this dissertation. However, this idea will be investigated shortly.

7.2.3 Cross-tree Optimizations

The principal deficiency of the technology mapping techniques is the the local aspect of all the operations; no global improvements are possible. This deficiency is caused by the local nature of the technology mapping algorithms; the Boolean network is split into trees which are mapped individually. Even the modification described in the previous section does not overcome this problem. Some of the transformations are initial attempts at the cross-tree optimizations, but further research is needed to investigate further ways of reducing delay through this technique.

7.2.4 Logic Decomposition

As can be seen from the improvement in delay through path resynthesis, current decomposition techniques for building the Boolean network do a good job at area minimization, but are suboptimal for delay optimization. Some early work used simple delay information to help guide the use of algebraic (weak) division of functions in the Boolean network[BH85], but no further work seems to have been done in this area. One area of future research is to model and study the trade-offs between the sharing of common minterms by different Boolean equations and the effects on area and delay.

7.2.5 Testability Issues

The input circuits are assumed to be 100% testable before transformation. Appendix C shows that most of the transformations do not effect the testability of the circuit. One extension of this work is to include transformations that improve the testability by removing redundancies. Some work has already been done in this area that only considered improved testability[BBL89], but it may be possible to combine the goals of improved performance and improved testability together.

BIBLIOGRAPHY

[BBH+86] K. Bartlett, R. Brayton, G. Hachtel, R. Jacoby, A. Sangiovanni-Vincentelli, and Wang. A. Multilevel Logic Minimization Using Implicit Don't Cares. In *Proceedings of 1986 ICCD*, pages 552–557, 1986.

[BBH+88] K.A. Bartlett, R.K. Brayton, G.D. Hachtel, R.M. Jacoby, C.R. Morrison, R.L. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang. Multilevel Logic Minimization Using Implicit Don't Cares. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-7(8):723–740, June 1988.

[BBL89] D. Bryan, F. Brglez, and R. Lisanke. Redundancy Identification and Removal. In *Proceedings of 1989 International Workshop on Logic Synthesis*. MCNC/ACM SIGDA, May 1989.

[BCD+88] R.K. Brayton, R. Camposano, G. DeMicheli, R.H.J.M. Otten, and J. van Eijindhove. The Yorktown Silicon Compiler. In D.D. Gajski, editor, *Silicon Compilation*, chapter 7, pages 204–310. Addison-Wesley, 1988.

[BCDH86] K. Bartlett, W. Cohen, A. DeGeus, and G. Hachtel. Synthesis and Optimization of Multilevel Logic Under Timing Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-5(4):582–596, October 1986.

[Ber88] R. A. Bergamaschi. Automatic Synthesis and Technology Mapping of Combinational Logic. In *International Conference on Computer-Aided Design ICCAD-88 Digest of Technical Papers*, pages 466–469, 1988.

[BH85] K.A. Bartlett and G.D. Hachtel. Library Specific Optimization of Multilevel Combinational Logic. In *Proceedings of 1985 ICCD*, pages 411–415, 1985.

[BHJ+87] D. Bostick, G.D. Hachtel, R. Jacoby, M.R. Lightner, P. Moceyunas, C.R. Morrison, and D. Ravenscroft. The Boulder Optimal Logic Design System. In *Proceedings of 1987 ICCD*, pages 62–65, 1987.

[BHMSV84] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, 1984.

[BJ88] M.R.C.M. Berkelaar and J.A.G. Jess. Technology Mapping for Standard-cell Generators. In *International Conference on Computer-Aided Design ICCAD-88 Digest of Technical Papers*, pages 470–473, 1988.

[Bra83] D. Brand. Redundancy and Don't Cares in Logic Synthesis. *IEEE Transactions on Computers*, C-32(10):947–952, October 1983.

[BRSVW87] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang. MIS: A Multiple-level Logic Optimization System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(6):1062–1081, November 1987.

[BS89] R.K. Brayton and F. Somenzi. Boolean Relations. In *Proceedings of 1989 International Workshop on Logic Synthesis.* MCNC/ACM SIGDA, May 1989.

[DBG+84] J. Darringer, D. Brand, J. Gerbi, W. Joyner, and L. Trevillyan. LSS: A System for Production Logic Synthesis. *IBM Journal of Research and Development*, 28(5):537–545, September 1984.

[DeM87] G. DeMicheli. Performance-oriented Synthesis of Large-scale Domino CMOS Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(5):751–765, September 1987.

[DJBT81] J.A. Darringer, W.H. Joyner, C.L. Berman, and L. Trevillyan. Logic Synthesis through Local Transformations. *IBM Journal of Research and Development*, 25(4):272–280, July 1981.

[EG81] A. El Gamal. Two-dimensional Stochastic Model for Interconnections in Master Slice Circuits. *IEEE Transactions on Circuits and Systems*, CAS-28(2):127–138, February 1981.

[Elm48] W.C. Elmore. The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. *Journal of Applied Physics*, 19:55–63, January 1948.

[FD85] J.P. Fishburn and A.E. Dunlop. TILOS: A Polynomial Programming Approach to Transistor Sizing. In *International Conference on Computer-Aided Design ICCAD-85 Digest of Technical Papers*, pages 326–328, 1985.

[GBdAH86] D. Gregory, K. Bartlett, deGeus A., and G. Hachtel. SOCRATES: A System for Automatically Synthesizing and Optimizing Combinational Logic. In *Proceedings of 23rd Design Automation Conference*, pages 79–85, 1986.

