# Randomized Algorithms in Path Sensitization for Circuit Optimization and Delay Fault Tolerance

by

David M. Wessels

B.Sc., University of Victoria, 1989
M.Sc., University of Victoria, 1990

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy
in the Department of Computer Science

We accept this thesis as conforming
to the required standard

Dr. J. Muzio, Supervisor (Department of Computer Science)

Dr. M. Fellows, Departmental Member (Department of Computer Science)

Dr. V. Bhargava, Outside Member (Department of Electrical Engineering)

Dr. C. Miers, Outside Member (Department of Mathematics)

Dr. E. Cerny, External Examiner (Univ. de Montréal, Montréal, P. Q.)

Supervisor: Dr. Jon C. Muzio

## Abstract

Identification of the longest sensitizable path in a circuit is a key part of many modern timing analyzers for digital designs. For large circuits, however, identification of this path is often hampered by the presence of long false paths. It has been shown that identifying the longest true path in a circuit is an NP-complete problem, and many previous identification algorithms require an unacceptable amount of computation time for large designs. A number of heuristic techniques have been developed to reduce the computation time required in the identification process, however the running time can still be prohibitive for worst case circuits, and in some techniques only an upper bound on the length of the longest true path is obtained.

In this dissertation, randomized algorithms are used to search for the longest sensitizable path in a circuit. The goal in using these algorithms is to rapidly identify upper and lower bounds on the maximum true delay of a circuit.

Two new randomized techniques are developed and analyzed in detail. In the first, the problem is treated as one of searching a topological space: delays are mapped to vectors within the circuit input space, and by selecting random starting points and examining adjacent input vectors we search for maximal delays over the

entire input space. The second technique uses a set of CNF formulae to model an analysis circuit based on the original circuit. The analysis circuit includes circuit delay information. A randomized algorithm is used to seek satisfying assignments o the CNF formulae and in so doing provide bounds on the maximum delay possible in the original circuit.

The path identification algorithms developed are used to accurately estimate optimal circuit operating speeds, and also as the basis for a classification scheme for component delay-sensitivity. This classification scheme is applied in a timing optimization program, and is recommended as an aid to the efficient development of test sets for delay faults.

With respect to modeling circuit delay behaviour, an evaluation is conducted of some commonly used delay models. It is shown that the less sophisticated models have significant analytical shortcomings, and should not be used in analyzing path sensitization behaviour. Recommendations are made as to the minimum requirements for such models.

Examiners:

Dr. J. Muzio, Supervisor (Department of Computer Science)

Dr. M. Fellows, Departmental Member (Department of Computer Science)

Dr. V. Bhargava, Outside Member (Department of Electrical Engineering)

Dr. C. Miers, Outside Member (Department of Mathematics)

Dr. E. Cerny, External Examiner (Univ. de Montreal, Montréal, P. Q.)

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

One of the key problems in the area of circuit optimization and acceleration is the efficient determination of maximum circuit delay. Without knowing the maximum time a circuit requires to compute its function we are unable to determine the maximum *safe* operating speed of the circuit.

Most techniques to estimate the maximum delay of a circuit are based on the lengths of paths within a circuit. In such techniques a circuit is modeled as a set of components (eg. gates) connected by lines, with each line/component contributing a delay to signals propagated through it. The length of a path is the total delay encountered by a signal as it propagates from the start of the path (eg. at a circuit input) to the end of the path (eg. at a circuit output).

It is widely accepted that the simplest path-based estimation technique - the longest topological path in a circuit - is almost always a needlessly pessimistic estimator of circuit delay. This is due to the fact that in many cases there is no combination of input values which causes a signal to propagate along that path. The topologically-longest path in such a case is termed a "false", or non-

sensitizable, path[1]. While the existence of false paths has long been a concern, the growth of high-level synthesis systems is increasing the ease with which these paths are introduced to a design. To provide an accurate estimate of maximum circuit delay, therefore, a variety of techniques have been developed to determine the length of the longest "true" path in a circuit. Such techniques are the subject of this dissertation.

Because of the difficulty of the false path problem[2], most existing algorithms either sacrifice accuracy for speed of computation or are unable to analyze large circuit designs in an acceptable time frame. Complex circuit designs may contain hundreds of thousands of gates and millions of long false paths. Most existing techniques adopt heuristics for reducing the required computation time, but still require exponential run time in the worst case.

The approach adopted in this dissertation is to apply randomized algorithms to develop upper and lower bounds on the length of the longest sensitizable path in a circuit, and to establish these bounds in a "reasonable" time frame. As more computation time is allocated to the algorithms, better upper and lower bounds are obtained. A tradeoff can thus be established between the requirements of accuracy and allowable computation time.

Two new techniques are developed and analyzed in detail: a topological search algorithm, which is discussed in chapter 5, and a satisfiability algorithm based on a conjunctive normal form (CNF) representation of an analysis circuit, discussed in detail in chapter 6. Several other techniques, also based on randomization, are proposed in chapter 4, but their development is not pursued in this dissertation.

[1]Formal definitions are provided in chapter 2.

[2]Determining if a path is sensitizable is shown to be NP-complete in [38].

The application of the two new algorithms to a number of design and optimization problems is discussed in chapter 7. The applications discussed include:

- determination of optimal clock speeds (for purely combinational circuits) by using the algorithms to estimate the length of long sensitizable paths and taking the difference of this and the length of the topologically-shortest path in the circuit,
- the development of delay sensitivity criteria, used to classify circuit components by the sensitivity of the circuit to alterations in the delay associated with that component,
- circuit timing optimization through resizing key components - achieved through the use of both the new randomized algorithms and the sensitivity criteria to target key components in a circuit, and
- delay fault test set development - aided through the use of the randomized algorithms and sensitivity criteria to identify the types of delay faults to which the circuit is most sensitive.

Throughout this dissertation, discussion is limited to gate level models of combinational logic circuits. Familiarity with basic Boolean logic and circuit behaviour is assumed[3]. An important issue which must be addressed is the choice of gate delay model used to analyze expected circuit behaviour. In chapter 3, a number of commonly used delay models are analyzed to show the impact of the choice of model on predicted circuit behaviour - and hence on circuit analysis results. A particular concern is the current widespread use of models, such as the unit delay model, which fail to distinguish between gate rise and fall delays. These models

[3]See [50] or any introductory text on digital design.

are shown to be unreliable for accurate analysis of circuit behaviour, particularly with respect to path sensitization.

Chapter 2 contains an introduction to the problem of determining maximum circuit delays and how this is related to path sensitization and the false path problem. Several examples are included, along with the definitions and notation used throughout this dissertation. The chapter also contains an overview of the difficulty of the problem, previous work in the area, and an introduction to the randomization techniques applied in this dissertation. The delay model analysis is provided in chapter 3. Chapters 4 through 6 contain the descriptions and analysis of the new randomized algorithms, and chapter 7 shows the application of these algorithms in the areas of timing analysis, optimization, and delay fault testing. The results of chapters 3 through 7 are summarized in chapter 8, along with a set of notes on planned and suggested future work.

# Chapter 2

# Circuit delays and path sensitization

This chapter contains an introduction to the problem of determining maximum circuit delays, and how this is related to the path sensitization problem. Also contained here is a summary of existing work in the area, an overview of the techniques used in this dissertation, and the definitions and notation used throughout the balance of the dissertation.

First, an explanation of the problem is provided along with a set of examples, in sections 2.1 and 2.2 respectively. Terminology is used informally in sections 2.1 and 2.2 while intuitive explanations of the problems are provided. The terminology and problems are then formalized in later sections. Definitions and notation are provided in sections 2.3 and 2.4, while in sections 2.5 through 2.8 existing works in the area are introduced and discussed.

This dissertation concentrates on timing analysis issues for combinational logic. One of the most fundamental issues in circuit timing is the amount of time required for a circuit's outputs to stabilize after a vector has been applied to the circuit

inputs. Knowledge of the stabilization time is a key factor in determining when to sample the circuit outputs. If the outputs are sampled too early, then the correct logic values may not have stabilized at the circuit outputs, yielding incorrect results. If the input values are allowed to change, or destabilize, after a time then late sampling of the output values implies the circuit output values may also have changed. In either case, incorrect logic values can be sampled.

In section 2.1 we discuss the use of the longest sensitizable path in a circuit as an estimator for circuit clock speed - the maximum rate at which one can safely alter circuit inputs and sample circuit outputs yet still guarantee correct circuit behaviour. The sensitizable path estimator is further refined in chapter 7, where both the circuit's longest sensitizable path *and* shortest destabilizing path are considered.

## 2.1 The longest sensitizable path problem

In a combinational circuit composed of simple logic gates, the stabilization time of the circuit is a function of the delay a signal encounters when propagated along paths through the circuit. With respect to individual gates in the circuit, each time the logic values on a gate's inputs are altered there is a delay (the propagation delay) before the impact of those alterations is observable on the gate's output. The new output signal from the gate is then transmitted to subsequent gates in the circuit, where further propagation delays may be encountered. A simple estimate for the stabilization time of a circuit can be obtained by considering the longest path in the circuit from any primary input to any primary output. Length in this context is measured by the sum of the propagation delays encountered by the signal as it passes from a primary input to a primary output through a series of intermediate gates.

The longest topological path in a circuit is easily obtainable in time $O(|c|+|s|)$ for $c$ gates and $s$ line segments [29]. This path's length provides a rough estimate of stabilization time, but the estimate it provides is frequently pessimistic, and more accurate results can be obtained. Indeed, to meet the timing requirements of modern designs, more accurate results are a necessity [3]. One way of obtaining more accurate results is by measuring the length of long true (sensitizable) paths, rather than false (non-sensitizable) paths. The identification of true and false paths is the subject of the following section.

### 2.1.1 Introduction: true and false paths

Even if long topological paths are present in a circuit, it is possible that the circuit outputs *always* stabilize in an amount of time significantly shorter than that represented by the length of the longest topological path. This is caused by logic values propagating along shorter paths to dictate or 'control' the circuit output values regardless of the logic values present on the topologically longest path.

If the signals propagated along a particular path from a primary input to a primary output dictate the stabilization time of the primary output, then that path is said to be a true, or sensitizable, path. This path is said to be sensitized by the input vector which generated the signals. A path which can never be sensitized by any input vector is said to be a false, or non-sensitizable, path. Specific examples of true and false paths are presented in the following section.

## 2.2 Examples of the problem

This section contains intuitive examples and informal descriptions of the problems faced in path sensitization and timing analysis. Formal definitions and notation are introduced in sections 2.3 and 2.4, following which the problems are considered in greater detail. For the sake of simplicity, these examples use only logic circuits composed of simple logic gates, and it is assumed each gate takes one unit of time to compute its logic function on any inputs. The circuit is composed of AND, OR, NOT, and FANOUT gates, the symbols for which are shown in figure 2.1.


Figure 2.1: Symbols for logic gates

Here we consider a sensitization problem in which the initial internal logic values are regarded as undefined. The use of these "floating" values allows us to ignore circuit history (the internal logic values present due to previously applied input vectors) when determining circuit delays without underestimating the potential delay. For example, the output of an AND gate settles at logical *false* one time unit after the first of the gate's inputs settles at *false*, regardless of the presence of *true* or *undefined* logic values on the other gate inputs. The output of the AND gate settles at logical *true* one time unit after the last of the gate's inputs settles at *true*, i.e. when there are no inputs left carrying either *false* or *undefined* values.

The problem of determining the length of the longest sensitizable path in the

circuit (measuring length by delay) can be re-stated as follows: if a set of input values is applied to the circuit at time 0, how long must you wait to be certain the final output from the circuit is correct? Sensitization examples are considered in sections 2.2.1 and 2.2.2.

### 2.2.1 Example 1 - path delays and sensitization

A simple example circuit is shown below. To understand the problems of path sensitization and output stabilization, observe that different circuit input vectors force the circuit outputs to require different amounts of time to stabilize. For the purposes of determining the worst-case stabilization time of a circuit, it is necessary to find the slowest case. To illustrate the problem, consider the circuit of figure 2.2. In this circuit, input vector $< x_1, x_2, x_3, x_4 >=< 0, 1, 1, 1 >$ takes 1 gate delay to arrive at the correct answer whereas vector $< 1, 1, 1, 0 >$ takes 3 gate delays.

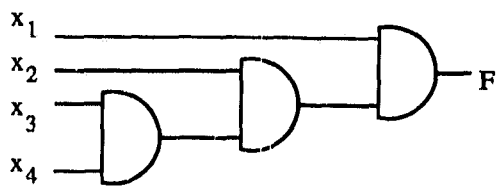


Figure 2.2: A circuit with maximum delay 3

In the first case, the input value $x_1$ forces the output $F$ to carry logic value 0 (*false*) after a single unit of delay, regardless of the values present on the other circuit inputs. $x_1$ is described as having a controlling value, or being a controlling input to the final AND gate in the chain.

In the second case, the value 0 must propagate from $x_4$ through all three AND

gates before $F$ stabilizes at logic value 0. Thus input vector $< 0, 1, 1, 1 >$ is said to sensitize the length 1 path from $x_1$ to $F$, and the input vector $< 1, 1, 1, 0 >$ is said to sensitize the length 3 path from $x_4$ to $F$.

### 2.2.2 False paths and easy upper bounds

Clearly under the unit delay model described above the maximum delay of a circuit can be no more than the number of gates on the "longest" path from any primary input to the circuit output (in the example above this would be from $x_4$ to $F$).

Well known polynomial-time algorithms are available to find this longest path [29], however the length of this path may be a pessimistic estimator. It is possible that a circuit will always compute the correct answer in time less than that required for a signal to follow the longest topological path in the circuit[1]. That is, a set of shorter paths exists which, for all input vectors, fully dictates the stabilization time of the circuit outputs.

In the example below, the longest path $(x_2, g_1, g_2, g_4, g_5)$ never carries a controlling signal to the circuit output, hence never determines the delay of the circuit. This can be shown as follows: when $x_3 = 0$ then $g_3$ is set to zero *before* the signal from $g_2$ arrives, hence $g_5 = 0$ and this determines the circuit output after a delay of 2. On the other hand, when $x_3 = 1$ then $g_2 = 1$ before the signal from $x_2$ to $g_1$ to $g_2$ arrives, and $g_2$ being 1 implies $g_3 = g_4 = 1$ and then $g_5 = 1$, for a delay of 3.

Here we say that the longest topological path is a *false* path, or a path that is never sensitized. Most large designs contain false paths. Some very common

[1]Indeed, it is shown in chapter 3 that under realistic delay models this is usually the case.

types of circuit, such as carry bypass adders or multipliers, contain thousands of false paths which are longer than any sensitizable path in the circuit [37].

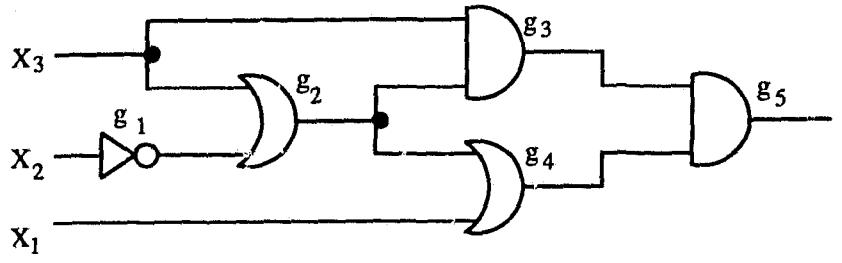


Figure 2.3: A circuit with a false path

If we wish to use the length of the longest true path in the circuit as an estimator for the the worst case output stabilization time, our problem can be restated as: given a circuit, find the length of the longest true (sensitizable) path it contains.

## 2.3 Definitions and notation

In this section we introduce the definitions and notation used throughout the rest of this dissertation.

### 2.3.1 Circuit description definitions

**gate:** A gate is a component implementing any one of the basic Boolean logic functions (AND, NAND, NOR, OR, NOT, XOR, XNOR). The gate has a set of input lines, or fanin, each carrying a Boolean logic value, and a set of output lines, or fanout, all carrying identical logic values (determined by the type of gate and logic values of the gate's inputs).

**line:** A line, or line segment, is a directed connection from the output of one gate or primary input to either a primary output or an input of another gate within the circuit.

**path:** A path is an alternating sequence of lines and gates, where each line connects an output of one gate to the input of another gate. Note that in combinational circuits these paths are acyclic.

**controlling value:** A controlling value for a gate is a logic value which, when applied to a gate input, determines the logic value of the gate output regardless of the logic values of the other gate inputs (eg. *false* for AND and NAND gates, *true* for OR and NOR gates).

### 2.3.2 Circuit input space definitions

**input vector:** An input vector is an assignment of logic values *true* or *false* to the primary inputs of a circuit.

**adjacent vectors:** A pair of input vectors are said to be adjacent if they differ in logic value at exactly one primary input. For example, $< 1,1,0,1,0 >$ and $< 1,1,1,1,0 >$ differ in exactly one bit position (the third) and are therefore adjacent.

**cube:** Given $m$ primary inputs, $x_1, \ldots, x_m$, a cube is a set of $2^k$ input vectors ($k \leq m$), in which fixed logic values are defined for $(m-k)$ of the primary inputs, and all combinations of Boolean values are permitted for the remaining $k$ primary inputs. For example, given $m = 3$, the cube $x_1\overline{x_3}$ represents the pair of input vectors $x_1x_2\overline{x_3}$ and $x_1\overline{x_2}\overline{x_3}$ (and here $k = 1$), whereas the cube $x_3$ represents the four input vectors $\overline{x_1}\overline{x_2}x_3$, $\overline{x_1}x_2x_3$, $x_1\overline{x_2}x_3$, $x_1x_2x_3$ (and $k = 2$).

### 2.3.3 Circuit delay behaviour definitions

**gate delay:** The gate delay, or **propagation delay** through a gate, is the time difference between the point at which a gate's input values change and the point at which the corresponding change takes place on the gate's output. For example, using the unit delay model discussed earlier, each gate in the circuit has a gate delay of 1.

**path delay:** For a path from primary input to primary output, the path delay is the accumulated propagation delays along all gates in the path as a result of a change in the logic values of the circuits primary inputs. For example, if a path goes through three gates, then in the unit delay model the path delay would be three.

**fall delay:** Fall delay refers to the gate delay which results from a change in gate input values which cause the gate's output logic value to change from logic value *true* (high) to *false* (low). The "fall" terminology reflects the high-to-low, or falling, transition[2].

**rise delay:** Similar to fall delay, rise delay refers to the gate delay on a change in gate outputs from low to high (from *false* to *true*).

**floating delay:** Floating delay refers to a delay model in which path delays are calculated on the assumption that the logic values on all lines in the circuit are unknown, or undefined, prior to the application of the input vector. Throughout this dissertation circuit behaviour is simulated using floating delays, as opposed to transition delays (see below).

---

[2]In chapter 3 it is shown that models which fail to distinguish between the rise and fall delays of gates are generally less accurate at modeling circuit behaviour.

**transition delay:** In a transition delay model (also called a two-vector delay), path delays for an input vector, $X_i$, are calculated on the assumption that the logic values on all lines in the circuit were established based on the application of a preceding input vector, $X_j$.

### 2.3.4 Path sensitization definitions

**dominating line:** A line is said to dominate a gate if the line is a fanin to the gate and either (a) the line carries a controlling value to the gate and is the earliest controlling input to reach the gate, or (b) the line carries a non-controlling value to the gate, all other fanin to the gate carry non-controlling values, and the line carries the last non-controlling value to reach the gate. For example, in the circuit of section 2.2.1 a logic value of *false* on line $x_1$ dominates the final gate in the chain on input $< 0, 1, 1, 1 >$.

**sensitizable path:** or **true** path: A path is sensitizable if, for at least one input vector, each line in the path is a dominating line for the succeeding gate.

**false path:** A path is a false path, or non-sensitizable, if there is no input vector under which the path is sensitizable.

**critical path:** A critical path, or **longest sensitizable path**, is a sensitizable path with associated path delay $T_c$ such that no other sensitizable path in the circuit has a greater associated path delay.

**topologically-longest path:** The topologically-longest path in a circuit is the path with maximum length, where length is calculated as the sum of the maximum delay associated with each gate in the path, disregarding sensitization criteria.

**topologically-shortest path:** The topologically-shortest path in a circuit is the path with minimum length, where length is calculated as the sum of the minimum delay associated with each gate in the path, disregarding sensitization criteria.

**sensitizing vector (cube):** For a specific sensitizable path, a sensitizing vector is a vector under which each line in the path is a dominating line for the succeeding gate. A sensitizing cube is a cube of input vectors, each of which is a sensitizing vector for the path. ($k$ inputs are "don't-cares" with respect to sensitizing the path.)

**critical vector (cube):** A critical vector (cube) is a vector which is a sensitizing vector (cube) for at least one critical path in the circuit.

### 2.3.5 Notation

The following notational conventions are used throughout this dissertation.

*true, false*: denote the Boolean logic values true and false, respectively.

$C$: denotes the circuit under analysis.

$m, n$: denotes the number of primary inputs and outputs, respectively, of circuit $C$.

$X =< x_1, \ldots, x_m >$: denotes an $m$-bit primary input vector to circuit $C$.

$F =< f_1, \ldots, f_n >$: denotes an $n$-bit primary output vector from circuit $C$.

$c, s$: denote the number of gates (components) and line segments in $C$.

$G = \{g_1, \ldots, g_c\}$: denotes the set of gates comprising circuit $C$.

$L = \{l_1, \ldots, l_s\}$: denotes the set of line segments comprising circuit $C$.

$T_s(X_i)$: denotes the stabilization time for circuit $C$, given $X_i$ as a primary input vector.

$T_{clock}$: denotes the clock speed at which circuit $C$ is being operated, i.e. the rate at which input vectors are being applied and output vectors are being sampled.

$T_{top}$: denotes the length (measured by delay) of the topologically-longest path in circuit $C$.

$T_{short}$: denotes the length (measured by delay) of the topologically-shortest path in circuit $C$.

$T_c$: denotes the length of the longest sensitizable path in $C$, i.e. the maximum $T_s(X_i)$ over all $X_i$.

$T_u, T_l$: denote the current known upper and lower bounds, respectively, on $T_c$ for circuit $C$, that is, $t_{short} \leq T_l \leq T_c \leq T_u \leq T_{top}$.

$\epsilon$: denotes the allowable fraction of error in an estimation. If the value being estimated is $T$, then the acceptable range for the estimate is $\frac{T}{1+\epsilon}, T(1+\epsilon)$.

$1 - \alpha$: denotes the desired level of probability that an estimate is correct to within the $\epsilon$ value specified above, $0 \leq \alpha \leq 1$.

## 2.4 Why is the problem of interest?

As pointed out in section 2.1, the longest sensitizable path problem has applications in a number of areas in circuit timing. Three key areas are: determining optimal clock speeds, circuit timing optimization, and delay fault testing. These

three areas are introduced in sections 2.4.1 through 2.4.3, and discussed in detail in chapter 7.

### 2.4.1 Circuit timing behaviour

The first area of application for the algorithms presented here is in the accurate analysis of circuit timing behaviour, specifically with respect to determining optimal clock speeds. As circuit designs become larger and circuit components become smaller and more densely packed, the timing constraints placed on the design become more and more restrictive. Accurate analysis techniques, once something of a luxury, are becoming a design necessity.

It has recently been shown [13] that an excellent estimator for optimal clock speed for combinational circuits is given by the difference between the maximum stabilization time of a circuit and the shortest topological path length of the circuit, i.e. $T_c - T_{short}$. In chapter 7 this is explained further, and used in conjunction with the new techniques presented in chapters 5 and 6 to provide the basis for an accurate timing analysis tool.

### 2.4.2 Circuit timing optimization

Once the timing characteristics of a circuit have been analyzed, this information can be used to improve the optimal clock speed of a circuit by limited re-design of the circuit. In chapter 7, using the circuit stabilization and destabilization criteria discussed above, a method is presented by which each gate in the circuit is classified as zero tolerant to delay increases, zero tolerant to delay decreases, both, or neither.

Having classified the circuit components, we show that optimization of the timing behaviour of the circuit - reducing the optimal clock speed - can be rapidly achieved by resizing a small number of gates in the circuit to increase or decrease their associated delay. The goal is to resize a small a number of gates while still achieving significant timing improvements. The criteria used to target gates for resizing is to choose gates which are intolerant of either delay increases or delay decreases, but not intolerant of both. The results of applying these criteria in a simple optimization loop are given in chapter 7.

### 2.4.3 Sensitivity to delay faults

The same sensitivity criteria used for timing optimization can also be used to decrease the size of test sets needed for delay fault testing, and to decrease the development time associated with creating such test sets. This process is also discussed in chapter 7.

## 2.5 Difficulty of the problem

Because of the practical design implications of the problem, many efforts have been made to analyze the false path problem and develop more efficient solutions. Recent deterministic solutions are summarized in section 2.6 of this chapter, while in this section existing complexity results for the problem are summarized.

McGeer [38] has shown that each of the following three problems is NP-complete:

- finding the longest sensitizable path in a circuit,

- calculating the length of the longest sensitizable path in a circuit, and
- determining if the topologically-longest path in the circuit is sensitizable.

Thus, given a circuit with a large number of gates and primary inputs[3], the critical path problem is prohibitively expensive computationally for most intuitive approaches.

A number of deterministic techniques have been developed in attempts to overcome this difficulty, and are discussed in section 2.6. Each of these techniques has associated drawbacks, however. These drawbacks are either in the form of worst-case exponential run times, or in sacrificing accuracy for speed of computation when analyzing large complex circuits. In chapters 5 and 6 two new algorithms are proposed and analyzed. Each of these algorithms is based on randomization techniques, which are used to rapidly provide upper and lower bounds on the length of the longest sensitizable path in a circuit.

## 2.6 Existing approaches

There are four basic styles of approach used in existing deterministic solutions to the critical path problem: timing verifiers, basic path enumeration, and dynamic analysis, and timed test generation. These styles are summarized below, as are the disadvantages associated with each. A thorough introduction to deterministic timing analysis is found in [38]. In section 2.7 our approaches to the problem, based on the use of randomization, are introduced. These approaches are considered in detail in chapters 4 through 6. The three basic styles of deterministic

[3]The benchmark circuits considered in this dissertation have from 3 to 256 inputs, and from 15 to 10,000 gates.

approach are:

**Timing verifiers:** In this approach the circuit is modeled through a set of differential equations, and the length of the longest path in the circuit determined by solving the set of equations. The primary disadvantage of these techniques is a failure to accurately consider sensitization criteria - i.e. the length obtained may be very accurate, but the path it represents may not be sensitizable.

**Path enumeration:** In these approaches, paths in the circuit are enumerated and checked for sensitivity. Enumeration is usually achieved through some variant of depth-first search, and sensitivity checking through a variant of the D-algorithm for stuck-at fault testing. Such methods are also impractical for large circuits with many long false paths, as each such path must be checked before a true sensitizable path is found[4]. While techniques have been developed to test sets of paths in parallel [34], the computational costs are still prohibitively high for large designs.

**Dynamic analysis:** Under this approach, the ordering of paths to be searched is improved using best-first techniques. This is well detailed in [38] [39]. While these methods are a significant improvement over earlier approaches, the algorithms are still worst-case exponential and still must search a large number of false paths for many large designs.

**Timed test generation:** these techniques, based on automatic test pattern generation (ATPG) algorithms such as PODEM [22] select a time value, $T$, and simultaneously propagate a set of signals through the circuit along all sensitizable paths of length at least $T$. These algorithms operate very quickly

[4]It has been shown that circuits (large multipliers and carry bypass adders are frequently cited examples) may contain tens of thousands of long false paths.

on irredundant circuits, and currently represent the "state-of-the-art" in false path algorithms. Unfortunately the algorithms do *not* perform well on circuits containing redundant line segments.

Other heuristic attempts to accelerate the process, based on a timed Boolean algebra, are given in [26]. These approaches complete in polynomial time, but the accuracy obtained in that time may be relatively poor.

Our approach is also to allow a trade-off between accuracy and speed, the longer the algorithms are permitted to run the better the results obtained. To achieve this, randomized - rather than deterministic - approaches to the critical path problem are introduced in section 2.7.

## 2.7 Randomized approaches

In chapter 4 six randomized approaches are proposed as possible solutions to the critical path problem. Two of the approaches are developed in detail, in chapters 5 and 6, while the detailed development of the other four approaches is left to future work.

The six randomized approaches can be divided into two categories: techniques which perform input space searches and simulation in floating delay mode, and techniques which are based on establishing delay bounds through examination of an analysis circuit based on viability criteria proposed by McGeer in [38].

### 2.7.1 Input-space searches

Two of the six techniques mentioned are based on searching the circuit input space for vectors which sensitize maximum length paths. One of these two techniques is simply random test pattern generation, widely used as a fault detection technique. The second of these techniques is termed topological search. This is a new technique which uses a directed search of the circuit input space from a series of randomly chosen starting points. The topological search is developed and analyzed in detail in chapter 5.

### 2.7.2 Analysis circuit approaches

The second category of randomized techniques consists of techniques which are based on an analysis circuit. The analysis circuit combines information about the logical state of the line segments in the original circuit with Boolean assertions about the stabilization times of those line segments.

Four different techniques are proposed in this category, each using the satisfiability of the analysis circuit under different input conditions to establish and improve upper and lower bounds on the maximum true delay of the original circuit.

The primary solution from this category is developed and analyzed in detail in chapter 6, and involves translating the analysis circuit into a Boolean expression in conjunctive normal form (CNF) with a high percentage of binary clauses. A randomized algorithm is applied to rapidly find satisfying assignments to the formula.

# Chapter 3

# Delay models

In this chapter several gate delay models are examined to determine their impact on the identification of sensitizable paths and maximum circuit delays in combinational logic circuits. Recommendations are provided on the "minimum acceptable" gate delay model for identifying critical paths in a circuit, and a minimum acceptable model for determining maximum circuit delays.

It is shown that some of the existing simple gate delay models are not sufficiently accurate for effective use in current delay analysis techniques. In particular, the use of delay models which fail to distinguish between rise and fall delays often results in incorrect modeling of both path delays and path sensitization behaviour. Such models, including the commonly-used "unit-delay" model, are shown to significantly misrepresent circuit delay behaviour, particularly with respect to critical paths and long false paths.

## 3.1 Introduction

It is shown in this chapter that overly simplistic choices of delay models at the gate level result in unacceptable distortions of circuit behaviour. Of particular concern is the effect that the choice of model has on the identification of critical paths in a circuit, and on the maximum delay in the circuit as modeled.

In any delay analysis of a circuit, the results obtained are dependent on the model used to calculate delays. The choice of model also impacts both the speed and the accuracy of the simulation. Unfortunately, there is little available literature comparing the effects of different delay models in terms of which paths in a circuit are identified as critical, and which paths are identified as false under the model. Chen and Du [10] recommend criteria for comparing sensitization algorithms, but do not include delay models in these criteria. Other authors [25] [27] state that we can use unit delay models "without loss of generality", but this is only true with respect to the correct operation of the algorithms within the limitations of a given model - *not* with respect to the correct modeling of overall circuit behaviour and *not* with respect to the practical efficiency of the algorithm.

Among the techniques used here to analyze the models is one in which pairs of adjacent input vectors, $v_i, v_j$, are examined, as are the delays associated with them, $d_i, d_j$. If $d_i > d_j$ under the most accurate model used then it is desirable that this relation also hold true in the simpler models. In fact, for simpler delay models this is often not the case. It is shown that the predicted nature of circuit behaviour is very different under different models. This goes well beyond the simple problem of under/overestimating delay by several percent - changing the model often has a fundamental impact on predicted circuit behaviour.

In section 3.5, a set of recommendations is provided which should be followed

when selecting delay models for use in the identification of sensitizable paths. Recommendations are also made concerning which delay models should be used to obtain estimates of maximum circuit delay. Summaries of experimental results taken over a set of standard benchmark circuits are given to support these recommendations.

## 3.2 The delay modeling problem

In this section the basic delay modeling problem is introduced. Simplistic gate delay models, such as the unit delay model described with the MCNC library [1], are still widely used in research programs studying path sensitization and path delay faults. Many published works use the unit delay model while at the same time applying complex analytic techniques to determine true circuit delays[1]. While the use of the unit delay model simplifies description and analysis of algorithms, it also has a dramatic distorting effect on the results of circuit delay analysis, and so casts some doubt on the validity of their results. There seems to be little purpose in using such sophisticated analytical techniques when the underlying model is flawed.

In discussing simple delay estimators, such as the topologically longest path in a circuit, Benkoski et al [3] point out that rough estimates are no longer sufficient to meet modern VLSI timing constraints. However, given too simple a delay model, the estimates for maximum circuit delay are generally inaccurate regardless of the estimation technique chosen, as the modeled behaviour does not correspond to actual circuit behaviour. This is true both of delay estimates and of the identification of critical paths in the circuit. Paths which *are* identified as

[1]These works include [9] [14] [15] [17] [26] [36] [37] [40] [47].

critical in the circuit under more realistic models are often *not* identified as critical under the simpler delay models. Conversely, paths which are not identified as critical, or even sensitizable, under the more realistic models are often identified as critical under simple models.

Given that extensive efforts are currently underway to produce efficient algorithms to pinpoint maximum circuit delay and/or exactly identify the set of critical paths (see most of [3] - [53]), the continued use of delay models which are so simplistic as to significantly distort these paths and delays is extremely counter-productive.

The six different delay models considered in this chapter consist of the two standard models provided in the MCNC cell libraries, plus a series of simplifications on the more complex of these two models. The delay models are described in section 3.3. In section 3.4 the different models are used to evaluate the critical paths and maximum circuit delays for a set of 30 standard benchmark circuits. The results for each of the five simpler models are compared to the results obtained using the most realistic of the six models. Final evaluations of the models, and recommendations for the "bare-minimum" models for different purposes, are provided in section 3.5.

## 3.3 Choice of delay models

The models discussed here are all based at the gate level, as we regard this level as having the greatest potential for abuse in terms of choosing overly simplistic delay models. Furthermore, as mentioned in chapter 2, we calculate sensitization based on a floating delay mode, as opposed to the static and transition modes. The static delay model has been proven unsafe [38] for determining longest sensitizable paths,

and the transition model (which considers application of pairs of vectors) is more complex to analyze. Devedas, Keutzer, and Malik state in [15] that the longest sensitizable path calculated under floating delay mode provides an upper bound on the longest sensitizable path calculated under transition models, therefore a safe result is still obtained.

There are three further categories into which gate delay models are typically divided: fixed delay, monotone speedup, and bounded delay. In the fixed delay mode, an exact delay value, $D$, is specified for each component in a circuit. In the monotone speedup model the delay for a component is assumed to lie anywhere in the range $0 \ldots D_{max}$, for some delay value $D$. In the bounded delay model upper and lower bounds are specified for the possible delay value, i.e. $D_l \ldots D_u$. Devedas, Keutzer, and Malik (again in [15]) state that under floating delay mode all three of these categories provide the same longest sensitizable path, given that $D = D_{max} = D_u$. In this dissertation the models used are therefore restricted to fixed delays.

Given that modeling techniques chosen operate under floating delay mode with fixed delays per component, it remains to decide how specific delay values are to be allocated to component types. These models differ in whether or not distinctions are made between the delays associated with:

- gates of different types,
- gates with different numbers of inputs,
- different inputs to a gate,
- rising and falling delays.

Six component delay allocation models are chosen. The most accurate is

termed the "detailed estimator", or DE model. The simplest model is termed the "unit estimator", or UE model. The other four models (referred to as the DD, ID, GD, and SE models) are successively greater simplifications of the DE model. All six are described below:

**UE, unit estimator:** this is the simpler of the two models supplied in the MCNC cell library, and is very commonly used in delay estimation and path sensitization research. This model assumes a unit delay per gate, regardless of gate type, size, number of inputs, etc., with an additional delay of 0.2 units per gate fanout (though even the fanout distinction is ignored by many authors).