[GD85] L.A. Glasser and D.W. Dobberpuhl. *The Design and Analysis of VLSI Circuits.* VLSI System Series. Addison-Wesley Publishing Company, Reading, Massachusetts, 1985.

[GK83] D. Gajski and R. Kuhn. Guest Editors' Introduction: New VLSI Tools. *IEEE Computer*, 16(12):11–14, December 1983.

[Hed84] K.S. Hedlund. Models and Algorithms for Transistor Sizing in MOS Circuits. In *International Conference on Computer-Aided Design ICCAD-84 Digest of Technical Papers*, pages 12–14, 1984.

[Hed87] K.S. Hedlund. AESOP: A Tool for Automated Transistor Sizing. In *Proceedings of the 24th Design Automation Conference*, pages 114–120, 1987.

[HJ88] G.D. Hachtel and R.M. Jacoby. Verification Algorithms for VLSI Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(5):616–640, May 1988.

[HJKM89] G.D Hachtel, R.M. Jacoby, K. Keutzer, and C. Morrison. On the Relationship Between Area Optimization and Multifault Testability of Multilevel Logic. In *Proceedings of 1989 International Workshop on Logic Synthesis*, May 1989.

[HK87] M. Hofmann and J.K. Kim. Delay Optimization of Combinational Static CMOS Logic. In *Proceedings of 24th Design Automation Conference*, pages 125–132, 1987.

[HL87] G.D. Hachtel and M.R. Lightner. Don't Care Conditions in Top Down Synthesis. In *International Conference on Computer-Aided Design ICCAD-87 Digest of Technical Papers*, pages 316–319, 1987.

[HMD77] W.R. Heller, W.F. Mikhail, and W.E. Donath. Prediciton of Wiring Space Requirements for LSI. In *Proceedings of 14th Design Automation Conference*, pages 32–42, 1977.

[Hor83] M. Horowitz. *Timing Models for MOS Circuits.* PhD thesis, Stanford University, December 1983.

[Jou83a] N.P. Jouppi. Timing Analysis for nMOS VLSI Circuits. In *Proceedings of 20th Design Automation Conference*, pages 411–418, 1983.

[Jou83b] N.P. Jouppi. TV: An nMOS Timing Analyzer. In *Proceedings of the Third Caltech VLSI Conference*, pages 71–85, 1983.

[Jou87] N.P. Jouppi. Timing Analysis and Performance Improvement of MOS VLSI Designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(4):650–665, July 1987.

[JTB+86] Jr. W.H. Joyner, L.H. Trevillyan, D. Brand, T.A. Nix, and S.C. Gundersen. Technology Adaptation in Logic Synthesis. In *Proceedings of the 23rd Design Automation Conference*, pages 94–100, 1986.

[Kah86] M. Kahrs. Matching a Parts Library in a Silicon Compiler. In *International Conference on Computer-Aided Design ICCAD-86 Digest of Technical Papers*, pages 169–172, 1986.

[Keu87] K. Keutzer. DAGON: Technology Binding and Local Optimization by DAG Matching. In *Proceedings of 24th Design Automation Conference*, pages 341–346. ACM/IEEE, 1987.

[KH89] M.S. Kotliar and K.S. Hedlund. Speed Optimization of Combinational Circuits. In *Proceedings of the 1989 International Workshop on Logic Synthesis.* MCNC/ACM SIGDA, 1989.

[KR89] K. Keutzer and D. Richard. Computational Complexity of Logic Synthesis and Optimization. In *Proceedings of 1989 International Workshop on Logic Synthesis.* MCNC/ACM SIGDA, May 1989.

[LBK88] R. Lisanke, F. Brglez, and G. Kedem. McMAP: A Fast Technology Mapping Procedure for Multi-level Logic Synthesis. In *IEEE International Conference on Computer Design*, 1988.

[Lea89] D. Lea. User's Guide to GNU C++ LIbrary. Free Software Foundation, May 1989.

[Lis88] R. Lisanke. Logic Synthesis and Optimization Benchmarks User Guide Version 2.0. Technical report, Microelectronics Center of North Carolina, PO Box 12889, Research Triangle Park, NC, December 16 1988.

[LKB87] R. Lisanke, G. Kedem, and F. Brglez. Decaf: Decomposition and Factoring for Multi-level Logic Synthesis, Version 1.0. Technical Report TR87-15, Microelectronics Center of North Carolina, August 1987.

[LS84] C.M. Lee and H. Soukup. An Algorithm for CMOS Timing and Area Optimization. *IEEE Journal of Solid-State Circuits*, SC-19(5):781–787, October 1984.

[Mat85] M.D. Matson. Optimization of Digital MOS VLSI Circuits. In *Proceedings of 1985 Chapel Hill Conference on VLSI*, 1985.

[MB88] H.-J. Mathony and U.G. Baitinger. CARLOS: An Automated Multi-level Logic Design System for CMOS Semi-custom Integrated Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-7(3):346–355, March 1988.

[MB89] P.C. McGeer and R.K. Brayton. Provably Correct Critical Paths. In *Proceedings of the 1989 Decennial Caltech VLSI Conference*, pages 119–142, 1989.

[MC80] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1980.

[McC86] E.J. McCluskey. *Logic Design Principles With Emphasis on Testable Semicuston Circuits.* Prentice-Hall, Englewood Cliffs, N.J., 1986.

[MEG86] D.P. Marple and A. El Gamal. Area-delay Optimization of Programmable Logic Arrays. In C.E. Leiserson, editor, *Advanced Research in VLSI - Proceedings of the Fourth MIT Conference*, pages 171–194, Cambridge, April 1986. MIT Press.

[MJD83] J. Mavor, M.A. Jack, and P.B. Denyer. *Introduction to MOS LSI Design.* Microelectronics Systems Design Series. Addison-Wesley Publishing Company, Reading, Massachusetts, 1983.

[OK88] F.W. Obermeier and R.H. Katz. Combining Circuit Level Changes with Electrical Optimization. In *International Conference on Computer-Aided Design ICCAD-88 Digest of Technical Papers*, pages 218–221. IEEE, IEEE Computer Society, November 1988.

[Ous83] J.K. Ousterhout. Crystal: A Timing Analyzer for nMOS VLSI Circuits. In *Proceedings of the Third Caltech Conference on VLSI*, pages 57–70, 1983.

[Ous84] J.K. Ousterhout. Switch-level Delay Models for Digital MOS VLSI. In *Proceedings of 21st Design Automation Conference*, pages 542–548, 1984.

[PL88] B.T. Preas and M.J. Lorenzetti, editors. *Physical Design Automation of VLSI Systems.* Benjamin/Cummings Company, Menlo Park, CA, 1988.

[RSVD85] R.L. Rudell, A. Sangiovanni-Vincentelli, and G. DeMicheli. A Finite-state Machine Synthesis System. In *Proceedings of International Symposium on Circuits and Systems*, pages 647–650, June 1985.

[Rub87] S.M. Rubin. *Computer Aids for VLSI Design.* VLSI System Series. Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

[Sas86] T. Sasao. MACDAS: Multi-level And-Or Circuit Synthesis Using Two-variable Function Generators. In *Proceedings of 23rd Design Automation Conference*, pages 86–93, 1986.

[Sho88] M. Shoji. *CMOS Digital Circuit Technology.* Computer System Series. Prentice-Hall, 1988.

[SP86] S. Sastry and A.C. Parker. Stochastic Models for Wirability Analysis of Gate Arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-5(1):52–65, January 1986.

[Str86] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley Publishing Co., 1986.

[SWBSV88] K.J. Singh, A.R. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli. Timing Optimization of Combinational Logic. In *International Conference on Computer-Aided Design ICCAD-88 Digest of Technical Papers*, pages 282–285. IEEE, IEEE Computer Society, November 1988.

[Tie89] M.D. Tiemann. User's Guide to GNU C++. Free Software Foundation, May 1989.

[Tom88] C. Tomovich. MOSIS - A Gateway to Silicon. *IEEE Circuits and Devices*, 4(2):22–23, March 1988.

[Tre87] L. Trevillyan. An Overview of Logic Synthesis Systems. In *Proceedings of 24th Design Automation Conference*, pages 166–172, 1987.

[Tya87] A. Tyagi. HERCULES: A Power Analyzer for MOS VLSI Circuits. In *International Conference on Computer-Aided Design ICCAD-87 Digest of Technical Papers*, pages 530–533. IEEE, 1987.

[Ull84] J.D. Ullman. *Computational Aspects of VLSI.* Priciples of Computer Science Series. Computer Science Press, Rockville, MD, 1984.

[Vee84] H.J.M. Veendrick. Short Circuit Dissipation of Static CMOS and Its Impact on the Design of Buffer Circuits. *IEEE Journal of Solid-State Circuits*, SC-19(4):468–473, August 1984.

[WE85] N.H.E. Weste and K. Eshraghian. *Principles of CMOS VLSI Desgin.* VLSI Systems Series. Addison-Wesley Publishing Company, Reading, Massachusetts, 1985.

Appendix A
RC Model

This appendix describes the RC delay model and discusses why it is a reasonable first order model for circuit delay. The next section introduces the important properties of the MOS transistor that are used in the model. Then, the second section discusses the theoretical basis for the RC model. In the third section, the sources of resistance and capacitance are described. Then, the final section concludes with the limitations of this model.

A.1 MOS Transistor

A MOS (Metal Oxide Semiconductor) transistor has three regions of operation: cut-off, resistive, and saturated. When a transistor is in the cut-off region, it acts, to the first order, as an open switch. There is only a very small leakage current through the transistor that is ignored. In the second region, the resistive region (also called non-saturated), the transistor acts as a voltage-controlled resistor. In the final region, the saturated region, the transistor acts as a current source, where the current is proportional to the square of the drain-to-source voltage. The first order equations for the drain-to-source current, I_{ds}, in the three regions are as follows:

$$I_{ds} = \begin{cases} 0 & \text{when } V_{gs} - V_t \leq 0 \quad \text{(cut-off)} \\ \beta((V_{gs} - V_t)V_{ds} - V_{ds}^2 & \text{when } 0 < V_{ds} < V_{gs} - V_t \quad \text{(resistive)} \\ \frac{\beta}{2}(V_{ds} - V_t)^2 & \text{when } 0 < V_{gs} - V_t < V_{ds} \quad \text{(saturation)} \end{cases} \tag{A.1}$$

The term, β, is called the *MOS gain factor*, and its value is given by the following equation:

$$\beta = (\frac{\mu\varepsilon}{t_{ox}})(\frac{W}{L}) \tag{A.2}$$

where μ is the effective mobility of the electrons in the channel, ε is the permittivity of the gate insulator, t_{ox} is the thickness of the gate insulator, W is the width of the channel, and L is the length of the channel. When the transistor is in the resistive region, the resistance of the channel, R_c, is given by the following equation:

$$R_c = k\frac{L}{W} \text{ where } k = (\mu\frac{\varepsilon_0\varepsilon_r}{t_{ox}}V_{gs} - V_t)^{-1} \tag{A.3}$$

A.2 Delay Model

Consider the simple circuit shown in Figure A.1, below, in which a simple inverter drives a capacitive load, C_L. Let V_{in} be equal to logic 0, and V_{out} be equal to logic one.