**DE, detailed estimator:** the more accurate of the two MCNC library models[2], this model is the most accurate considered in this work and is used as the basis for all comparisons. Separate delay levels are set for each fanin to the gate, by gate type, size, number of inputs, fanout loads, plus different delay levels for rising and falling signals.

**DD, delay distinct:** this model is a simplification of the DE model, and retains the distinctions between different gate types and rising/falling delays, but a standard delay is set across all inputs of any gate. (The standard delay is chosen as the maximum delay across all inputs for the gate type in the DE model.)

**ID, input distinct:** also based on the DE model, this model uses distinct delays for different gate inputs but does not distinguish between rising and falling signals. (The greater of the rising/falling delays under the DE model is chosen for each input.)

[2]A MOSIS 2*u* standard cell library.

**GD, gate distinct:** this model is a simplification on both the DD and the ID model. In this model no distinction is made between different gate inputs, and no distinction is made between rising/falling delays. (Each input to the gate uses the maximum value given over all the gate's inputs in the ID model.)

**SE, simple estimator:** under this model distinctions are eliminated between gate types and sizes, as well as between rising/falling delays and different gate inputs. This model is similar to the UE model, the only difference being the unit and 0.2 values used for gates and fanouts in the UE model are replaced by maximum values selected from the DE model.

Note that in these definitions the models SE, GD, ID, and DD are tailored to over-estimate delays compared to our chosen "accurate" model (DE) whereas the UE model may either over-estimate or under-estimate the circuit delay.

The most accurate model studied in this dissertation is the DE model described above, as this is the most accurate delay model we have observed in use at the gate level. There are various other models used in the research community:

- Numerous modeling techniques exist at the switch and transistor levels [6]. [18]
- Given layout information, topological wire length can be used as a delay estimator [7]. [44]
- Fang and Jone [20] use a variant on the unit delay model which involves a unit delay per fanout and no delay for the gate itself.
- Lam, Brayton, and Sangiovanni-Vincentelli [30] model circuit delays using timed Boolean functions based on gate delays of individual components.

In addition to considering bounded delays for components and path sensitization calculated on transition delays, a more accurate model could conceivably be developed by extending the DE model to consider the delays associated with each gate input given known logic states on the other gate inputs. This is an interesting area for future work, but for this dissertation we restrict our study to the six gate delay models described above.

In the remainder of this chapter, a set of standard benchmark circuits is examined to show that only the DD model should be considered an acceptable simplification of the DE model when a circuit is being analyzed for path sensitization. It is shown that, on average over the benchmark circuits, using the delay models which do not distinguish between rise and fall delays results in delay estimates which are no more accurate than those obtained by simply considering the longest topological path length under the DE delay model. In recent years it has been accepted that topologically-longest paths do not provide a sufficiently accurate estimate of circuit delay [3] [14]. It follows, therefore, that the "exact" delay estimates obtained using complex analytical techniques but simple delay models are also insufficiently accurate.

## 3.4 Evaluation of delay models

In this section the five simpler models, DD, ID, GD, UE, SE, are compared with the DE model by using each model to analyze a set of benchmark circuits[3]. For each delay model, the analysis covers the following circuit characteristics:

[3]The 30 benchmark circuits considered are the 30 small benchmarks listed in appendix 1: 9symml, C17, alu2, alu4, b1, cc, cm138a, cm150a, cm151a, cm152a, cm162a, cm163a, cm42a, cm82a, cm85a, cmb, cu, decod, f51m, ldd, majority, mux, parity, pcle, pm1, sct, tcon, vda, x2, z4ml.

- the length of the longest sensitizable path in the circuit,
- the length of the longest topological path in the circuit
  (i.e. including paths which are not sensitizable),
- the set of vectors which sensitize at least one longest sensitizable path in the circuit,
- the relative delays associated with adjacent input vectors
  (i.e. for adjacent vectors $x, y$, is the delay associated with input vector $x$ greater than, equal to, or less than the delay associated with input vector $y$).

In sections 3.4.1 through 3.4.3 the analysis results are summarized, and the implications with respect to the delay models are discussed. The benchmark circuits used are summarized in appendix 1, and the raw data used to obtain the results presented in this chapter are supplied in appendix 2.

The analysis in each case is carried out by exhaustive simulation over the benchmark set, using fixed delays per component and floating delay mode. Similar experiments have been conducted using larger benchmarks and random sampling of the input space, and preliminary results appear to corroborate the data provided here for the exhaustive analysis of smaller benchmarks.

### 3.4.1 Analysis using *true/false* path lengths

In this section, for each delay model, the lengths of the longest sensitizable paths and the topologically longest paths in the circuit are considered. These two values are different only when there is a false path in the circuit which is longer than the longest true path in the circuit. These values are of interest in two ways:

**Firstly,** how accurate is the determined maximum circuit delay[4] under the model compared to the maximum delay as determined under the DE model.

**Secondly,** for the model under analysis, what is the ratio

$$\frac{\textit{length of the longest sensitizable path in the circuit}}{\textit{length of the longest topological path in the circuit}}$$

again as compared to the ratio obtained using the DE model.

Table 3.1: Circuit delays and path lengths

(a) maximum circuit delay under given model vs. maximum delay under DE model

| | DE-topological | UE | SE | GD | ID | DD |
|---|---|---|---|---|---|---|
| weighted | 1.072 | •1.031 | 3.032 | 1.166 | 1.086 | 1.076 |
| unweighted | 1.099 | 1.004 | 3.066 | 1.188 | 1.089 | 1.087 |

(b) length of longest sensitizable path vs. topologically longest path length

| | DE | UE | SE | GD | ID | DD |
|---|---|---|---|---|---|---|
| weighted | 0.935 | 0.992 | 0.992 | 0.987 | 0.989 | 0.937 |
| unweighted | 0.910 | 0.971 | 0.970 | 0.967 | 1.000 | 0.909 |

(c) fraction of benchmark circuits for which no topologically longest path is sensitizable

| DE | UE | SE | GD | ID | DD |
|---|---|---|---|---|---|
| 0.933 | 0.067 | 0.067 | 0.133 | 0.138 | 0.933 |

• However, the average "error" compared to DE is ± 15.4%.

For the benchmark circuits studied, table 3.1 summarizes these values for each delay model. To obtain the weighted values the ratio is calculated for each circuit separately and the ratios are averaged. To obtain the unweighted values, on the other hand, the sum of the path lengths is taken over all circuits before calculating the ratios. The raw data from which the weighted and unweighted values are obtained is in included in appendix 2, tables 2.1 and 2.2.

[4]Here assuming the maximum circuit delay is estimated using the length of the longest sensitizable path in the circuit.

To analyze these results, let us first examine the maximum circuit delay estimates obtained under each model. As is shown in the data of table 3.1(a), circuit delays are overestimated an average of 7.6% when modeled under the DD delay model (the 7.6% figure is for the weighted results, 8.7% is the average overestimate using the unweighted results). Even worse estimates are produced using any of the other simple models except UE. The UE model produces better average results, but this is due to the fact that the UE model sometimes underestimates the delay, and sometimes overestimates. The average magnitude of error for the UE model is in fact $\pm$ 15.4% (again for the weighted case). Also included in table 3.1(a) is the ratio of the length of the topologically-longest path in the circuit to the length of the longest sensitizable path in the circuit, both under the DE model. It can be seen that, on average, the topologically-longest path under the DE model is at least as good an estimator of maximum circuit delay as the maximum sensitizable path length taken under simpler models. Since the topologically-longest path has been widely discredited as a delay estimator, it follows that obtaining maximum sensitizable path lengths under the simpler delay models is also inadequate. Unfortunately, this is exactly what is done in many current research efforts ( [9] [14] [15] [17] [26] [36] [37] [40] [47] ).

The next factors considered in the analysis are the relative lengths of the longest sensitizable path in the circuit and the topologically longest path in the circuit. The data values in table 3.1(b) show that, under the two models which make a distinction between rising and falling delays (the DE and DD models), the longest sensitizable path tends to be significantly shorter (by 6.5 to 9.1%) than the topologically-longest path in the circuit. Under the other delay models, however, the length differences are considerably smaller (0.8 to 3.3%). The values in table 3.1(c) also show that under the simpler models it is far more likely that at least one of the topologically-longest paths is identified as sensitizable. These

results are particularly significant with respect to path enumeration and checking techniques. Such techniques search for longest sensitizable paths in a circuit by successively analyzing different paths in the circuit, starting with the longest. If the topologically longest paths in a circuit are sensitizable these algorithms should quickly find a maximum length sensitizable path. Thus these algorithms appear to operate much more quickly when used with a simple delay model than would be the case if a more realistic delay model was applied.

### 3.4.2 Analysis using the sets of sensitizing input vectors

In this section the delay models are further evaluated, here with respect to the sets of vectors which sensitize critical paths in the circuit. It is shown that the models which fail to distinguish between rising and falling delays are generally inadequate for the purpose of circuit delay analysis.

Table 3.2: Ratio of sizes of sensitizing sets for critical paths

| Ratio of number of critical path sensitizing vectors under given model to number obtained using DE model | | | | | |
|---|---|---|---|---|---|
| | UE | SE | GD | ID | DD |
| weighted | 4.948 | 6.810 | 3.453 | 2.375 | 1.463 |
| unweighted | 2.156 | 2.178 | 2.202 | 1.652 | 1.021 |

In table 3.2, the relative size of the sets of sensitizing input vectors for critical paths is compared under each delay model. Weighted results are obtained as follows:

1. For each circuit, take the ratio of the size of the sensitizing set of vectors under the model to the size of the DE sensitizing set.
2. The average of these ratios across all circuits gives the weighted average for the model.

The unweighted results are obtained as follows:

1. First take, for each model, the sum of the sizes of the sensitizing vector sets across all circuits.
2. The unweighted average for the model is then the ratio of the sum under the model versus the sum under the DE model.

As can be seen from the data in table 3.2[5], analysis under the simpler delay models rarely identifies the same set of sensitizing vectors for critical paths as are identified under the DE model. In general, under the simple models the number of vectors which sensitize critical paths is overestimated. This is largely due to the characterization of some paths as critical which are *not* characterized as critical under the more advanced delay models.

In the discussion below the distribution of these sensitizing vectors across the input space is considered, again as analyzed under each of the different delay models.

### Critical cubes

In this section, the analysis focuses on how the sensitizing vectors for critical paths are grouped. Specifically considered is the arrangement of sensitizing vectors as cubes in the input space. A subcube is a set of $2^k$ input vectors, in which specific values are set for $m-k$ of the input variables, but the remaining $k$ input variables may take any values. The relevance in path sensitization is that the $k$ variables are "don't-cares" with respect to sensitizing a particular path.

[5]The raw data can be found in appendix 2, table 2.3.

In analyzing the cubes of critical-path sensitizing vectors, the sensitizing vectors are grouped into the largest cubes possible, and each cube is identified by the lexicographically-least input vector in the cube.

Table 3.3: Ratio of sizes of sensitizing cubes for critical paths

(a) Ratio of number of critical sensitizing cubes under model to number obtained using DE model

| | UE | SE | GD | ID | DD |
|---|---|---|---|---|---|
| weighted | 1.567 | 1.667 | 1.533 | 1.000 | 1.533 |
| unweighted | 1.010 | 1.011 | 1.006 | 0.998 | 5.914 |

(b) Fraction of benchmark circuits in which long path sensitization behaviour matches DE model

| DE | UE | SE | GD | ID | DD |
|---|---|---|---|---|---|
| 1.000 | 0.033 | 0.033 | 0.033 | 0.103 | 0.733 |

For each of the five simpler models, the ratio of

$$\frac{number\ of\ critical\ cubes\ under\ the\ simpler\ model}{number\ of\ critical\ cubes\ under\ the\ DE\ model}$$

is recorded in table 3.3(a). It is evident from the data in table 3.3[6] that under all cases but one the number of sensitizing cubes is very similar. The one exception is under model DD, and the deviation here is substantially due to the modeled behaviour on a single benchmark circuit, the "parity" benchmark circuit[7]. The number of cubes is generally very low in all models, frequently in the range 1-8 for the benchmarks considered. This grouping of the sensitizing vectors into a few relatively large cubes is also of interest in the analysis of circuit delay behaviour carried out in chapter 5, and is noted here for future reference.

The data in table 3.3(b) shows the fraction of benchmark circuits for which the set of critical paths under the model is the same as the set of critical paths under

[6]The raw data can be found in appendix 2, table 2.4.

[7]See appendix 1 for a summary of the benchmark circuits.

the DE model, and for which the same sets of input vectors and cubes of vectors sensitize critical paths. Only the DD model has sensitization behaviour matching that of the DE model for the majority of the benchmark circuits. The simpler models are incorrect for more than 88% of the benchmark circuits considered.

### 3.4.3 Analysis using relative delays of adjacent vectors

In this final evaluation section, all pairs of adjacent input vectors (input vectors which differ in only a single bit) are considered. If the delay associated with vector $x$ is less than the delay associated with an adjacent vector, $y$, in the DE model, then ideally the same relationship holds in simpler models. If this is not the case then clearly the simpler model is providing an inaccurate description of circuit delay behaviour, regardless of the absolute magnitude of the delays associated with the input vectors.

Table 3.4: Errors in relative delays for adjacent input vectors

| (a) fraction of adjacencies giving bidirectional errors | | | | | | |
|---|---|---|---|---|---|---|
| | DE | UE | SE | GD | ID | DD |
| weighted | 0.000 | 0.013 | 0.011 | 0.014 | 0.009 | 0.019 |
| unweighted | 0.000 | 0.003 | 0.005 | 0.002 | 0.002 | 0.002 |
| (b) total fraction of adjacencies giving errors | | | | | | |
| | DE | UE | SE | GD | ID | DD |
| weighted | 0.000 | 0.103 | 0.100 | 0.096 | 0.046 | 0.052 |
| unweighted | 0.000 | 0.040 | 0.049 | 0.037 | 0.025 | 0.010 |

| (c) fraction of circuits containing no bidirectional errors | | | | | |
|---|---|---|---|---|---|
| DE | UE | SE | GD | ID | DD |
| 1.000 | 0.267 | 0.267 | 0.133 | 0.267 | 0.167 |

To obtain the results summarized in table 3.4[8], all pairs of adjacent input vectors, $x, y$, are examined under each of the six delay models. Results obtained

[8]The raw data can be found in appendix 2, tables 2.5 and 2.6.

under each of the five simpler models are then compared to those obtained under the DE model. The value $d(x)$ is defined to be the delay associated with input vector $x$, and $d(y)$ to be the delay associated with input vector $y$. The following error classifications are made:

1. A bidirectional error is said to occur if $d(x) < d(y)$ under the DE model but $d(x) > d(y)$ in the model under examination, or conversely if $d(x) > d(y)$ under the DE model but $d(x) < d(y)$ in the model under examination.

2. A unidirectional error is said to occur if $d(x) = d(y)$ under the DE model but $d(x) \neq d(y)$ in the model under examination, or vice versa.

3. The total number of errors is the sum of the unidirectional and bidirectional errors.

4. The frequency of errors is the total number of errors divided by the total number of pairs of adjacent input vectors, i.e. $m2^{m-1}$ for all possible pairs of vectors.

Table 3.4 contains a summary of the number of bidirectional errors and the total number of errors occurring under each model. As in tables 3.1 through 3.3, both weighted and unweighted summaries are given. The final row of the table, 3.4(c), indicates the fraction of circuits for which no bidirectional errors are found in comparison with the DE model.

Analyzing these models in terms of minimizing the frequency of errors shows that using the ID model provides results which are slightly superior to the other simple models when considering the weighted values, while using the DD model gives slightly superior results when considering the unweighted values. However, no simple model gives results totally free of errors for any of the benchmark

circuits. Furthermore, for almost three quarters of the benchmark circuits bidirectional errors exist under each of the simple models.

In section 3.5 the results of the model evaluations are summarized, and recommended delay models are specified according to the form of circuit delay analysis which is to be carried out.

## 3.5 Chapter summary

Overall, it is recommended that no simplification of the detailed delay estimation model be used in the analysis of circuit delays at the gate level. If a simplification *must* be used, then as an absolute minimum one should choose a model which makes a distinction between rising and falling delays. Each of the simpler delay models clearly has inadequacies when used in estimating maximum circuit delay, and sensitization of long paths in a circuit.

With respect to maximum circuit delays, taking the topologically-longest path under the detailed model usually provides a better estimate of true circuit delay than finding the longest sensitizable path as identified under simpler models. Thus even advanced sensitization algorithms provide unreliable results if a simplistic delay model is used. The only delay model which enables estimates which are, on average over the benchmark circuits, superior to the DE-modeled topological-maximum is the DD model (the model which distinguishes between rise and fall delays, but not between gate inputs).

With respect to the longest sensitizable path in a circuit, the use of the simpler delay models frequently results in the identification of long paths in the circuit as sensitizable when using more realistic delay models reveals these paths are *not*

sensitizable. As a result, algorithms which enumerate and check long paths in the search for a longest sensitizable path require less computation time when a simple delay model (such as the unit delay model) is used, but can produce results which do not reflect the circuit behaviour under more accurate models. Such algorithms may well experience significant performance degradation if applied using a more realistic delay model.

When analyzing a circuit for path sensitization, choosing a model which distinguishes between rising and falling delays appears to be more important than choosing a model which provides distinct delays for different gate inputs. In 22 of the 30 benchmark circuits examined the DD model *did* correctly sensitize the longest path in the circuit (as identified by the DE model), without sensitizing any longer paths in the circuit which are identified as false paths under the DE model. No other simple model exhibited correct sensitization behaviour for long paths in more than 4 of the 30 circuits examined.

When comparing the relative delays associated with adjacent vectors, in analysis over the benchmark set, *no* circuits were found in which *any* of the simpler delay models accurately represented all aspects of the circuit delay behaviour.

Thus our conclusion is that, if accurate analysis of the delay behaviour of a circuit is desired, the simplest delay model which should be used is one which distinguishes between rising and falling delays. Ideally one should not use any simplification on the delay estimation technique (described here as the detailed estimator) which is provided with a library of standard gates[9] in the MCNC benchmarks.

---

[9]Cell library 2.

# Chapter 4

# Randomized delay algorithms

In this chapter six new randomized techniques are outlined, each with the potential for use in estimating the maximum true delay of a circuit. The first two techniques are based on simulating a subset of a circuit's input vectors in an attempt to find at least one vector which sensitizes a longest true path in the circuit. The remaining four techniques analyze the circuit logic and attempt to derive the maximum true delay in the circuit. The two most promising of the six techniques are considered in detail in later chapters: the topological search algorithm is developed in detail in chapter 5, and the CNF translation algorithm in chapter 6. Development and analysis of the remaining four techniques is left as a topic for future work.

Of the two techniques covered in detail, the first is referred to as "topological search". This technique involves performing a directed search of the circuit input space from a series of random starting points, with the goal of quickly finding a vector which sensitizes a critical path in the circuit. The second technique is referred to as "CNF translation", and in this technique the circuit under analysis is first translated into a circuit which incorporates delay information as part of the logic function. This new analysis circuit is translated into a CNF formula,

and it is possible to determine increasingly accurate upper and lower bounds on the length of the critical path(s) in the original circuit by determining whether or not the formula is satisfiable under a number of different input conditions.

Of the remaining four techniques, one - random test pattern generation (RTPG) - is based on a search of the input space, while the other three make use of the analysis circuit described for the CNF translation algorithm. These latter three techniques are based on implication propagation, path tracing, and integer programming, respectively.

In the seven sections of this chapter the two input space searching techniques are introduced first, RTPG and topological search. The properties of the analysis circuit are then summarized (these properties are examined in detail in chapter 6) and each of the logic analysis algorithms is briefly examined (CNF translation, implication propagation, path tracing, and integer programming).

## 4.1 Random Test Pattern Generation (RTPG)

This section contains a description of a simple randomized algorithm to estimate the maximum delay possible in the circuit. The technique operates by simulating the behaviour of a number of input vectors, and outputs the correct maximum circuit delay if at least one of the vectors simulated sensitizes a maximum length true path in the circuit. If none of the selected vectors sensitize a maximum length path then the method underestimates the worst case delay of the circuit. Thus, while the method quickly determines the correct delay in many cases, when it errs the result can be particularly dangerous. (If maximum sensitizable path length is used as a clock speed estimator, then it is safer to overestimate the delay when an exact result cannot be obtained.)

The technique is termed Random Pattern Test Generation (or RTPG), and simply involves the generation of a series of random input vectors. The behaviour of each of a predetermined number of pseudo-random input vectors is simulated, in the hope that at least one of those vectors sensitizes a maximum length path.

In the worst case, if only a single sensitizing vector exists out of the $2^m$ possibilities, then the chance of failure after $t$ trials is $(1 - \frac{1}{2^m})^t$. In such a case, to guarantee a chance of failure less than some constant $\alpha$ requires an exponential number of tests.

This algorithm works well only if many input vectors sensitize longest paths. Because of the similarities between RTPG and the topological search algorithm discussed in detail in chapter 5, the RTPG technique is implemented and applied to a set of benchmark circuits. The results are presented in chapter 5, and demonstrate that the algorithm performs well on many of the benchmark circuits. However, such behaviour obviously cannot be guaranteed for all circuits. If high reliability is a requirement then exponential running times must be accepted for this algorithm, and the algorithm is thus unacceptable for circuits with unknown internal properties and a large number of primary inputs.

## 4.2 Topological search

The second technique uses a directed search of the circuit input space, seeking vectors which are local maxima with respect to their associated circuit delays. The basis of the algorithm is as follows:

```
Set the maximum known delay to 0
Repeat until the desired confidence level is reached:
    1) Select a random input vector
```

```
2) Repeat until no new delay vectors are discovered:
   a) determine the circuit delay given the current input vector
      (the maximum input-to-output delay, assuming indeterminate
       internal values initially)
   b) update the maximum known delay if necessary
   c) examine each adjacent input vector to determine if one gives
      a larger delay than in (a), if one is found then call (2)
      recursively on that new vector
```

This algorithm is developed and analyzed in detail in chapter 5, along with enhancements to the algorithm which concentrate on eliminating duplicate paths from the search tree, and/or optimizing the delay calculations.

## 4.3 The analysis circuit

An analysis circuit, denoted $C_A$, is used as a tool in each of the four techniques discussed in sections 4.4 through 4.7. This analysis circuit is based on the original circuit under consideration, $C$, and includes all the logic of the original circuit plus some additional logic used to calculate circuit delay information.

The analysis circuit is developed in detail in chapter 6, here its main properties are summarized so the reader can see the relationship between the analysis circuit and the randomized algorithms discussed in the next four sections of this chapter.

Given that the original circuit has $m$ primary inputs, the analysis circuit takes as input the same $m$ primary inputs, plus a Boolean vector $T$ which is an encoding of time. The analysis circuit has only a single output, $\chi$, and given inputs $X$ and $T$, the output of the analysis circuit is *true* iff some output of the original circuit takes time *at least* $T$ to stabilize given input vector $X$ at time 0.

The impact of having such an analysis circuit, is as follows: if the analysis

circuit is not satisfiable for some given $T$ value, then $T$ is an upper bound on the length of the critical path(s) of the original circuit. If the circuit *is* sensitizable for $T$, then $T$ is a lower bound on the length of the critical path(s). The algorithms discussed in the next four sections of this chapter rely on these properties of $C$ and $C_A$.

## 4.4 CNF translation

The CNF translation algorithm is considered in detail in chapter 6, but a brief sketch of the algorithm is provided here. This algorithm makes use of an analysis circuit, $C_A$, based on the original circuit. The primary feature of the analysis circuit is that for input $X$ and time $T$ the output of $C_A$ is *true* iff $C$ takes time at least $T$ to compute its function given $X$ as input.

In the CNF translation technique, the $C_A$ circuit is analyzed, and a series of improvements are sought for the known upper and lower bounds on the length of the critical path(s) in the original circuit. The bounds are established by translating $C_A$ into a Boolean expression in conjunctive normal form (CNF), then trying to determine time values for which the expression is provably satisfiable or provably unsatisfiable. When referring to CNF formulae, a formula is described as 2CNF if all clauses in the formula contain at most 2 literals, and as $k$CNF if all clauses in the formula contain at most $k$ literals.

The analysis of $C_A$ is carried out as follows:

- The gate description of $C_A$ is first translated into a CNF formula.
- Because the majority of the clauses in this formula contain only two literals, the formula is split into two parts: the binary clauses are modeled using an

implication graph, and the remaining clauses are retained as a CNF formula.

- A solution to the 2CNF formula represented by the implication graph is easily found, however finding such a solution which is also a solution to the more general formula is more difficult. A probabilistic algorithm is invoked to search the 2CNF solution space for a solution to the general $k$CNF formula.
- This search is invoked for a series of $T$ values, each new $T$ value improves either the known upper or lower bound on the length of the critical path(s) in $C$, and also allows simplification of the implication graph and the $k$CNF formula.
- The search can be terminated either when the upper and lower bounds converge, or when a fixed number of $T$ values have been searched.

Three other randomized techniques, which also utilize $C_A$, are proposed in the next three sections. Each of these techniques may have the potential to be developed into a solution to the false paths problem, but detailed development is left as problem for future work.

## 4.5 Implication propagation

The problem of determining whether or not paths are sensitizable in the original circuit, $C$, or the problem determining whether or not the analysis circuit, $C_A$, is satisfiable for certain time values, can each be solved by considering the implications of the logic values carried by sets of lines in a circuit.

The logic value carried by each line in a circuit has implications for the logic values carried by other lines in the circuit. For instance, a logic value of *true* on

the output of an AND gate implies that the value carried by every input line is a *true*. Such implications provide one way of determining the satisfiability of a circuit under set conditions.

Several existing solutions to the false path problem take advantage of logical implications within a circuit [3] [9] [34] [46]. However, such approaches can be extremely time consuming. McGeer [38] has shown that the implications of each individual input to a circuit can be propagated throughout the circuit in polynomial time ($O(n^4)$), but there is no known polynomial time algorithm to propagate the implications of *sets* of input values.

Much of the overhead in this process involves the propagation of information which is either incomplete or redundant. Existing techniques apply a variety of heuristics to improve the efficiency of propagation. This seems inherently amenable to some form of randomization. For example, if an input to an AND gate is *true*, and there are many other as yet unset input values, then the *true* is not likely to be a controlling value, so with some probability $p$ do not attempt to make further deductions until other inputs to the gate have been settled. The detailed development of such an algorithm is left to future work.

## 4.6 Path tracing

Another potential source for a randomized solution is in the form of path searching or tracing on the analysis circuit. In such an approach, one would repeatedly select $T$ values, and, given the restrictions of the $T$ value selected, follow paths through the circuit from the primary outputs to the primary inputs in search of a satisfying assignment of input variables. Finding an assignment reveals a new lower bound on $T_c$, and the provable absence of any such assignment reveals a new upper bound

on $T_c$.

The suggested use of randomization in such an algorithm is at the decision stage for each branch point in a path (i.e. fanin to a gate in $C_A$). A possible set of decision criteria could be based upon the number of unassigned variables present in the subtree, and the restrictions imposed by the output logic value required for the current logic gate. For instance, if a logic value of *true* is required on the output of a 3-input OR gate, then the probability of obtaining that value might be partitioned into a probability of $\frac{1}{7}$ for each of the input combinations $001, 010, 100, 011, 101, 110, 111$.

The development of the path tracing is not, however, pursued further in this dissertation.

## 4.7 Integer programming

The final potential approach discussed here involves the translation of the problem from the Boolean domain to the integer domain. The method presented here is *not* yet fully adapted for an integer programming approach, but does provide some insights. This approach is based on translation of the Boolean formula implemented by $C_A$ into an arithmetic expression, then seeking to maximize the $T$ value for which $C_A$ is satisfiable.

The basic principles of translating Boolean AND, OR, NOT expressions into arithmetic expressions are as follows: The integer 0 represents logical *false*, and 1 represents logical *true*. For the complement operation, $\overline{x}$ becomes $(1 - x)$, and the product operation remains the same $1 \cdot 1 = 1$. The sum operation is obtained using DeMorgan's law, $x$ OR $y$ is the same as the complement of ($\overline{x}$ AND $\overline{y}$).

Arithmetically, this evaluates to $1-(1-x)(1-y)$.

Given this translation scheme from Boolean logic into integer arithmetic, consider the Boolean expression for a logic gate, illustrated here with the expression for an OR gate. The derivation for this formula is presented in detail in chapter 6 (section 6.1.2).

$$NOR, OR\ gates: \chi^{g_i,t} = \sum_{i=1}^{k} \left[ \chi^{a_i,t-1} \cdot \prod_{j=1,j\neq i}^{k} \left( \chi^{a_j,t-1} + \overline{a_j} \right) \right]$$

Where $\sum$ is the OR operation, $\cdot$ is the AND operation, $g_i$ represents the output of a gate, $a_i$ represents the $i^{th}$ input to the $k$-input gate, and $\chi^{g,t}$ is *true* iff $g$ takes time at least $t$ to stabilize after an input vector has been applied to the circuit primary inputs.

Given the logic expression, apply the Boolean-to-integer translation, and obtain the arithmetic expression:

$$\chi^{g,t} = 1 - \prod_{i=1}^{k} \left[ 1 - \chi^{a_i,t-1} \prod_{j=1,j\neq i}^{k} \left( 1 - (1-a_j)(1-\chi^{a_j,t-1}) \right) \right]$$

Similar translations can be applied to the other basic types of logic gate.

Finally, to determine the maximum true delay of $C$, it is necessary to find a solution to the above formula which maximizes $T$ with the following constraints:

- $\chi^{F,T} = 1$
- each input variable $x_i$ may be set to 1 or 0 for $1 \leq i \leq m$
- $\chi^{x_i,T-r} = 1$ iff $T \leq r$

A linear solution to this problem should be much easier to obtain than the integer solution, and may provide a reasonable estimator either indirectly, through supplying an estimator for the input vector $X$, or directly for the time $T_c$. In using the linear solution to supply an input vector $X$, if $x_i < 0.5$ in the linear solution then assign the logic value *false* to the $i^{th}$ input in the current attempt to satisfy $C_A$, otherwise assign the logic value *true*.

## 4.8 Chapter summary

In this chapter six potential randomized algorithms for delay determination are discussed. Two of these, random test pattern generation and topological search, are based on randomized searches of the circuit input space to find a vector which sensitizes a critical path in the circuit. Failing to find such a vector results in the algorithm producing an underestimate of the true delay of the circuit.

The other four algorithms considered use an analysis circuit, based on the original circuit logic, to improve known bounds on the true delay of the circuit. These four algorithms are referred to as CNF translation, implication propagation, path tracing, and integer programming.

Two of the six algorithms described in this chapter are studied in greater detail in later chapters. The topological search is a very simple yet often very effective algorithm, and is developed and analyzed in chapter 5. The CNF translation algorithm addresses some of the key concerns raised by McGeer and Brayton in [38] with respect to the use of algorithms based on the analysis circuit, and is developed and analyzed in chapter 6. The RTPG algorithm is the simplest of the six randomized techniques, and is briefly considered in comparison with the topological search technique in chapter 5. The path tracing technique has only

been presented here at a conjectural level, and considerable work is required to show its viability. The remaining two techniques, based on integer programming and implication propagation, are regarded by the author as highly promising, but also require extensive development. This development is left as a subject for future work.

# Chapter 5

# Topological search

In this chapter the topological search algorithm is developed and analyzed in detail. This algorithm involves a directed search of the input space of a circuit in an attempt to find input vectors which sensitize critical paths in the circuit.

Section 5.1 of this chapter contains a description of the algorithm, plus discussion of some efficiency issues involved in its implementation. In sections 5.2 through 5.5 a detailed analysis of the algorithm is given, while comparisons with other existing techniques are supplied in section 5.5. Both the algorithm and the analysis are summarized in section 5.6.

## 5.1 The algorithm

In this section the basic topological search algorithm is described, along with the use of hash tables as a technique to make the algorithm more efficient.

### 5.1.1 Basic algorithm and implementation

As stated in chapter 4, the topological search algorithm involves a search of the input space of a circuit starting from a series of random[1] input vectors. From each starting vector, the search proceeds by determining the delay of the circuit (using floating delay mode) given the input vector, then determining the delay associated with each input vector adjacent to the starting vector. The process is called recursively on every adjacent vector which has an associated delay greater than that of the current vector. (Note that to determine the delay associated with an input vector, the algorithm requires some form of simulator for the circuit.) Pseudo-code for the algorithm is given below:

```
Maxdelay = 0;
Repeat for a predetermined number of starting input vectors:
   1) generate a pseudo-random input vector, v
   2) local_maxdelay = Find_Local_Max(v)
   3) if (local_max_delay > Maxdelay) then Maxdelay = local_max_delay

Find_Local_Max(input vector v)
   1) this_delay = get_delay(v)
      current_max = this_delay
   2) for all vectors, v_adj, adjacent to v:
      (i) adj_delay = get_delay(v_adj)
      (ii) if (adj_delay > this_delay) then
              adj_delay = Find_Local_Max(v_adj)
      (iii) if (adj_delay > current_max) then
              current_max = adj_delay
   3) return(current_max)

Where the get_delay(v) routine simulates vector v on the
circuit to determine the floating delay associated with v.
```

The algorithm thus searches out all vectors which are in the input space local to the starting vector and whose delays are local maxima. The maximum delay

[1] Random values in the actual implementation of the algorithm were generated using the Unix "random()" pseudo-random number generator.

of a circuit is found by detecting an input vector which is a global maxima for the input space. This occurs once a search is begun at an input vector, $X_{start}$, which is on the "hill" surrounding a global maximum, $X_{max}$. That is, the algorithm detects the maxima if there exist a set of input vectors, $X_1, \ldots, X_k$, such that all of the following conditions are satisfied:

1. for $i \in 2 \ldots k$: the delay associated with $X_i$ is greater than the delay associated with $X_{i-1}$ and $X_i$ is adjacent to $X_{i-1}$
2. $X_1$ is adjacent to $X_{start}$ and $X_1$ has a greater associated delay than $X_{start}$
3. $X_{max}$ is adjacent to $X_k$ and $X_{max}$ has a greater associated delay than $X_k$

A detailed analysis of the algorithm is given in sections 5.2 through 5.5, and some key efficiency problems with the algorithm are addressed in section 5.1.3, but first we consider the modifications to the algorithm needed to produce a topological search for the shortest sensitizable path in the circuit.

### 5.1.2 Shortest sensitizable paths

In chapters 2 and 4 it is mentioned that the algorithm presented here is easily modifiable to produce a topological search for the shortest sensitizable path in a circuit. This section contains discussion of the modifications to the algorithm of section 5.1.1 needed to detect shortest sensitizable paths.

The only changes required are in switching from tracking and detecting maxima to tracking and detecting minima throughout the algorithm. The pseudo-code therefore becomes:

```
Mindelay = 0;
Repeat for a predetermined number of starting input vectors:
   1) generate a pseudo-random input vector, v
   2) local_mindelay = Find_Local_Min(v)
   3) if (local_min_delay < Mindelay) then Mindelay = local_min_delay

Find_Local_Min(input vector v)
   1) this_delay = get_delay(v)
      current_min = this_delay
   2) for all vectors, v_adj, adjacent to v:
      (i) adj_delay = get_delay(v_adj)
      (ii) if (adj_delay < this_delay) then
              adj_delay = Find_Local_Min(v_adj)
      (iii) if (adj_delay < current_min) then
               current_min = adj_delay
   3) return(current_min)

Where the get_delay(v) routine simulates vector v on the
circuit to determine the floating delay associated with v.
```

In fact, a close bound on the difference between output stabilization time and minimum destabilization time can be quickly determined through the use of the topological search algorithm. In chapter 7, this is shown to be well suited for use in the identification of critical components for circuit timing optimization and fault sensitivity.

In the rest of this chapter our analysis focuses on the topological search for a longest sensitizable path in a circuit, though comparable results hold for the search for a shortest sensitizable path.