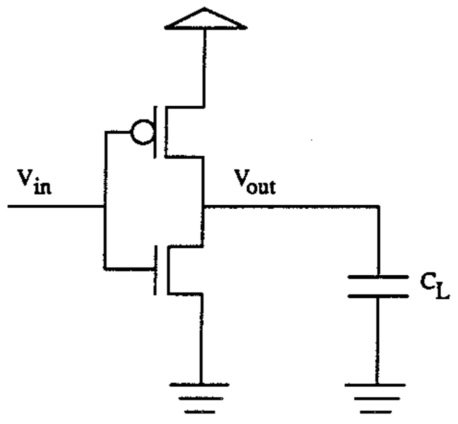


Figure A.1: Inverter Driving A Capacitive Load

Changing the value of V_{in} from logic 0 to logic 1 as a step function, turns off the p-type transistor (it goes into the cut-off region of operation), and turns on the n-type transistor, first operating in the saturation region, and then in the resistive region. As a result of this change, the capacitor, C_L, is discharged. The circuit in Figure A.1 can be represented by two equivalent circuits, shown in Figure A.2 corresponding to the two regions of operation that the n-type transistor goes through while discharging the capacitor.

The first equivalent circuit represents the original circuit when $V_{ds} \geq V_{gs} - V_t$, that is, when the n-type transistor is saturated. The second equivalent circuit represents the original circuit after $V_{ds} < V_{gs} - V_t$, when the n-type transistor is resistive. The following two equations describe the behavior of the two equivalent circuits:

$$C_L\frac{dV_0}{dt} + \frac{\beta}{2}(V_{dd} - V_t)^2 = 0 \tag{A.4}$$

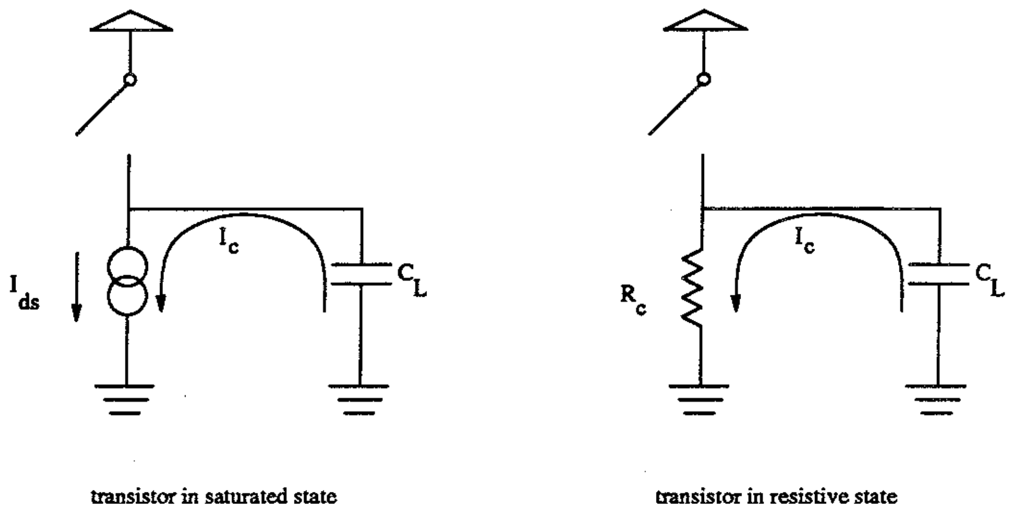


Figure A.2: Equivalent Circuits For Inverter Driving A Capacitive Load

$$C_L \frac{dV_0}{dt} + \beta(V_{dd} - V_t)V_0 - V_0^2 = 0 \tag{A.5}$$

To determine the equation to compute the time for the output to change from $0.9V_{dd}$ to $0.1V_{dd}$, we must integrate the two previous equations and sum the resulting two terms. The resulting equation for the time spent in saturation and resistive mode is:

$$t = \frac{2C_L(V_t - V_{dd})}{\beta(V_{dd} - V_t)} + \frac{C_L}{\beta(V_{dd} - V_t)} \frac{\ln(19V_{dd} - 20V_t)}{V_{dd}} \tag{A.6}$$

Assuming $V_t = 0.2V_{dd}$, the previous equation simplifies to:

$$t = \frac{4C_L}{\beta V_{dd}} \tag{A.7}$$

Notice that the term $\frac{4}{\beta V_{dd}}$ is a resistance. It is called the *effective resistance* of the transistor. Thus, the equation for time is simplified to:

$$t = R_{\text{eff}} C_L \tag{A.8}$$

This equation forms the basis for the RC delay model. Each stage in the circuit has an effective resistance and a capacitive load that it must drive. Thus for each stage, the delay is that value computed by equation A.8. Then the sum of successive stage delays gives the value of the delay through the circuit. A similar derivation can be used for the case when the p-type transistor is used to charge the capacitive load.

If we have an arbitrary complex CMOS gate, then the effective resistance for the gate can be approximated by finding the longest series transistor path in the gate. The resistance is then the number of transistors in the path multiplied by the effective resistance of one transistor. This method assumes that all the transistors have the same size.