### 5.1.3 Efficiency issues

One key efficiency problem with the search technique as described in section 5.1.1 is that search spaces often overlap in the execution of the algorithm. Suppose a vector, $X$, is adjacent to several other vectors, $X_1, X_2, X_3$, and has an associated

delay greater than each of them. If the Find_Local_Max routine described above is called on each of $X_1, X_2, X3$ then as part of their own searches, *each* of them in turn will initiate a search that starts at vector $X$.

The implementation problem is to reduce the number of duplicate searches of the input space *without* prematurely truncating any branch of the search, and without creating excessive requirements for the storage of search information. For a circuit with several hundred inputs it is clearly infeasible to maintain a complete record of which input vectors have already been considered.

As a compromise between storage limitations and the desire not to truncate valid searches, the technique adopted here is to use a hash table of fixed size, $H$, to record which input vectors have been searched for the current random starting vector. A hash function is applied to the bitstring corresponding to the current input vector, and maps a value in the range $0 \ldots 2^m - 1$ into the range $0 \ldots H - 1$ (recall $m$ is the number of inputs to the circuit). If an input vector is re-visited, this is revealed by the contents of the hash table and the search can be truncated at this point.

Searches *can* be prematurely cut in such an approach. Given two vectors, $X_i, X_j$, in which the delay associated with $X_j$ is greater than that associated with $X_i$, a search can be prematurely truncated if all the following hold:

1. $X_i$ and $X_j$ hash to the same value,
2. there exists a chain of adjacent vectors from $X_i$ to $X_j$ with monotonically increasing associated delays,
3. there is no other valid chain of vectors from the random start vector to $X_j$ which is not also prematurely cut by hash aliasing.

Because adjacent vectors are considered so frequently in the algorithm, it is also important that the hash function not weight any elements of the input bit vector "heavier" than any other elements. The hash function implemented (based on [29]) is $((MX+N)mod(P))mod(H)$ where $M$, $N$, $P$ are all large primes, $X$ is the input vector, and $H$ is the number of entries in the hash table.

It seems obvious that the hash table should be made as large as can be practically permitted for the system on which the program is being run - the larger the hash table the smaller the chance of premature search cuts. An interesting question, however, is whether allowing premature cuts (i.e. reducing the size of the hash table) may speed up the search process: is it beneficial to search a smaller portion of the input space from a larger number of random starting points? Some preliminary experimentation has been carried with regards to this, but the results at this point are inconclusive.

Theoretical analysis of the algorithm is contained in the following sections, 5.2 - 5.4, observed behaviour over a set of benchmark circuits is provided in section 5.5, and comparisons with other algorithms are provided in section 5.6.

## 5.2 Theoretical analysis

In the preceding section, 5.1, the topological search algorithm is developed and possible modifications to improve the efficiency of the algorithm are described. This section provides an overview of the theoretical analysis provided in sections 5.3 and 5.4.

In sections 5.3 and 5.4 the algorithm is modeled and analyzed with respect to worst-case and expected-case operation, and in section 5.5 the expected case

behaviour is compared to observed behaviour over a set of benchmark circuits.

Most of the analysis is based on a simplified model of circuit delay behaviour (quite distinct from the modeling of delay behaviour at the gate level discussed in chapter 3), in which it is assumed that every input vector produces a unique delay. Given this assumption, we are able to make certain predictions about both the expected nature of the input space for an unknown circuit and the expected behaviour of the algorithm for the circuit.

Note that the number of vectors examined by the algorithm from any random starting vector is bounded above by the size of the hash table. The time taken to examine an individual vector is $O(mL)$, where $m$ is the number of circuit inputs and $L$ is the number of lines in the circuit. This result is obtained as follows: the time taken to simulate the effects of the input vector on the circuit is linear in the number of lines in the circuit, and the number of simulations which must be performed is one for the vector itself, plus one for each of the $m$ adjacent vectors. Thus if the size of the hash table is $H$, then the time taken to search the input space starting at any random input vector is $O(HmL)$.

In further analysis of the worst and expected cases, the analysis is restricted to considering the number of random starting vectors needed to find the maximum true delay of the circuit, with the understanding that the search from each such starting vector can require $O(HmL)$ computations.

## 5.3 Theoretical worst case

In this section the worst case scenario for the algorithm is described. The circumstances under which the algorithm is least successful at locating an input vector

which sensitizes a critical path is the case in which:

1. only a single vector, $X$, sensitizes the critical path(s) through the circuit, and
2. every vector adjacent to $X$ is a local minima with respect to its associated delay.

In such a case, the only way in which the true maximum delay of the circuit can be determined is if one of the random starting vectors either *is* $X$, or is one of the $m$ vectors adjacent to $X$. Thus the probability that the true maximum delay of the circuit has been found after $t$ random starting vectors have been processed is given by

$$1 - (1 - \frac{m+1}{2^m})^t.$$

In such a case, to guarantee at least a $0 < (1 - \alpha) < 1$ chance that the true maximum delay has been found requires that the number of random starting vectors processed by the algorithm be at least:

$$t \geq \frac{log_2(1-\alpha)}{log_2(2^m - m) - m}.$$

For most circuits, however, not all input variables play a role in the sensitization of the critical path. In such cases, there is a set of $2^{m-k}$ input vectors which sensitize the critical path, where $k$ is the number of input variables which do not contribute to the critical path.

The worst case for such a circuit would be that in which every vector adjacent to, but not part of, the cube of $2^{m-k}$ sensitizing vectors is a local minima. There are $k2^{m-k}$ such adjacent vectors, $k$ adjacent to each vector in the cube of sensitizing vectors. In this case, the probability that a true maximum delay is detected given

$t$ random starting vectors is given by

$$1-(1-\frac{(k+1)2^{m-k}}{2^m})^t = 1-(1-\frac{k+1}{2^k})^t,$$

and to guarantee at least a $0 < 1-\alpha < 1$ chance that the true maximum delay has been found requires that

$$t \geq \frac{log_2(1-\alpha)}{log_2(2^k-k)-k}.$$

Thus, in the worst case the algorithm can require an exponential number of random starting vectors to provide a given confidence that the maximum true delay of the circuit has been detected, even without considering searches prematurely shortened due to hash aliasing.

## 5.4 Expected behaviour: theoretical model

In section 5.3, the algorithm is analyzed with respect to worst case situations given a cube of $2^{m-k}$ vectors which sensitize the critical path(s) in the circuit. In this section, a simplified model of the input space is developed and used to provide expected-case behaviour for the algorithm.

First the model is developed, and then it is used to make predictions about the behaviour of the algorithm and also about the nature of the input space for $m$-input circuits. In section 5.5, benchmark circuits are examined and the results of the analysis are compared with predicted results to analyze the accuracy of the model presented here.

### 5.4.1 Modeling the circuit input space

In this section a simple model for the input space is proposed. This model is used in subsequent sections to predict the nature of the circuit input space and to predict the behaviour of the topological search algorithm on $m$-input circuits.

The model used is simply to assume that each of the $2^m$ input vectors has a unique associated delay - i.e. the circuit outputs do not require exactly the same amount of time to stabilize for any two different input vectors. The model makes analysis much simpler, as is shown below, and is not unreasonable given the nature of physical devices[2].

The actual values of the delays associated with the input vectors are unimportant with respect to the topological search algorithm. Only the relative rank of the delays is significant. For this reason, we describe delay values by their relative ranks, $1 \ldots 2^m$.

The problem of analyzing the delay topography of the input space for $m$-input circuits can now be expressed as a problem of analyzing how the values $1 \ldots 2^m$ can be assigned to the vertices of an $m$-cube.

### 5.4.2 Expected circuit and algorithm behaviour

In this section, the model of section 5.4.1 above is used to predict several features of the input space of $m$-input circuits, and these features are used to predict the behaviour of the topological search algorithm. First, definitions are provided for several terms used in the following sections:

[2]This technique for modeling circuit delay behaviour is less useful if used in conjunction with one of the simple gate delay models discussed in chapter 3. Using simple gate delay models, such as the unit delay model, many input vectors produce identical circuit stabilization times. This directly conflicts with a circuit model which assumes distinct stabilization times.

**local maxima:** A maximal vector, or local maxima, is a vector whose associated delay is not less than the associated delay of any vector adjacent to it.

**global maxima:** A global maxima is a vector whose associated delay is not less than the associated delay of any other vector in the input space.

**local minima:** A minimal vector, or local minima, is a vector whose associated delay is not greater than the associated delay of any vector adjacent to it.

**global minima:** A global minima is a vector whose associated delay is not greater than the associated delay of any other vector in the input space.

**local peak:** A local peak is a set of $2^k$ local maxima which form a $k$-cube and which cannot be combined with any other set of $2^k$ local maxima to form a $(k+1)$-cube.

**global peak:** A global peak is a set of $2^k$ global maxima which form a $k$-cube and which cannot be combined with any other set of $2^k$ global maxima to form a $(k+1)$-cube.

**hill:** A hill is the set of vectors which form a peak, plus all those vectors connected to at least one vector in the peak by at least one chain of successively adjacent vectors with monotonically increasing associated delays (i.e. all the vectors which, if used as a starting point for the topological search, have the potential to find the peak).

**hillsize:** The hillsize is the total number of vectors in a hill (including the vectors in the peak).

Given the above definitions, and the circuit delay model of section 5.4.1, the following values and behaviours are predicted:

**The expected number of local maxima:** Let $M_{local}$ denote the expected number of local maxima. If $l$ represents the delay level associated with a particular input vector, then the probability that $l$ is a local maxima is 0 if $l \leq m$, and otherwise is given by

$$\frac{\binom{l-1}{m}}{\binom{2^m-1}{m}} = \prod_{j=1}^{m} \frac{l-j}{2^m-j}$$

Taking the sum of this value across all $l \in 1 \ldots 2^m$, gives the following formula for the expected number of local maxima:

$$M_{local} = \sum_{l=m+1}^{2^m} \prod_{j=1}^{m} \frac{l-j}{2^m - j} = \frac{2^m}{m+1}.$$

**The expected number of worst case local maxima:** Let $M_{worst}$ denote the expected number of worst-case local maxima. The expected number of local maxima which are surrounded by local minima can be obtained as follows: Given that a vector is a local maxima, the probability that any vector adjacent to it is a local minima is given by

$$2^{-m} \sum_{l=m+1}^{2^m} \prod_{j=1}^{m-1} \frac{l-j}{2^m - j}$$

and thus the expected number of worst case local maxima is given by

$$M_{worst} = \left( 2^{-m} \sum_{l=m+1}^{2^m} \prod_{j=1}^{m-1} \frac{l-j}{2^m - j} \right)^m \frac{2^m}{m+1}$$

**The expected average hillsize:** Let $H_a$ denote the expected average hillsize. The size of the average hill can be calculated as shown below. Given the following:

- a local maxima, $X_m$, with associated delay of rank $P$,
- a vector, $X$, whose minimal distance from the local maxima is exactly $d$,
- a path of length $l$ from $X_m$ to $X$,

then the number of possible ways in which monotonically-decreasing delay assignments may be made to the vectors along that path is $\binom{P}{l}$, and the total number of ways delay assignments may be made is given by $\binom{2^m}{l}(l!)$. Let $Paths(l, d)$ denote the number of acyclic paths of length exactly $l$ which exist between two vectors of a hypercube, where the minimum distance between

the two vectors is exactly $d$. Then the chance that no valid length $l$ path exists between $X_m$ and $X$ is given by

$$\left[1 - \prod_{i=0}^{l-1} \frac{P-i}{(l-i)(i+1)}\right]^{Paths(l,d)}.$$

If we further note that the number of vectors of minimum distance exactly $d$ from $X_m$ is given by $\binom{n}{d}$, then the expected number of vectors in the hill for which $X_m$ is the local maxima is given by

$$Hillsize(P) = \sum_{d=1}^{m} \binom{m}{d} \left(1 - \prod_{l=d}^{P} \left[1 - \prod_{i=0}^{l-1} \frac{P-i}{(l-i)(i+1)}\right]^{Paths(l,d)}\right).$$

The average number of vectors in the hill "below" any given vector can therefore given as

$$2^{-m} \sum_{P=1}^{2^m} Hillsize(P).$$

The problem of solving $Paths(l,d)$ remains to be solved however.

**$Paths(l,d)$ transformed:** An alternative expression to the $Paths(l,d)$ problem can be stated as follows: suppose we are given a string, $S$, with the following restrictions:

- the length, $l$, of the string is given by $l = (2i + d)$,
- the characters of the string are drawn from an alphabet of size $n$,
- exactly $d$ of the characters in the alphabet appear in the string an odd number of times,
- no non-empty substring of $S$ contains an even number of each character in the alphabet (it is permissible for characters to appear 0 times in $S$).

We wish to answer the question: how many such $S$ exist?

**$Paths(l,d)$, lower bound:** until a solution to this problem is found, the estimator used is to only consider the paths of length $l = d$, for which we

know $Paths(l,d) = d!$. Note that this cannot overestimate the size of a hill, it can however underestimate the size of a hill.

**The expected hillsize for global peak:** let the expected hillsize for the global peak be denoted $H_P$. (Remember that under this model there exists only a single global peak, consisting of one vector with an associated delay of rank $2^m$.) Given the analysis for expected average hillsize above, the (underestimated) expected hillsize for the global peak can be expressed as

$$H_P = \sum_{d=1}^{m} \binom{m}{d} (1 - (1 - \frac{1}{d!})^{d!}).$$

**The expected number of start vectors required by topological search:** Let $T_e$ denote the expected number of start vectors required by topological search.

$T_e$ can be calculated simply as

$$T_e = \frac{2^m}{H_P}$$

**The expected number of start vectors required given a worst-case circuit:** The absolute worst case for the algorithm is discussed in section 5.3, and is denoted $T_w$. In this section we describe an adjusted worst case, $T'_w$, in which it is assumed the number of global maxima is given by $M_{global} \geq 1$ Let $T'_w$ denote the expected number of start vectors required given an adjusted-worst case circuit. If the number of global maxima is given by $M_{global}$, then the worst case arises when the vectors form a $k$-cube, where $k = log_2(M_{global})$. In this case, the size of the global peak is $(m+1-k)M_{global}$, and the expected number of start vectors required is $T'_w = \frac{2^m}{(m+1-k)M_{global}}$ (the absolute worst case being the one in which $M_{global} = 1$).

### 5.4.3 Edge-differences and implications in delay modeling

Given that we have assigned the delay ranks $1 \ldots 2^m$ to the vectors of the $m$-cube, the results of two theoretical works by Harper [23] [24] can be directly applied to analyze circuit delay behaviour.

Harper's works examine the different ways in which values $1 \ldots 2^m$ can be assigned to the vertices of a hypercube. For every pair of vertices in the hypercube, the **edge-difference** for that pair is the absolute value of the difference in the values assigned to the pair. Harper considers the sum of all edge-differences in the hypercube, and determines what the maximum possible sum is (for any assignment of values to vertices), what the minimum possible sum is, and what the minimum worst-case edge difference is. That is, out of all possible labelings, if we consider the largest edge-difference in each labeling, which labeling has the smallest 'largest edge-difference'?

For our purposes, the values $1 \ldots 2^m$ correspond to the relative rankings of the delay values, and the edge-differences represent the differences in required stabilization time for adjacent input vectors. Given the simplifying assumptions that (1) we are dealing with purely combinational logic, (2) the circuit output does not fluctuate before stabilizing at a new value, and (3) only a single input bit changes at a time, then the largest edge-difference reflects a lower bound on the clock period required for the circuit, and the average edge-differences reflect a lower bound on the average stabilization time required for the circuit.

The following conclusions result from directly applying Harper's edge-difference results:

**Average rank difference:** across all possible assignments of delay ranks to input vectors, the average rank difference between two adjacent vectors is $\frac{2}{3}2^{m-1} + \frac{1}{3}$.

**Greatest combined rank differences:** the assignment of delay ranks to input vectors which maximizes the total difference in ranks between adjacent vectors gives an average rank difference (between adjacent vectors) of $2^{m-1}$.

**Smallest combined rank differences:** the assignment of delay ranks to input vectors which minimizes the total difference in ranks between adjacent vectors gives an average rank difference (between adjacent vectors) of $\frac{2}{m}2^{m-1} - \frac{1}{m}$.

**Minimal worst difference:** considering the largest difference between adjacent vectors under an ordering, the orderings which minimize this difference are the ones in which vectors of weight $i$ have delay ranks less than the delay ranks of vectors of weight greater than $i$, and the "minimum maximum-difference" is given by $\sum_{i=0}^{m-1} \binom{i}{\frac{i}{2}}$. Thus a lower bound on the minimum clock requirements under the assumptions discussed above is also given by $\sum_{i=0}^{m-1} \binom{i}{\frac{i}{2}}$.

For example, given an 8-input circuit, and assignment of delay ranks $1 \ldots 256$ to the input vectors, then:

- the expected average difference in rank between two adjacent vectors is $\frac{2}{3}2^{8-1} + \frac{1}{3} \approx 85.7$,
- the greatest the average difference could be (averaged across all adjacent pairs) is $2^{8-1} = 128$, and
- the least the average difference could be is $\frac{2}{8}2^{8-1} - \frac{1}{8} = 31.875$.

Thus, if every input vector has a unique associated delay, then *regardless of the circuit implemented* the average rank difference between adjacent vectors in the input space falls between $2^{m-1}$ and $\frac{2^m-1}{m}$. The average stabilization time required between adjacent vectors thus also falls in this range.

Furthermore, any circuit which has a minimal worst-case delay difference between adjacent input vectors has the following property: there exists an $m$-bit Boolean vector $B$, such that for every pair of input vectors, $X_i, X_j$, to the circuit, the delay of $X_i$ is less than the delay of $X_j$ *iff* the weight of $X_i \oplus B$ is less than the weight of $X_j \oplus B$.

In section 5.5 each of the values predicted for circuits of a given size[3] is compared with values found by analyzing a number of benchmark circuits of that size.

## 5.5 Expected results vs. observed behaviour

In this section the predicted results from sections 5.3 and 5.4 are compared with results obtained by analyzing both the input space of a set of benchmark circuits, and the behaviour of the topological search algorithm when run on these circuits.

Three tables are used to summarize the data collected. The first table, 5.1, summarizes the number of local maxima and number of local peaks predicted for the benchmark circuits based on the techniques of section 5.4, the number of primary inputs to each circuit, and also the number of maxima and peaks actually observed while analyzing a series of benchmark circuits. The other two tables, 5.2 and 5.3, summarize the expected number of random starting vectors required for

[3]Where size in this case is measured by the number of circuit inputs.

the topological search algorithm to detect the maximum sensitizable delay in the benchmark circuits, the values predicted for both the worst case and the expected case given the number of inputs to each circuit, and the average number of vectors the algorithm actually required during sample runs.

Table 5.1: Predicted/actual numbers of maxima for small benchmarks

| circuit | predicted ($M_{local}$) maxima | observed local maxima | observed local peaks | observed global maxima | observed global peaks | predicted hillsize average ($H_a$) | predicted hillsize peak ($H_P$) | observed hillsize ∀ glb. peaks |
|---|---|---|---|---|---|---|---|---|
| b1 | 2 | 2 | 2 | 1 | 1 | 4.3 | 4.7 | 6.0 |
| cm42a | 3.2 | 4 | 1 | 4 | 1 | 5.6 | 6.7 | 16.0 |
| majority | 5.3 | 4 | 3 | 1 | 1 | 7.0 | 9.2 | 22.2 |
| C17 | 5.3 | 2 | 1 | 2 | 1 | 7.0 | 9.2 | 32.0 |
| decod | 5.3 | 16 | 1 | 16 | 1 | 7.0 | 9.2 | 32.0 |
| cm82a | 5.3 | 2 | 2 | 1 | 1 | 7.0 | 9.2 | 31.2 |
| cm138a | 9.1 | 8 | 1 | 8 | 1 | 8.7 | 12.6 | 64.0 |
| z4ml | 16.0 | 6 | 6 | 2 | 2 | 10.7 | 17.3 | 89.8 |
| f51m | 28.4 | 19 | 9 | 1 | 1 | 13.0 | 23.6 | 129.5 |
| ldd | 51.2 | 112 | 6 | 32 | 1 | 15.9 | 32.4 | 385.0 |
| 9symml | 51.2 | 66 | 57 | 1 | 1 | 15.9 | 32.4 | 198.5 |
| x2 | 93.1 | 112 | 9 | 8 | 1 | 19.4 | 44.7 | 373.0 |
| alu2 | 93.1 | 42 | 14 | 1 | 1 | 19.4 | 44.7 | 841.4 |
| cm152a | 170.7 | 208 | 3 | 128 | 1 | 23.9 | 61.7 | 1908.0 |
| cm85a | 170.7 | 118 | 48 | 16 | 8 | 23.9 | 61.7 | 1202.0 |
| cm151a | 315.1 | 154 | 8 | 32 | 1 | 29.7 | 85.6 | 2948.0 |
| cm162a | 1092.3 | 328 | 8 | 48 | 1 | 46.7 | 165.6 | 5296.0 |
| cu | 1092.3 | 1984 | 3 | 64 | 1 | 46.7 | 165.6 | 1952.0 |
| alu4 | 1092.3 | 567 | 65 | 2 | 2 | 46.7 | 165.6 | 1824.0 |
| pm1 | 3855.1 | 7648 | 9 | 256 | 1 | 75.8 | 322.1 | 16448.0 |
| cmb | 3855.1 | 4713 | 7 | 16 | 1 | 75.8 | 322.1 | 1920.0 |
| cm163a | 3855.1 | 768 | 3 | 256 | 1 | 75.8 | 322.1 | 34048.0 |
| parity | 3855.1 | 5096 | 5096 | 2048 | 2048 | 75.8 | 322.1 | 48000.0 |
| vda | 7281.8 | 7052 | 18 | 2048 | 1 | 97.6 | 449.7 | 78976.2 |
| tcon | 7281.8 | 65280 | 1 | 65280 | 1 | 97.6 | 449.7 | 122240.1 |
| pcle | 26214.4 | 256 | 1 | 256 | 1 | 165.3 | 876.5 | 345087.5 |
| sct | 26214.4 | 63744 | 8 | 5120 | 1 | 165.3 | 876.5 | 208384.0 |
| mux | 95325.1 | 71816 | 45 | 1024 | 1 | 285.9 | 1706.5 | 202752.7 |
| cc | 95325.1 | 215280 | 8 | 117464 | 1 | 285.9 | 1706.5 | 1837060.9 |
| cm150a | 95325.1 | 55680 | 13 | 8192 | 1 | 285.9 | 1706.5 | 997375.7 |

### 5.5.1 Analysis of table 5.1

Table 5.1 shows the predicted values of several characteristics for the set of benchmark circuits, calculated as described in section 5.4, and also the observed values

for these characteristics. The characteristics described include:

- the expected number of local maxima ($M_{local}$),
- the expected average number of vectors in a hill ($H_a$), and
- the expected number of vectors in the global peak's hill ($H_P$).

The corresponding values obtained through analysis/observation include:

- the actual number of local maxima in the benchmark circuit,
- the number of peaks formed by these local maxima,
- the number of global maxima in the circuit,
- the number of peaks formed by these global maxima, and
- the actual hillsize for the combined global peaks in the circuit.

In examining the data contained in table 5.1, the following observations are made:

**Local maxima:** the $M_l$ estimator seems to provide a rough estimate of the number of local maxima in the circuit (within an order of magnitude).

**Relationship of local maxima to global maxima and peaks:** the number of peaks observed (both global and local) is generally *much* lower than the number of local maxima - frequently by at least one order of magnitude. Since the estimates of local maxima seem reasonable, this would imply that the maxima in the benchmark circuits are more tightly grouped into peaks than would be the case in "random" circuits. This in turn implies that in random circuits it is much more likely that all (or most) inputs contribute to longest paths, as grouping of maxima into cubes implies the maxima are insensitive to a subset of the circuit inputs.

**Hillsizes:** the worst case estimators for hill sizes are extremely pessimistic estimators of the actual hillsize for the global peak(s). Thus these estimators also provide extremely pessimistic estimates for the required number of starting vectors for the topological search algorithm.

Having compared the predicted and actual topography of the input space of the benchmark circuits, the expected number of starting vectors required for the topological search algorithm (as predicted in several ways in section 5.4.1) is now compared with the average number actually required as observed over the set of benchmark circuits.

### 5.5.2 Analysis of data (Tables 5.2, 5.3)

Tables 5.2 and 5.3 contain predicted values for random circuits of $m$ inputs, as obtained by the methods outlined in section 5.4.1, plus the values observed through simulation on a number of benchmark circuits. Table 5.2 provides values for a set of smaller benchmarks circuits, for which exhaustive analysis of the input space is performed, while table 5.3 provides similar values for a set of larger benchmarks. For the larger benchmarks only a random subset of the input space is considered. The predicted values include:

- the expected number of random start vectors required in the worst case, $T_w$ (only a single global maximal vector exists, and all vectors adjacent to it are local minima),
- the expected number of random start vectors required assuming a worst case adjusted for the true number of global maxima ($M_{global}$), this is $T'_w$,
- the expected number of vectors required assuming the hillsize for the global peak is the value $H_P$, calculated in section 5.4.1. The number of vectors predicted here is $T_e$ (as in 5.4.1).

Table 5.2: Number of random starting vectors required - small benchmarks

| | Observed behaviour | | expected/worst-case predictions | | |
|---|---|---|---|---|---|
| | by random | by | worst case estimates | | expect case |
| benchmark circuit | sampling (RTPG) | topological search | by number of inputs ($T_w$) | Adjusted by $M_{global}$ ($T'_w$) | by number of inputs ($T_e$) |
| b1 | 8.0 | 1.3 | 2 | 2.0 | 1.7 |
| cm42a | 4.0 | 1 | 3.2 | 6.4 | 2.4 |
| majority | 32.0 | 1.4 | 5.3 | 5.4 | 3.5 |
| C17 | 16.0 | 1 | 5.3 | 3.2 | 3.5 |
| decod | 2.0 | 1 | 5.3 | 1.0 | 3.5 |
| cm82a | 32.0 | 1.0 | 5.3 | 5.4 | 3.5 |
| cm138a | 8.0 | 1 | 9.1 | 2.0 | 5.1 |
| z4ml | 64.0 | 1.4 | 16 | 9.2 | 7.4 |
| f51m | 256.0 | 2.0 | 28.4 | 23.1 | 10.8 |
| ldd | 16.0 | 1.3 | 51.2 | 3.3 | 15.8 |
| 9symml | 512.0 | 2.6 | 51.2 | 49.9 | 15.8 |
| x2 | 128.0 | 2.7 | 93.1 | 17.0 | 22.9 |
| alu2 | 1024.0 | 1.2 | 93.1 | 98.5 | 22.9 |
| cm152a | 16.0 | 1.1 | 170.7 | 3.2 | 33.2 |
| cm85a | 128.0 | 1.7 | 170.7 | 14.5 | 33.2 |
| cm151a | 128.0 | 1.4 | 315.1 | 14.6 | 47.9 |
| cm162a | 341.3 | 3.1 | 1092.3 | 31.4 | 99.0 |
| cu | 256.0 | 8.4 | 1092.3 | 28.4 | 99.0 |
| alu4 | 8192.0 | 9.0 | 1092.3 | 585.1 | 99.0 |
| pm1 | 256.0 | 4.0 | 3855.1 | 33.3 | 203.5 |
| cmb | 4096.0 | 34.1 | 3855.1 | 258.5 | 203.5 |
| cm163a | 256.0 | 1.9 | 3855.1 | 29.2 | 203.5 |
| parity | 32.0 | 1.4 | 3855.1 | 5.1 | 203.5 |
| vda | 58.3 | 1.7 | 7281.8 | 8.5 | 291.5 |
| tcon | 2.0 | 1.1 | 7281.8 | 1.0 | 291.5 |
| pcle | 2340.6 | 1.5 | 26214.4 | 192.0 | 598.2 |
| sct | 93.6 | 2.5 | 26214.4 | 12.4 | 598.2 |
| mux | 3276.8 | 10.3 | 95325.1 | 258.5 | 1228.9 |
| cc | 17.7 | 1.1 | 95325.1 | 3.4 | 1228.9 |
| cm150a | 252.1 | 2.1 | 95325.1 | 28.1 | 1228.9 |

Table 5.3: Number of random starting vectors required - large benchmarks

| | Observed behaviour | | expected/worst-case predictions | | |
|---|---|---|---|---|---|
| | by random | by | worst case estimates | | expect case |
| benchmark circuit | sampling (RTPG) | topological search | by number of inputs ($T_w$) | Adjusted by $M_{global}$ ($T'_w$) | by number of inputs ($T_e$) |
| ttt2 | 481.9 | 4.4 | 671089.0 | 48.6 | 3635.4 |
| lal | 163.8 | 1.3 | 2485510.0 | 19.6 | 7526.3 |
| pcler8 | 1365.3 | 3.0 | 4793490.0 | 119.6 | 10849.5 |
| frg1 | 819.2 | 6.9 | 9256400.0 | 76.7 | 15663.5 |
| c8 | 15.7 | 1.8 | 9256400.0 | 3.2 | 15663.5 |
| comp | 16384.0 | 51.2 | 1.302e+08 | 1092.3 | 69384.0 |
| C6288 | >84020500 | 1024.0 | 1.302e+08 | 3074940.0 | 69384.0 |
| myadder | 546133.0 | 1.9 | 2.526e+08 | 27226.5 | 101266.5 |
| C1908 | 819200.0 | 15.5 | 2.526e+08 | 39682.5 | 101266.5 |
| term1 | 3276.8 | 12.6 | 4.909e+08 | 258.5 | 148222.4 |
| count | 819200.0 | 2.8 | 9.544e+08 | 39682.5 | 217616.8 |
| C432 | 16384.0 | 22.3 | 1.857e+09 | 1092.3 | 320540.9 |
| unreg | 16.5 | 1.4 | 1.857e+09 | 3.3 | 320540.9 |
| toolarge | 5461.3 | 17.7 | 7.048e+09 | 407.1 | 702721.4 |
| b9 | 2340.6 | 11.6 | 5.236e+10 | 192.0 | 2345474.7 |
| C499 | 16384.0 | 146.3 | 5.236e+10 | 1092.3 | 2345474.7 |
| C1355 | 4096.0 | 51.2 | 5.236e+10 | 315.1 | 2345474.7 |
| k2 | 712.4 | 16.5 | 7.649e+11 | 68.0 | 12360530.0 |
| cht | 16.4 | 1.5 | 2.932e+12 | 3.3 | 29063101.6 |
| apex7 | 11.5 | 1.2 | 1.126e+13 | 2.5 | 69423029.0 |
| C3540 | 409600.0 | 78.8 | 2.208e+13 | 20851.3 | 107923383.6 |
| x1 | 2048.0 | 17.1 | 4.330e+13 | 170.7 | 168415527.7 |
| C880 | 819200.0 | 102.4 | 1.890e+16 | 39682.5 | 1.086156e+10 |
| example2 | 248.2 | 1.4 | 4.498e+23 | 27.7 | 3.822879e+15 |
| x4 | 18.6 | 1.2 | 2.085e+26 | 3.6 | 5.181088e+17 |
| apex6 | 5461.3 | 28.4 | 3.203e+38 | 407.1 | 9.492872e+27 |
| x3 | 8192.0 | 8.6 | 3.203e+38 | 585.1 | 9.492872e+27 |
| rot | >84020500 | 102.4 | 3.203e+38 | 3074940.0 | 9.492872e+27 |
| frg2 | 565.0 | 2.1 | 7.743e+40 | 55.7 | 1.135396e+30 |
| pair | >84020500 | 113.8 | 6.881e+49 | 3074940.0 | |
| C5315 | 8192.0 | 6.2 | 2.140e+51 | 585.1 | |
| C7552 | 2048.0 | 2.8 | 9.889e+59 | 170.7 | |
| C2670 | 162044.0 | 48.8 | 5.899e+67 | 9854.1 | |
| des | 23.4 | 1.4 | 4.506e+74 | 4.2 | |

The values obtained through observation/simulation include:

- the average number of vectors which would be required using simple random test pattern generation (RTPG), i.e. $\frac{2^m}{M_{global}}$.
- the average number of start vectors required in actual runs of the topological search algorithm on the benchmark circuit (based on the average number of vectors to detect known maximum length paths for the respective circuits).

In analyzing the data contained in tables 5.2 and 5.3, the following observations are made:

**Worst case estimators:** the worst case estimator ($T_w$) is, fortunately, an extremely pessimistic estimator of the algorithm's behaviour over the benchmark circuits. The adjusted worst case ($T'_w$), which takes the number of global maxima into account, is a much more reasonable estimator. Frequently this is within an order of magnitude of the average number of vectors required during runs of the algorithm on the benchmark set. Unfortunately, the number of global maxima is a quantity which is rarely known in advance.

**Expected case estimator:** the expected case estimator ($T_e$) is also an extremely pessimistic estimator of the number of random starting vectors required by the algorithm. In practice, even RTPG (with no searching of the input space beyond the given starting vectors) generally outperforms the expected case estimator.

**Larger circuits:** the estimators become increasingly inaccurate as the number of inputs to the circuits increase. This, combined with the relatively low number of starting vectors required by the topological search algorithm even for circuits with hundreds of inputs, suggests that the circuits being designed (or at least the circuits which comprise the benchmark set) have a very structured input space.

### 5.5.3 Summary of model implications and predictions

In summary, the main observations drawn from the tables and data presented in this section are as follows:

- The number of local maxima predicted by the model appears to be a reasonable estimate of the number of locally maximal input vectors.

- The number of local maxima, however, is much higher than the the number of peaks in the input space of the benchmark circuits. This implies that in random circuits most circuit inputs contribute to most delay paths, whereas in "practical" circuits delays are often insensitive to a subset of the circuit inputs.

- The model grossly underestimates the size of hills associated with global maxima, and as a result significantly overestimates the number of starting vectors required by the topological search algorithm. Even the expected search estimate, $T_e$, overestimates the required number of vectors by orders of magnitude.

- The frequency of worst-case situations, both as predicted by the model and as encountered in analyzing the benchmark circuits, is extremely low.

- The observed number of starting vectors required by the topological search algorithm increases very slowly as the number of circuit inputs increase. This suggests that, while the circuits are becoming much larger, the complexity of the input space is growing at a much slower pace.

- Using the model and the assumption that circuit outputs do not fluctuate before stabilizing, Harper's results on edge-differences provide us with the average stabilization times required for transitions between adjacent input vectors, and also provides us with a guaranteed lower bound on required clock speed: that bound being $\sum_{i=0}^{m-1} \binom{i}{\frac{i}{2}}$.

In section 5.6, the topological search algorithm is compared with a number of existing techniques for determining maximum circuit delays.

## 5.6 Comparison with other algorithms

In this section the topological search algorithm is compared with other algorithms for determining maximum true circuit delay. The comparisons are divided into three sections:

1. In section 5.6.1 the algorithm is compared to algorithms which function by enumerating and checking long paths within the circuit.
2. In section 5.6.2 the algorithm is compared to algorithms which do not require enumeration or checking of specific paths in the circuit.
3. In section 5.6.3, the computation times required by the topological search algorithm for a set of benchmark circuits is compared to the published CPU times required for by a number of existing algorithms.

### 5.6.1 Path-checking algorithms

Path checking algorithms, algorithms which operate by enumerating the longest paths in a circuit and checking each path in turn to see if it is sensitizable, are clearly superior to the topological search algorithm if one of the longest paths in the circuit is sensitizable, i.e. there are no (or very few) long false paths. In such a case the path-checking algorithm can terminate almost immediately.

On the other hand, for circuits with a very large number of long false paths, the path-checking algorithms perform very poorly. It is important to note that much of the published work in the area has been performed using a unit delay model which does not distinguish between rising and falling delays. Under such models the number of long false paths is often much lower than is the case with

a more realistic model. Thus many of the path-checking models may be even less effective if a more accurate delay model is used.

### 5.6.2 Non-enumerative algorithms

In an attempt to circumvent the problem of dealing with thousands of long false paths, many of the most recent published works deal with non-enumerative algorithms. Algorithms in which the logic of the circuit is analyzed without specifically examining individual paths. These algorithms are proving to be much more adept at handling large circuits, as is demonstrated in table 5.5. The topological search algorithm does *not* appear to be competitive with several of these more recently published algorithms.