A.3 Resistance and Capacitance Values

In the simplest version of the RC model, the resistance comes from the pull-up chain (for a rising transition) or the pull-down chain (for a falling transition) of the driving gate, and the capacitance comes from the gate-to-substrate (or gate-to-bulk) capacitance, gate-to-source capacitance, and gate-to-drain capacitance of the load gates. All other resistances and capacitances are ignored. These sources are the primary sources of resistance and capacitance for delay. Additional sources, called parasitic resistances and capacitances, are sometimes included. One source of parasitic resistance that sometimes can be large is the resistance of the wires between the driving gate and the load. This resistance can be especially large if diffusion or polysilicon wires are used, or if the metal wires are very long. These wires also have a parasitic capacitance associated with them. This capacitance is due to capacitance with the substrate or other layers, and from capacitive coupling with wires on the same layer. Another source of parasitic capacitance is from the diffusion regions inside the driving gate associated with transistors that are turned off but adjacent to the nodes in the pull-up or pull-down path.

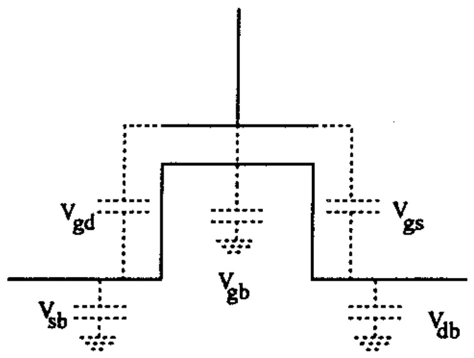


Figure A.3: Parasitic Capacitance of a MOS Transistor

In the data, the parasitics resistances and capacitances are not available. Since the parasitics are highly dependent on the mask layout of the circuit and no layout is available when the circuits in the test cases are optimized, there is no information on these parasitics. Assuming particular values for these parasitics can be done, if there is an implicit assumption about the implementation, such as standard cells or gate arrays. Since this work should be applicable to different implementations, no assumption about typical parasitics values can be made. However, if the wiring parasitics are available, they are included in the delay analysis.

A.4 Problems with RC Model

The RC model has several drawbacks. First, the effective resistance depends on the input waveform. To overcome this problem, different values are used for the resistance, depending on the input waveform. However, this solution still assumes a uniform trigger voltage. If varying beta ratios occur, this assumption is no longer valid. A more detailed model[Hor83, MEG86], modifies the RC model in order to a better job of modeling delay with slow-rising inputs. In this model, delay d, is given by the following equation:

$$d = \sqrt{(\tau_r \ln V_s)^2 + 2(1 - V_s)\tau_{\text{in}}\tau_{g_m}} \tag{A.9}$$

where

$$\begin{aligned}
\tau_r &= R_{\text{gate}} C_{\text{load}} \\
\tau_{g_m} &= G^{-1}_{m_{\text{gate}}} C_{\text{load}} \\
\tau_{\text{in}} &= R_{\text{prev}} C_{\text{prev}} \\
V_s &= \text{normalized switching voltage} \\
G_m &= g_{m_e} \tfrac{W}{L} \\
g_{m_e} &= \text{current gain for minimum-sized transistor}
\end{aligned}$$

Another problem with the RC model arises from unequal rise and fall times for a stage caused by different resistance values for n-type and p-type transistors which arise from the differences in the mobility of electrons and holes in the substrate. The worst case method, using the largest delay for each gate, is too pessimistic and results in values for path delay that are much worse than other models. This problem is especially noticeable in nMOS circuits. In CMOS circuits, the rise and fall times can be made nearly equal by sizing transistors properly, and therefore this problem is not significant for circuits discussed in this dissertation.

Appendix B
Raw Experimental Data

This appendix contains the raw numbers from running the experiments, which are shown in Table B.1, below. In this table, the first column is the example name and the second column contains the tool used to generate the example. The remaining columns in the table show the number of inputs and outputs, and the size of each circuit in terms of gates and transistors. The last column indicates if the original benchmark was a two-level description (PLA) or a multi-level description (BLIF). The tools used to generate the examples are DECAF, MISII, and BOLD. In addition, for a few examples, the logic decomposition and minimization of BOLD are combined with the technology mapping of MISII to produce circuits. Not all the tools were run on all the examples. DECAF can only accept two-level input descriptions, and BOLD requires excessive CPU times (on the order of several SPARC CPU-days) for all but the smallest example.

The raw data for the delay analysis of the examples from both experiment one and two is shown in Table B.2, below. In Table B.3, the raw data for the area measurements from experiment one and two is shown. Table B.4 contains the raw data for the measurements of efficiency, AT^2. These values are derived from Tables B.2 and B.3.

Example Name	Synthesis Tool	Inputs	Outputs	Gate	Transistor Count	Type
5xpl	DECAF	7	10	80	360	PLA
9symml	MISII	9	1	79	518	BLIF
9symml_b	BOLD	9	1	127	556	BLIF
C880	MISII	60	26	211	1136	BLIF
apex6	MISII	135	99	385	2130	BLIF
apex7	MISII	49	37	127	658	BLIF
apex7_b	BOLD	49	37	199	734	BLIF
b9	MISII	41	21	78	388	BLIF
b9_b	BOLD	41	21	95	364	BLIF
duke2b	BOLD	22	29	457	1498	PLA
duke2bm	BOLD/MISII	22	29	240	1064	PLA
duke2d	DECAF	22	29	259	1198	PLA
duke2m	MISII	22	29	229	1138	PLA
f51m	MISII	8	8	61	370	BLIF
f51m_b	BOLD	8	8	67	278	BLIF
misex1d	DECAF	8	7	40	180	PLA
misex1m	MISII	8	7	37	282	PLA
misex2b	BOLD	25	18	128	416	PLA
misex2bm	BOLD/MISII	25	18	70	300	PLA
misex2d	DECAF	25	18	69	294	PLA
misex2m	MISII	25	18	58	282	PLA
rd53	DECAF	5	3	27	132	PLA
rd53m	MISII	5	3	27	140	PLA
rd53bm	BOLD/MISII	5	3	49	252	PLA
rd84b	BOLD	8	4	122	510	PLA
rd84bm	BOLD/MISII	8	4	83	417	PLA
rd84d	DECAF	8	4	114	561	PLA
rd84m	MISII	8	4	117	594	PLA
rot	MISII	135	107	377	1896	BLIF
z4ml	MISII	7	4	30	162	BLIF