### 5.6.3 Comparison through benchmarks

In this section the average CPU time required by the topological search algorithm for the circuits in the benchmark set is compared with the times required by random test pattern generation (RTPG) and a number of existing algorithms. In both cases, the average time to detection is based on tests applied versus the known maximum length paths in the circuit.

Table 5.4: RTPG vs topological search - CPU times (seconds)

| circuit | RTPG time | Top. search | circuit | RTPG time | Top. search |
|---|---|---|---|---|---|
| b1 | 0.001 | 0.001 | pcler8 | 0.877 | 0.129 |
| C17 | 0.001 | 0.001 | vda | 0.915 | 11.888 |
| cm138a | 0.001 | 0.003 | pcle | 1.053 | 0.065 |
| decod | 0.001 | 0.005 | cmb | 1.060 | 0.430 |
| majority | 0.002 | 0.001 | ttt2 | 1.123 | 3.332 |
| cm152a | 0.002 | 0.006 | mux | 1.910 | 0.632 |
| cm82a | 0.005 | 0.004 | alu2 | 2.043 | 3.229 |
| cm42a | 0.006 | 0.003 | b9 | 2.167 | 1.762 |
| tcon | 0.011 | 0.019 | des | 2.959 | 139.828 |
| cc | 0.018 | 0.042 | x1 | 6.965 | 34.305 |
| parity | 0.021 | 0.046 | frg2 | 8.102 | 41.956 |
| cm151a | 0.023 | 0.019 | term1 | 12.840 | 43.056 |
| unreg | 0.025 | 0.104 | comp | 18.020 | 8.930 |
| cm85a | 0.026 | 0.054 | C1355 | 18.760 | 54.510 |
| ldd | 0.029 | 0.110 | k2 | 20.305 | 670.908 |
| z4ml | 0.036 | 0.046 | C432 | 29.820 | 13.961 |
| c8 | 0.042 | 0.605 | apex6 | 36.510 | 164.360 |
| cht | 0.051 | 0.550 | toolarge | 44.090 | 244.487 |
| apex7 | 0.053 | 0.659 | alu4 | 50.320 | 17.794 |
| x2 | 0.063 | 0.040 | C7552 | 63.148 | 256.367 |
| cm163a | 0.074 | 0.024 | C499 | 70.060 | 343.771 |
| cm162a | 0.092 | 0.051 | x3 | 76.290 | 172.948 |
| sct | 0.095 | 0.263 | C5315 | 182.770 | 433.094 |
| cu | 0.117 | 0.197 | myadder | 843.363 | 1.734 |
| cm150a | 0.119 | 0.109 | count | 910.010 | 1.058 |
| pm1 | 0.128 | 0.055 | C2670 | 1796.800 | 704.049 |
| x4 | 0.138 | 3.563 | C880 | 2600.010 | 91.920 |
| lal | 0.198 | 0.378 | C1908 | 4625.090 | 76.596 |
| f51m | 0.258 | 0.301 | C3540 | 4850.130 | 558.561 |
| frg1 | 0.715 | 1.016 | rot | >536410.000 | 1018.390 |
| example2 | 0.751 | 2.218 | pair | >1442560.000 | 3189.610 |
| 9symml | 0.772 | 1.495 | C6288 | >2136410.000 | 3378.760 |

In table 5.4, the average running time of the topological search algorithm is compared with the average running time of RTPG. The entries are ordered on the computation time required by the RTPG search algorithm. In only 25 of the 64 benchmarks is the behaviour of the topological search superior to that of RTPG. The significant difference is in the extreme behaviour, as can be seen from the cases in which the RTPG algorithm failed to complete in over 48 hours[4], while the average time to complete for the topological search algorithm was less than 1 hour for all benchmarks considered.

RTPG, on the other hand, in several cases did not complete in over 48 hours.

Thus the observed behaviour of the algorithm is worse than that of RTPG in more than half of all cases, but the worst case behaviour of the algorithm appears to be far superior to that of RTPG.

### Comparisons with existing techniques

In table 5.5, the topological search algorithm is compared to a number of existing algorithms for determining maximum true circuit delays. The techniques examined in table 5.5(a) are based on path-enumeration or path sensitization, whereas the techniques examined in table 5.5(b) are based on functional analysis of the circuit - similar in some ways to the CNF translation approach discussed in chapter 6.

The average time to detection for the topological search algorithm is based on frequency of detection of the known maximum length paths in the circuit versus

[4]C6288, one of the two instances, is a 16 by 16 multiplier which contains several stuck-at-fault redundancies, many paths of equal lengths, and many long false paths.

the length of CPU time for which the algorithm is run. Clearly, such a priori knowledge of maximum path length would not be available during analysis of a "new" circuit, and development of a stopping criteria is required.

The 9 ISCAS [5] benchmark circuits are chosen for the comparison, as these are the circuits for which the most published analysis results are available. Details on these 9 benchmark circuits are included in appendix 1. All algorithms listed succeeded in determining the length of the longest sensitizable path in the circuits[5]. Memory requirements for most of the algorithms have not been published, so the comparisons made here are based only on published computation times, without normalization based on processor type.

### 5.6.4 Choice of delay model

For the purposes of accurate timing comparisons, the topological search algorithm has been re-run on the comparison benchmarks using the simple unit delay model. This is the model used by the other techniques to produce the timing results presented below, and significantly reduces the complexity of the analysis. As described in chapter 3, this model is inadequate for accurate description of circuit behaviour, and frequently characterizes the longest topological paths in a circuit as sensitizable when in fact they are identified as false paths using more realistic models. Thus the efficiency of path-checking algorithms is artificially high when this model is used, and the logic analysis performed by other algorithms is also artificially simplified. This "artificial efficiency" is also exhibited in our algorithm when run with the simple model. For example, with benchmark circuit C432, the algorithm completion time using a more realistic delay model (distinct rise and

[5]Except for two algorithms which ran out of memory while analyzing circuit C6288, and were unable to complete.

fall delays for each gate type, and each input to a gate) the algorithm requires an average of 13.9 CPU seconds to locate a global maximum, whereas when using the simple unit delay model the algorithm requires on average only 0.5 CPU seconds. As no published results are available for more realistic models, I have chosen to perform the benchmark comparisons of my algorithm with existing techniques using the simple (unrealistic) delay model. Timing results were obtained running the search program on a sparc 10 workstation.

Table 5.5: Time comparisons with existing techniques

| (a) | Topological | path enumerative or search/sensitization techniques | | | | | | |
|---|---|---|---|---|---|---|---|---|
| circuit | search | [15] | [33] | [25] | [27] | [30] | [20] | [47] |
| C6288 | 772.9 | 802.6 | 16763.0 | 19729.9 | • | • | 34416.0 | 4810.0 |
| C1908 | 8.3 | 3674.5 | 1774.4 | 655.0 | 39468.4 | 12140.0 | 11538.0 | 72.8 |
| C432 | 0.4 | 0.1 | 0.5 | 348.6 | 54.5 | 1815.6 | 2.0 | 0.4 |
| C499 | 1.0 | 0.4 | 0.1 | 4.8 | 36.7 | 191.6 | 4.0 | 1.6 |
| C1355 | 0.3 | 0.4 | 0.1 | 427.6 | 32.9 | 376.0 | 63.0 | 0.7 |
| C3540 | 459.1 | 181.6 | 373.4 | 969.7 | 0.1 | 592.8 | 33804.0 | 20.0 |
| C880 | 9.9 | 0.6 | 0.1 | 609.5 | 15.8 | 3.7 | 3.0 | 0.2 |
| C5315 | 118.8 | 5.2 | 7.8 | 1374.9 | 62.0 | 455.6 | 311.0 | 22.5 |
| C7552 | 93.7 | 5.9 | 1.6 | 1753.6 | 4.7 | 2.9 | 822.0 | 14.8 |
| C2670 | 552.8 | 200.2 | 14.2 | 1800.9 | 59.1 | 1.8 | 813.0 | 1150.0 |

• could not complete - insufficient memory

| (b) | Topological | functional analysis techniques | | | | |
|---|---|---|---|---|---|---|
| circuit | search | [40] | [26] | | [9] | [36] |
| C6288 | 722.9 | 33.6 | 69.3 | (96%) | 25.7 | 196.9 |
| C1908 | 8.3 | 8.2 | 9.4 | (100%) | 31.6 | 18.1 |
| C432 | 0.4 | – | – | | 1.3 | 2.4 |
| C499 | 1.0 | – | – | | 1.9 | 2.9 |
| C1355 | 0.3 | – | – | | 4.8 | 7.4 |
| C3540 | 459.1 | 7.2 | 14.0 | (59%) | 39.4 | 58.4 |
| C880 | 9.9 | – | – | | 2.7 | 5.0 |
| C5315 | 118.8 | 10.5 | 20.6 | (42%) | 35.8 | 91.4 |
| C7552 | 93.7 | 6.8 | 49.6 | (44%) | 26.1 | 133.4 |
| C2670 | 552.8 | 22.4 | 8.8 | (35%) | 11.5 | 14.1 |

– results unavailable

The algorithms considered are very briefly summarized here:

- [15] - Devadas et al, 1993.

- [33] - Liu et al, 1991: uses a parallel search approach, similar to the D algorithm.
- [25] - Huang et al, 1991: uses an extended Boolean algebra.
- [27] - Ju and Saleh, 1991: uses an incremental technique for path building.
- [30] - Lam et al, 1993: applies timed Boolean functions.
- [20] - Fang and Jone, 1995: uses path sensitization as part of an optimization routine.
- [47] - Silva and Sakallah, 1993: uses concurrent path sensitization.
- [40] - McGeer et al, 1991: uses circuit analysis and path recursive functions
- [26] - Huang et al, 1993: uses an heuristic approach with polynomial cutoff time, the percentages are shown are their own rated accuracy of the output.
- [9] - Chang et al, 1993: uses VIPER,a path extraction tool.
- [36] - Maurer, 1992: uses compiled circuit simulation.

As can be seen from the data in table 5.5,[6] the topological search algorithm is clearly superior to the path-enumeration techniques in many cases, and is slightly inferior to these techniques when the circuits contain few long false paths. However, the topological search algorithm is just as clearly inferior to the more recent non-enumerative techniques. For these techniques there is frequently an order of magnitude of difference in the required computation time.

The topological search algorithm is can be easily adapted for use in certain critical-component identification routines, as is shown in chapter 7, but if speed at identifying maximum sensitizable path lengths is desired then techniques such as those described by McGeer et al are clearly more efficient.

In chapter 6 a new non-enumerative analysis algorithm is described, which is designed to operate much more efficiently on the large circuits.

[6]Descriptions of the benchmark circuits themselves are provided in appendix 1.

## 5.7 Chapter summary

In this chapter the first of our randomized algorithms is developed and analyzed. This algorithm, topological search, uses a directed search of the input space of a circuit to find input vectors which sensitize critical paths in the circuit. The search is performed starting at each of a series of random input vectors, and from each vector the search attempts to find local maxima (with respect to associated circuit delay) in the input space. Associated delays are determined by simulating the input vector on the circuit and a hash table is maintained to minimize the amount of redundant searching performed.

Also in this chapter, the algorithm is analyzed with respect to both worst case behaviour and expected average case behaviour for "random" circuits of $m$ inputs. A simple model of the input space is used to estimate expected case behaviour, this model is based on the assumption that no two input vectors produce identical delays in the circuit.

The number of local maxima predicted using the model appears to be a reasonable estimate of the number of locally maximal input vectors, but not a reasonable estimate of the number of peaks in the input space of the benchmark circuits. One conclusion drawn from this is that delays in random circuits are much more sensitive to changes in input variables than is the case for "practical" circuits.

Using Harper's works on the assignment of values to the vertices of a hypercube we are able to draw the following conclusion: if every input vector has a unique associated delay, then *regardless of the circuit implemented* the average rank difference between adjacent vectors in the input space falls between $2^{m-1}$

and $\frac{2^m-1}{m}$. The value

$$\sum_{i=0}^{n-1} \binom{i}{\frac{i}{2}}$$

is derived as a lower bound on the required clock speed for any circuit under the model.

It can also be observed that any circuit with has a minimal worst-case delay difference between adjacent input vectors has the following property: there exists an $m$-bit Boolean vector $B$, such that for every pair of input vectors, $X_i, X_j$, to the circuit, the delay of $X_i$ is less than the delay of $X_j$ *iff* the weight of $X_i \oplus B$ is less than the weight of $X_j \oplus B$

Predicted behaviour for the topological search algorithm, both worst case and expected case, is very poor. However the observed behaviour across the benchmark set is much better than that predicted by the models. As the number of inputs to the circuits grow, estimates based on the models become increasingly inaccurate.

In comparing the algorithm to other existing algorithms, we find the topological search is frequently superior to techniques based on path-checking, though it is slightly inferior to these techniques for circuits which contain few long false paths. (Of course, for circuits with no long false paths the topologically longest path in the circuit is 100% accurate as an estimator and is obtainable in linear time in the number of lines in the circuit.)

Of note with respect to analysis of the benchmark circuits, in most published works the unit delay model is used. This model often drastically oversimplifies circuit behaviour and generally reduces the apparent number of long false paths in a circuit. The topological search method might compare more favourably with existing techniques if they were run using a more realistic delay model. The algorithm is comparable, in terms of CPU time requirements, to path-enumerative tech-

niques, but is significantly less efficient than recently published non-enumerative techniques.

# Chapter 6

# Estimation through an analysis circuit

The algorithms discussed in chapters 2 and 4 have some critical failings. All perform well in many cases, but have worst-case exponential run-times. (As must be the case for any exact algorithm given the assumption that $P \neq NP$.) This chapter contains the development and analysis of a more complex randomized algorithm to estimate the maximum delay in the circuit. This algorithm is based on the use of a timing analysis circuit, and provides both upper and lower bounds on the maximum delay in the circuit.

The algorithm is based on ideas originally suggested by McGeer [38] and Larrabee [31]. An analysis circuit, as introduced in chapter 4 and discussed in greater detail here, is created based on the original circuit under study. The analysis circuit is translated into a CNF formula in which the majority of the clauses consist of only two literals. In this chapter a new randomized algorithm is introduced and used to either find a satisfying solution to the formula or prove that the formula is not satisfiable.

Three other possible algorithms are proposed, but not developed, in the latter sections of chapter 4.

## 6.1 The analysis circuit, $C_A$

In this section, we assume we are given a combinational circuit, $C$, and must identify a time $T_c$, which represents the worst case stabilization time for $C$. That is, $T_c$ is the length of the longest sensitizable path in $C$. Assume $C$ takes as input an $m$-bit vector, $X =< x_1, x_2, ..., x_m >$. From $C$ a timing analysis circuit, $C_A$, is created. This circuit takes as input a time $T$ (encoded as a bit vector) and vector $X$. The output of $C_A$ is a *true* iff $C$ takes time *at least* $T$ to stabilize given input vector $X$. A method for creating $C_A$ was originally proposed in [39], and is described in the translation section of this chapter (6.2.2). The algorithm studied in this chapter utilizes $C_A$ as follows:

Given any time, $T$, one of two situations exist:

**either** There is some vector $X$ which satisfies $C_A$ given $T$, (which implies $T$ is a lower bound on $T_c$),

**or** $C_A$ is not satisfiable given $T$, (which implies $T$ is an upper bound on $T_c$).

Let $T_u, T_l$ describe the known upper and lower bounds on $T_c$. As long as each new choice of a $T$ value satisfies $T_u > T > T_l$ then by analyzing $C_A$ with respect to $T$ it is possible to improve one of the bounds on the length of the longest sensitizable path in the circuit. Our contributions in this matter are in the form of the selection of $T$ values and new methods used to either determine satisfying conditions for $C_A$ or prove $C_A$ is unsatisfiable.

The satisfiability of the analysis circuit is examined by translating the analysis circuit into an annotated CNF formula. It is shown in this chapter that, given bounded fan-in and fan-out, the number of clauses in the formula is linear in the size of $C$. As a by-product of creating $C_A$, preliminary upper and lower bounds are obtained for the length of the longest sensitizable path(s), in the form of the length of the topologically longest and shortest paths of the circuit.

### 6.1.1 Notation

A few more elements of notation must be introduced at this point for use in the rest of this chapter.

Given that $g_i$ is the output of some gate in circuit $C$, and $X =< x_1 \ldots x_m >$ is a Boolean input vector to $C$, then define $\chi^{g_i,t}$ to be that Boolean function which is *true* iff $g_i$ takes time at least $t$ to stabilize after $X$ is supplied as input to $C$. Thus if $F$ is an output of $C$, then $\chi^{F,T}$ is satisfiable iff $T \leq T_c$.

We remind the reader that the sum $(+, \sum)$ and product $(\cdot, \prod)$ notation throughout this section represent Boolean OR and AND respectively.

### 6.1.2 Translation rules:

The analysis circuit, $C_A$, contains all the logic of the original circuit, $C$, plus additional logic reflecting the delay characteristics of $C$. To obtain the delay logic for $C_A$, the following translation rules are applied to each logic gate of $C$. Given that gate $g_i$ has inputs $a_1, ..., a_k$, then the formulae for translating the basic Boolean logic gates of $C$ into new logic gates in $C_A$ are as follows:

$$NOR, OR\ gates : \chi^{g_i,t} = \sum_{i=1}^{k} \left[ \chi^{a_i,t-1} \cdot \prod_{j=1, j\neq i}^{k} \left( \chi^{a_j,t-1} + \overline{a_j} \right) \right]$$

$$NAND, AND\ gates : \chi^{g_i,t} = \sum_{i=1}^{k} \left[ \chi^{a_i,t-1} \cdot \prod_{j=1, j\neq i}^{k} \left( \chi^{a_j,t-1} + a_j \right) \right]$$

$$XNOR, XOR\ gates : \chi^{g_i,t} = \sum_{i=1}^{k} \chi^{g_i,t-1}$$

$$NOT\ gates : \chi^{g_i,t} = \chi^{a,t-1}.$$

When the translation process is complete we find the inputs to $C_A$ are: $x_1, ..., x_m$, plus a number of inputs of the form $\chi^{x_i,T-r_j}$, where $\chi^{x_i,T-r_j} = true$ iff $T - r_j \leq 0$, i.e. $T \leq r_j$. An intuitive explanation of the translation process, using a three input OR gate as an example, is provided in section 6.1.3.