Table B.1: Details on Example Set

Example Name	Original Path Delay		Final Path Delay	
	Exp 1	Exp 2	Exp 1	Exp 2
5xp1	6.0	8.0	4.0	6.2
9symml	9.2	11.2	8.8	11.2
9symml_b	19.7	22.8	18.2	22.3
C880	23.0	28.2	17.5	22.2
apex6	14.5	17.6	9.5	10.7
apex7	12.0	12.8	10.2	10.9
apex7_b	11.5	12.6	9.3	9.4
b9	7.8	7.7	6.3	7.0
b9_b	5.4	6.0	4.2	4.1
duke2b	13.7	15.7	12.2	13.8
duke2bm	12.7	13.6	11.5	12.9
duke2d	14.0	14.5	9.4	11.0
duke2m	14.3	16.8	11.7	14.3
f51m	25.2	25.2	13.9	21.4
f51m_b	7.9	9.6	5.1	6.4
misex1d	6.1	5.9	4.2	5.1
misex1m	7.2	8.3	4.1	5.4
misex2b	6.8	7.3	5.9	5.9
misex2bm	6.0	6.9	5.7	5.8
misex2d	5.9	6.7	5.1	6.0
misex2m	7.7	9.2	6.7	6.4
rd53	6.2	6.5	3.1	3.3
rd53m	5.2	6.0	4.4	5.2
rd53bm	6.8	7.7	4.2	4.8
rd84b	13.7	16.9	10.6	13.7
rd84bm	13.2	16.5	6.8	13.6
rd84d	11.3	11.9	6.3	9.0
rd84m	10.3	12.3	8.2	12.3
rot	18.4	25.1	16.4	23.0
z4ml	6.8	8.1	5.1	5.5

Table B.2: Raw Numbers for Delay from Experiment One and Two

Example Name	Transistor Count		
	Original	Exp 1	Exp 2
5xp1	360	438	492
9symml	518	764	714
9symml_b	556	554	1706
C880	1136	1666	1666
apex6	2130	2334	2560
apex7	658	682	700
apex7_b	734	938	1074
b9	388	374	374
b9_b	364	356	350
duke2b	1498	1938	2120
duke2bm	1064	1196	1398
duke2d	1198	1480	1500
duke2m	1138	1382	1716
f51m	372	462	462
f51m_b	278	372	408
misex1d	180	278	278
misex1m	178	368	340
misex2b	416	462	416
misex2bm	300	344	468
misex2d	294	448	432
misex2m	282	396	522
rd53	132	130	132
rd53m	140	160	160
rd53bm	252	168	168
rd84b	510	664	698
rd84bm	416	426	450
rd84d	560	500	542
rd84m	594	612	834
rot	1896	1976	2252
z4ml	162	212	212

Table B.3: Raw Numbers for Area from Experiment One and Two

Example Name	AT^2			
	Original Sized	**Exp 1**	**Original Unsized**	**Exp 2**
5xp1	1.3×10^4	7.1×10^3	2.3×10^4	1.9×10^4
9symml	4.4×10^4	5.9×10^4	6.5×10^4	8.9×10^4
9symml_b	2.2×10^5	1.8×10^5	2.9×10^5	8.5×10^5
C880	6.0×10^5	5.1×10^5	9.0×10^5	8.2×10^5
apex6	4.5×10^5	2.1×10^5	6.6×10^5	2.9×10^5
apex7	9.5×10^4	7.1×10^4	1.1×10^5	8.3×10^4
apex7_b	9.6×10^4	8.1×10^4	1.2×10^5	9.6×10^4
b9	2.3×10^4	1.5×10^4	2.3×10^4	1.8×10^4
b9_b	1.1×10^4	6.3×10^3	1.3×10^4	5.8×10^3
duke2b	2.8×10^5	2.9×10^5	3.7×10^5	4.0×10^5
duke2bm	1.7×10^5	1.6×10^5	2.0×10^5	2.3×10^5
duke2d	2.4×10^5	1.3×10^5	2.5×10^5	1.8×10^5
duke2m	2.3×10^5	1.9×10^5	3.2×10^5	3.5×10^5
f51m	2.4×10^5	8.9×10^4	2.4×10^5	2.1×10^5
f51m_b	1.7×10^4	9.6×10^3	2.6×10^4	1.7×10^4
misex1d	6.6×10^3	4.8×10^3	6.3×10^3	7.2×10^3
misex1m	9.3×10^3	6.1×10^3	1.2×10^4	9.8×10^3
misex2b	1.9×10^4	1.6×10^4	2.2×10^4	1.4×10^4
misex2bm	1.1×10^4	1.1×10^4	1.4×10^4	1.6×10^4
misex2d	1.0×10^4	1.2×10^4	1.3×10^4	1.6×10^4
misex2m	1.7×10^4	1.8×10^4	2.4×10^4	2.2×10^4
rd53	5.1×10^3	1.3×10^3	5.5×10^3	1.5×10^3
rd53m	3.8×10^3	3.1×10^3	5.0×10^3	4.2×10^3
rd53bm	1.2×10^4	3.0×10^3	1.5×10^4	3.8×10^3
rd84b	9.5×10^4	7.4×10^4	1.5×10^5	1.3×10^5
rd84bm	7.3×10^4	1.9×10^4	1.1×10^5	8.3×10^4
rd84d	7.1×10^4	2.0×10^4	7.9×10^4	4.4×10^4
rd84m	6.2×10^4	4.1×10^4	9.0×10^4	1.3×10^5
rot	6.4×10^5	5.3×10^5	1.2×10^6	1.2×10^6
z4ml	7.5×10^3	5.5×10^3	1.1×10^4	6.4×10^3

Table B.4: Raw Numbers for AT^2 from Experiment One and Two

Appendix C
Invariance Proofs

In this appendix, proofs are presented that show the properties of primality and irredundancy, defined in chapter 3, are preserved by the transformations, except for path resynthesis. This results has important implications with regard to the testability of circuits, namely a combinational circuit is 100 % testable for single stuck-at faults if and only if it is prime and irredundant[BBH+88].

Theorem C.1 *The buffering transformation preserves the primality and irredundancy of circuits.*

Proof: Assume a circuit, C is prime and irredundant. Let F_i be the Boolean function associated with a gate in the circuit, and let y_i be a variable representing the output of function. By applying the buffering transformation, two gates are added to the circuit, each of which has a Boolean function associated with it: $F_1 = \overline{y_i}$ and $F_2 = \overline{y_1}$. Each of these new Boolean functions has only one literal and only one term, so there do not exist any literals or terms that can be removed without changing either of the Boolean functions. Therefore, by definition, they are prime and irredundant. □

Theorem C.2 *The critical signal isolation transformation preserves primality and irredundancy of circuits.*

Proof: The proof is similar to the previous theorem. Assume a circuit, C is prime and irredundant. Let F_i be the Boolean function associated with a gate in the circuit, and let y_i be a variable representing the output of function. By applying the critical signal isolation transformation, two inverter gates are added to the circuit, each of which has a Boolean function associated with it: $F_1 = \overline{y_i}$ and $F_2 = \overline{y_1}$. Each of these new Boolean functions have only one literal and only one term, so there do not exist any literals or terms that can be removed without changing either of the Boolean functions. Therefore, by definition, they are prime and irredundant. □

Theorem C.3 *The gate collapsing transformation preserves primality and irredundancy of circuits.*

Proof: From the point of view of logic synthesis, the gate collapsing transformation is algebraic resubstitution. Hachtel et al.[HJKM89] show that algebraic substitution preserves primality and irredundancy. When gate collapsing occurs across tree boundaries the first gate of the pair must be duplicated to preserve the intermediate node. While this operation converts any untestable single fault on the inputs to the first gate into a untestable multi-fault, it does not introduce a redundancy where none existed. Thus gate collapsing transformation preserves testability. □

Theorem C.4 *The gate decomposition transformation preserves primality and irredundancy of circuits.*

Proof: From the point of view of logic synthesis, gate decomposition is an algebraic decomposition operation. Hachtel et al.[HJKM89] show that algebraic decomposition preserves primality and irredundancy.

Theorem C.5 *The gate simplification transformation preserves primality and irredundancy of circuits.*

Proof: By definition, gate simplification tries to remove any non-prime literals or redundant terms from the gate. So if the circuit is already prime and irredundant, then gate simplification will not modify the circuit, hence preserving 100% testability. □

Theorem C.6 *The gate factoring transformation preserves primality and irredundancy of circuits.*

Proof: Gate factoring does not add or remove any literals or terms to the underlying two-level Boolean equation, therefore if the gate is part of a prime and irredundant circuit, then the circuit remains prime and irredundant. □

Theorem C.7 *The fan-in ordering transformation preserves primality and irredundancy of circuits.*

Proof: Fan-in ordering can be thought of multiple applications of the gate collapsing and gate decomposition transformations. Since both transformations have been shown to preserve primality and irredundancy, the fan-in ordering transformation also preserves primality and irredundancy. When this transformation is applied across tree boundaries and gates in the fan-in tree must be duplicated to preserve intermediate nodes, the circuit remains testable because this is the same operation as described in gate collapsing across tree boundaries. □

Theorem C.8 *The logic dual replacement transformation preserves primality and irredundancy of circuits.*

Proof: Assume the following Boolean equations are part of a Boolean network: $F_1 = \overline{a}$, $F_2 = \overline{b}$, $F_3 = \overline{y_1 \cdot y_2}$, and $F_4 = \overline{y_3}$. Since algebraic resubstitution preserves testability, it can be applied to the above equations to get $F_{new} = \overline{\overline{\overline{a} \cdot \overline{\overline{b}}}}$, which is just the same as $F_{new} = \overline{a + b}$ using deMorgan's Laws. So clearly, the testability remains the same. When applying this transformation across tree boundaries, a duplicate inverter may be required, similar the the duplicate gate in gate collapsing. Since this operation preserves primality and irredundancy, then the global application of this transformation preserves primality and irredundancy as well.

Theorem C.9 *The path resynthesis transformation does not preserves primality and irredundancy of circuits.*

Proof: Since current multi-level logic synthesis tools do not always produce prime and irredundant circuits, this transformation will not maintain primality and irredundancy if the logic synthesis tool used in resynthesis does not produce a prime and irredundant replacement path.

Appendix D
Implementation Details

D.1 Machine Environment For Experiment

All experiments were run on a Sun 4/260 with 64 megabytes of memory and 250 megabytes of swap space. The operating system was version 4-3.2 of the Sun Operating System, which is based on Unix 4.2 BSD. The software of POLO was written in C++[Str86] using GNU C++[Tie89] version 1.36.0 from the Free Software Foundation.

D.2 Software System Structure

The software used to run the experiments consists of three parts: POLO, AESOP, and MISII. POLO controls the operation as well as applying the transformation. POLO also calls AESOP to do transistor sizing, and calls MISII for certain logic synthesis operations. The author acknowledges the people who developed both of these tools since it saved countless hours of work that would have been required to write code to implement the same function.

D.3 Running POLO

Running POLO requires four different files. First, the circuit must be described in a gate-level netlist. Also, a file containing the input arrival times and output required times for the primary inputs and outputs is needed. The third file is an AESOP control file to allow running AESOP on the circuit. Finally, a technology file is needed to describe the set of gates used in the circuit and to describe the intrinsic technology parameters such as unit capacitance and resistance for the implementation technology.

There are several run-time options that are available when running POLO.

–F file name (required)

–T technology file name (required)

–a use all the transformations to speed up the circuit

–b apply the buffering and critical signal isolation transformations

–c apply the gate collapsing transformation

–d apply the gate decomposition transformation

–e apply the gate factoring transformation

–f apply the fan-in ordering transformation

–g apply the gate simplification transformation

–l apply the logic dual replacement transformation

–p apply the path resynthesis transformation

–r use path grouping (default is not to use path grouping)

–s use transistor sizing (default is not to use transistor sizing)

–P print out critical paths after each transformation

–D debugging flag

If the –a option is used, then it is an error if any flag is used that specifies a particular individual transformation. The separate option flags for each transformation allow for experimentation with subsets of the transformations. The option flags for experiment one (and for normal use) are: –F, –T, –a, –r, and –s.

D.4 Software Implementation Details

This section describes the software in POLO. The software, written in C++, is class-based, with each data structure defined as a class. The classes are grouped together into three hierarchies: a circuit class hierarchy, a technology class hierarchy, and a path class hierarchy. The circuit class hierarchy, shown in Figure D.1, holds information about the circuit including the gates, electrical nodes, and their connectivity. The transformations are implemented as functions defined as part of the class Circuit.

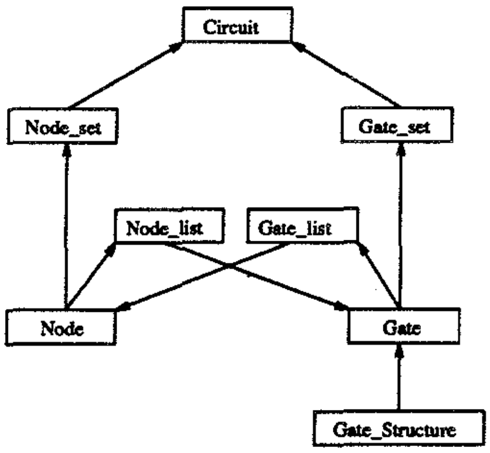


Figure D.1: Circuit Class Hierarchy

This decision, made early in the design, resulted in the class definition for Circuit being large. A better way to do it would have been to provide a few more basic operations in the definition of Circuit, and then define non-class operations for each of the transformations. Time limits did not allow this change to be made to the current implementation. In the future this change will be made.

The technology class hierarchy holds information about the implementation technology. This hierarchy, shown in Figure D.2, contains process technology information, such unit resistances and capacitances for different layers, as well as the types of gates available and their internal structure, down to the transistor level.

The final class hierarchy, the path class hierarchy, contains the critical paths for a circuit. Each path contains a list of gates on the path as well as the input, output and delay information. The hierarchy is shown in Figure D.3.

The implementation of the class library was aided by the use of class generators supplied in the GNU C++ Library[Lea89]. These generators quickly produced the sets, lists, priority queues, and hash tables that were used throughout the system, especially in the class hierarchies described above.

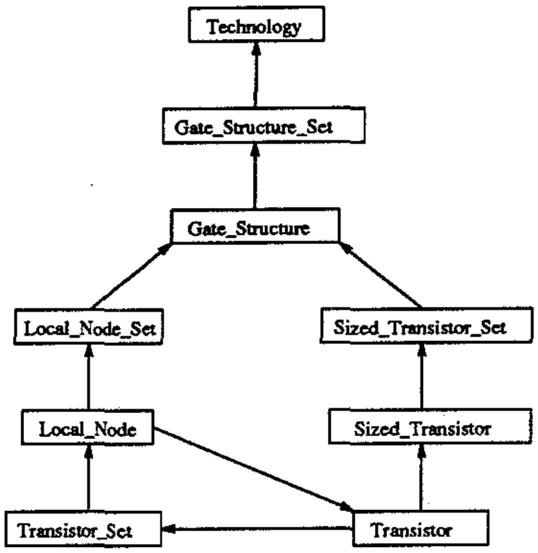


Figure D.2: Technology Class Hierarchy

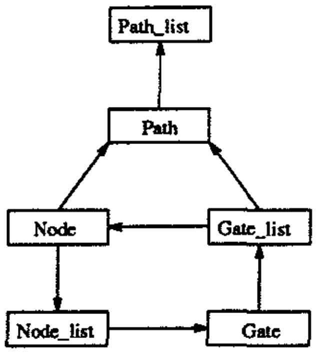


Figure D.3: Path Class Hierarchy