### 6.1.3 Example

The delay logic in $C_A$ is created on a gate-by-gate basis from $C$ as shown in the following example. Assuming unit delay per gate, to simplify the example. Figure 6.1 shows the translation of a 3-input OR gate into a set of gates for an analysis circuit. Consider a 3-input OR gate, $g_i$, which is present in circuit $C$. Then $g_i$ takes time at least $t$ to stabilize after an input vector $X$ is applied to the primary inputs of $C$ iff at least one of the following holds:

**case 1** $a_1$ takes time at least $t-1$ to stabilize *and* is the controlling input to gate $g_i$,

**case 2** $a_2$ takes time at least $t-1$ to stabilize *and* is the controlling input to gate $g_i$,

**case 3** $a_3$ takes time at least $t-1$ to stabilize *and* is the controlling input to gate $g_i$.

Case 1, is true if:

**first:** $a_1$ takes time at least $t-1$ to stabilize, i.e. $\chi^{a_1,t-1} = true$.

**and** *either* $a_2$ is a non-controlling input to $g_i$ *or* takes time at least $t-1$ to stabilize, i.e. $\overline{a_2}$ or $\chi^{a_2,t-1} = true$.

**and** *either* $a_3$ is a non-controlling input *or* takes time at least $t-1$ to stabilize, i.e. $\overline{a_3}$ or $\chi^{a_2,t-1} = true$.

A controlling input to an OR gate is an input which carries logical *true*, since the arrival of the first *true* input to an OR gate dictates that the output of the OR gate is *true*, regardless of the values carried by the other gate inputs. If no inputs to the OR gate carry logical *true*, then the arrival of the last signal to the gate dictates the time at which the gate output stabilizes at logical *false*.

The arguments for showing the second or third cases ($a_2$ controlling, $a_3$ controlling) to be true are symmetric with case 1.

Similar arguments hold for the other Boolean logic gates, with the following differences:

- *true* is the non-controlling value for AND/NAND gates,
- there are no controlling values for XOR/XNOR gates, the arrival of the last gate input determines the gate output stabilization time,
- there are no side inputs to NOT gates.

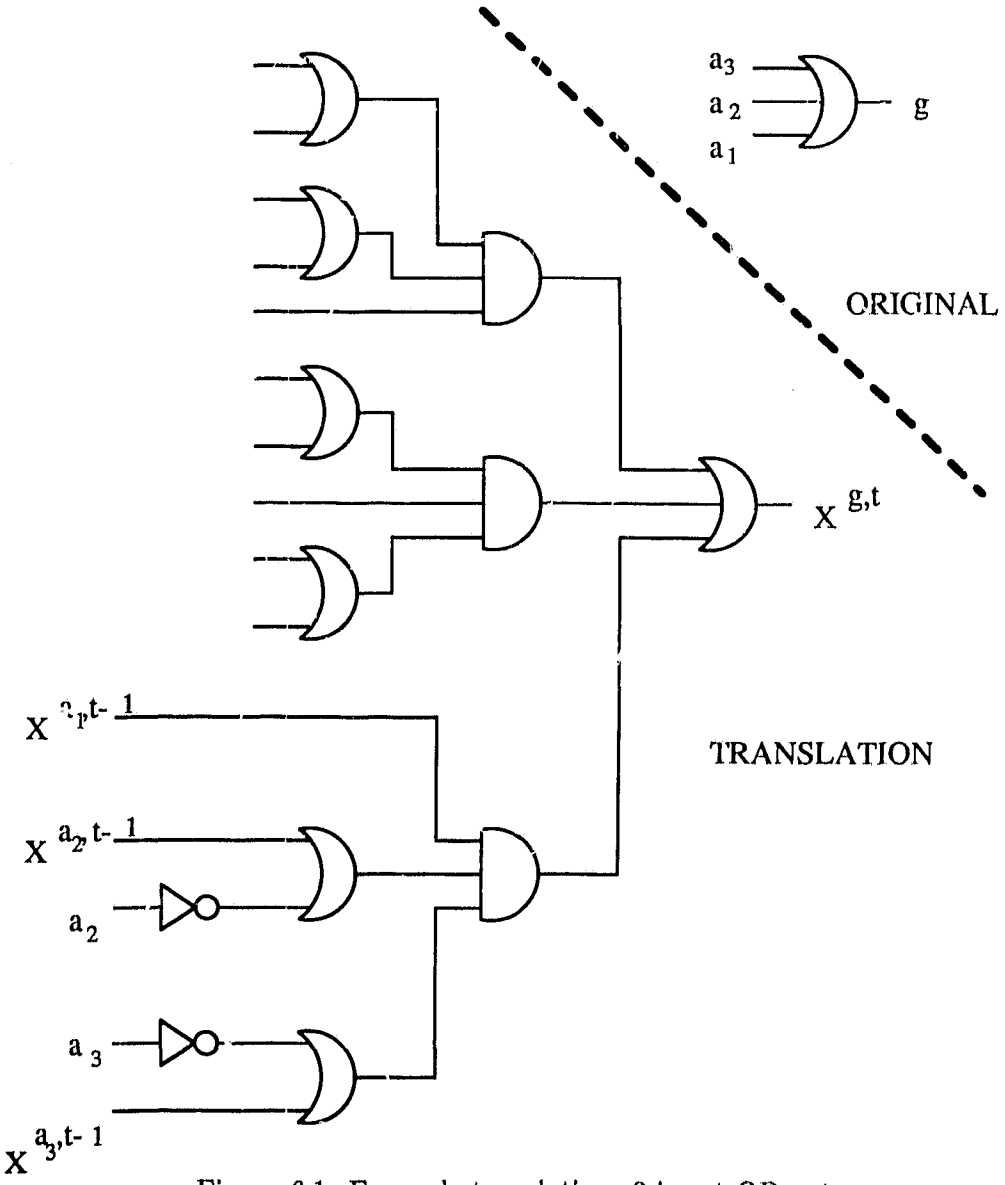


Figure 6.1: Example translation: 3-input OR gate
$X^{a_i,t-1}$ is *true* iff $a_i$ takes time at least $t-1$ to stabilize,
thus $X^{g,t}$ is *true* iff the output of $g$ takes time at least $t$ to stabilize.

### 6.1.4 Complexity of the analysis circuit

In this section the size of the analysis circuit is related to the size of the original circuit. $C_A$ includes all the original circuitry of $C$ (to determine the logic values of the side inputs) plus the following extra gates: each (N)AND/(N)OR gate in $C$ (with fan-in bounded by $k$) translates into $k$ AND gates plus one OR gate in $C_A$, each with fan-in equal to that of the original gate, plus an additional $k(k-1)$ 2-input OR gates. Thus each gate translates into at most $k^2+1$ gates, all still with fan-in $\leq k$.

If $k$ is some constant, then the size of $C_A$ is linear in the size of $C$, measuring size by the number of gates. The number of lines in a circuit of bounded fanin is apparently linear in the number of gates, therefore $C_A$ is also apparently linear in the size of $C$, basing size on the counting of lines.

#### The impact of delay models on complexity

Unfortunately, the "appearances" are not correct. Any given gate, $g_i$, might fan out to several different gates, each of which can provide a distinct $t$ value for use in the $\chi^{g_i,t}$ formula. An upper bound on the number of $\chi$ values associated with a gate, $g_i$, can be determined as follows: let $S$ be the set of paths from $g_i$ to the circuit primary outputs. Let $P$ be a partitioning of $S$ into equivalence classes, with the equivalence relation based on path length, paths of equal length are in the same equivalence class. The number of equivalence classes in $S$ gives an upper bound on the number of $\chi$ values be associated with gate $g_i$. Thus, the total number of $\chi$ values used in the circuit can potentially grow exponentially with the depth of the circuit. This number is bounded by the number of different pathlengths present in the circuit, but can still clearly be extremely large.

The approach adopted in the implementation discussed here is to use annotated clauses for any clause containing a $\chi$ value. The annotation is simply a list of all the different $t$ values for which $\chi^{g_i,t}$ must be calculated. Thus the number of annotated clauses, and the number of literals contained in those classes is linear in the size of the original circuit, but each clause may have an exponential number of annotated $t$ values.

## 6.2 Delay determination using $C_A$ and 2CNF

The problem of determining the maximum $T$ values for the analysis circuit is satisfiable is addressed in this section. In [31], Larrabee points out that a combinational circuit can be translated into a CNF formula in which the number of clauses is linear in the number of gates in the circuit. Furthermore, for circuits of simple logic gates at least two-thirds of the clauses contain only two literals. Larrabee also suggests that satisfying assignments for the circuit may be found by solving the 2CNF part of the problem (which can be done in polynomial time in the number of clauses and literals, using an implication graph, $I$) then extending that solution to satisfy the entire formula.

The problem with this technique is that there can be an exponential number of solutions to the 2CNF formula, yet only a few satisfying assignments to the overall problem. The problem then, is to search quickly through a subset of the 2CNF solutions to find one which may be extended, or else to prove that the overall formula is not satisfiable.

There is an interesting aside to this process involving the Lovasz local lemma. If all gates in the circuit have the same fan-in, $k$, then all the non-binary clauses in the CNF formula have exactly $k+1$ literals. This allows the use, at least

theoretically, of a pretest to determine the satisfiability of the formula.

### 6.2.1 Pretest:

The Lovasz local lemma states that, given a CNF formula in which each clause contains $k$ literals, then if each variable appears in at most $2^{\frac{k}{100}}$ clauses then *there is a satisfying assignment.* Furthermore, if there is a satisfying assignment then Beck [2] provides a randomized algorithm to find it. Note that for "practical" applications this would mean each variable appears in only a single clause (each variable can appear in at most $1.007^k$ clauses, which is $< 2$ for $k < 100$).

**Beck's algorithm if pretest succeeds:**

Here we provide Beck's algorithm for finding a satisfying assignment to a CNF formula, given that the Lovasz pretest has been used to determine such an assignment must exist.

If the pretest fails then move on to the next stage, otherwise proceed sequentially through the variables, fixing each equiprobably to *false* or *true.* After each setting, check to determine if, for any clause, the clause is not satisfied yet but $\frac{k}{2}$ of it's literals have been fixed. The remaining unset variables of this clause are removed from the sequential setting process and deferred to stage 2. Once all the variables which have not been deferred are set at *false* or *true*, begin stage 2. Stage 2: partition the surviving clauses into sets $S_1, S_2, ...$ such that each pair $S_i, S_j$ are disjoint in the variables they contain. Exhaustively examine the individual sets for satisfying assignments.

### 6.2.2 Our algorithm: an overview

An outline of the algorithm is shown here. More thorough descriptions of each stage are given in sections 6.3 through 6.6.

## The Algorithm

1. **Translate** $C$, the original circuit, into an analysis circuit, $C_A$, and set values $T_u, T_l$ to the lengths (delays) of the longest and shortest topological paths respectively.

2. **Translate** $C_A$ into a CNF formula, $f$. $f$ is composed of both a 2CNF subformula, and a kCNF subformula.

3. **Create an implication graph**, $G$, from the 2CNF portion of $f$.

4. **Further restrict** $T_l$ as follows:

   - pick a "small" set of random vectors, determine the longest path sensitized by any vector in this set, and use this value to set $T_l$,

5. **Main routine:** There are two different techniques which may be applied at this point to determine $T_c$: a non-constructive technique, and a constructive technique. Both techniques operate in part by estimating the number of non-satisfying solutions to a CNF formula.

   **Non-constructive technique:** in polynomial time, with probability at least $1-\alpha$, this technique finds the maximum $T$ value for which the number of satisfying assignments to the remainder of the CNF formula is at least $\frac{2^m}{p(m)}$, for some polynomial $p$. Furthermore, if the set of satisfying assignments forms a single cube on the input space, then the cube can be exactly identified in polynomial time, again with probability at least $1-\alpha$.

**Constructive technique:** based on the assumption that the number of assignments which satisfy the formula may be less than $\frac{2^m}{p(m)}$, here a binary search is used to find the maximum satisfiable $T$ value. A probabilistic search technique is used to determine satisfying assignments given a specific $T$ value. The process works as follows: repeatedly select $T$ values in the range from $T_l$ to $T_u$. Here we choose $T = \frac{1}{2}(T_u + T_l)$ (binary search) until, for some predefined $0 \leq \epsilon \leq 1$

$$T_l \geq \frac{T_u}{1+\epsilon}$$

For each chosen $T$ value perform the following:

- If there is a large number of inputs, then apply Lovasz's pretest, and if that indicates a satisfying assignment exists then apply Beck's algorithm to determine a satisfying assignment, $X$. Since the delay associated with vector $X$ on circuit $C$ may in fact be considerably larger than $T$, apply $X$ to $C$ to determine a new $T_l$.
- Copy and simplify $G$ and $f$ based on $T$, giving $G', f'$ That is, set the $\chi$ values implied by $T$, and simplify the CNF formulas given these settings.
- Test to see if there is an implied contradiction in $G'$, and if so then $T_u = T$. Otherwise, attempt to determine a new $T_l$ by finding a satisfying assignment to $G', f'$.
- Select and test a polynomial number of solutions to $G'$ as solutions to $f'$

  To obtain and test each $G'$ solution perform the following steps:

  - For each literal $x_i$ or $\overline{x_i}$, estimate the number of non-satisfying solutions to $f'$ given that variable setting. This is done using a randomized polynomial time algorithm which is the dual of the

algorithm used by Karp, Luby, and Madras [28] to estimate the number of satisfying assignments to a DNF formula.

- Order the variables by $|p_{x_i} - p_{\overline{x_i}}|$, the probabilities that setting $x_i$ to *true* or *false* results in $f'$ being unsatisfiable.
- Repeat the following steps until either variables $x_1, ..., x_m$ have each been assigned values (if this point is reached then a solution has been found for $G'$ which also extends to $f'$) or until a contradiction is detected (the solution being developed for $G'$ can not be a solution for $f'$):
  * select the next unassigned variable under the ordering
  * set the chosen variable to have logic value *true* with probability $\frac{p_{x_i}}{p_{x_i}+p_{\overline{x_i}}}$, otherwise set the variable value to *false*,
  * simplify $G', f'$ given the variable setting so far,
  * check for a contradiction in $G'$ or a non-satisfying assignment in $f'$

- If a solution for $f'$ was found then verify it against $f$:
  if the solution fails to satisfy $f$ then $T_u = T$, otherwise take the satisfying solution, $X$, and determine the maximum $T$ value for which $C_A$ is satisfiable given $X$, this is the new $T_l$ value. The $T$ value in this case can be easily determined in polynomial time in the size of $C_A$ by performing a binary search on $T_u, T_l$ given $X$. (It is much easier to determine the maximum satisfying $T$ given $X$ than it is to determine a satisfying $X$ given $T$.)

## 6.3 Algorithm pre-processing steps:

In this section, each of the four pre-processing stages of the algorithm is considered in detail.

**Create $C_A$:** The $C_A$ circuit is created from $C$. This corresponds to step 1 in the algorithm as presented in section 6.2.

**Create the CNF formula:** Step 2 in the algorithm is to translate $C_A$ into a CNF formula as described in section 6.3.1 and 6.3.2 below. The formula is considered in two parts, the 2CNF section, in which each clause contains two literals, and the kCNF section, in which each clause contains from 3 to $k$ literals.

**Create $G$:** From the 2CNF formula we obtain an implication graph, $G$. This graph records the implications between literals in the 2CNF formula. This corresponds to step 3 of the algorithm, and is discussed in sections 6.3.3 and 6.3.4 below.

**Initial bounds:** Step 4 of the algorithm involves preprocessing to obtain upper and lower bounds on the maximum true delay of $C$ (i.e. $T_u, T_l$). The proposed bounds are determined as follows: use the length of the topologically-longest path in $C$ as the initial upper bound, and simulate a (small) subset of random input vectors on $C$ to determine an initial lower bound.

### 6.3.1 Translating $C_A$ to CNF form

This section contains the rules for translating simple gates into a CNF expression. The reader is reminded that, in the actual implementation, the translation from

$C$ into $C_A$ and from $C_A$ into a CNF formula is combined into a single step, as is discussed in section 6.3.2 below.

$$AND :: g_i = \prod_{j=1}^{k} a_j$$

becomes $(a_1 + \overline{g_i})(a_2 + \overline{g_i}) \cdots (a_k + \overline{g_i})(\overline{a_1} + \overline{a_2} + \cdots + g_i)$.

$$NAND :: g_i = \overline{\prod_{j=1}^{k} a_j}$$

becomes $(a_1 + g_i)(a_2 + g_i) \cdots (a_k + \overline{g_i})(\overline{a_1} + \overline{a_2} + \cdots + \overline{g_i})$.

$$OR :: g_i = \sum_{j=1}^{k} a_j$$

becomes $(\overline{a_1} + g_i)(\overline{a_2} + g_i) \cdots (\overline{a_k} + g_i)(a_1 + a_2 + \cdots + a_k + \overline{g_i})$.

$$NOR :: g_i = \overline{\sum_{j=1}^{k} a_j}$$

becomes $(\overline{a_1} + \overline{g_i})(\overline{a_2} + \overline{g_i}) \cdots (\overline{a_k} + \overline{g_i})(a_1 + a_2 + \cdots + a_k + g_i)$.

$$NOT :: g_i = \overline{a}$$

becomes $(g_i + a)(\overline{g_i} + \overline{a})$.

$$FANOUT :: g_i = a$$

becomes $(g_i + \overline{a})(\overline{g_i} + a)$.

$$XOR2 :: g_i = \overline{a_1} \cdot a_2 + a_1 \cdot \overline{a_2}$$

becomes $(a_1 + a_2 + \overline{g_i})(\overline{a_1} + \overline{a_2} + \overline{g_i})(\overline{a_1} + a_2 + g_i)(a_1 + \overline{a_2} + g_i)$.

$$XNOR2 :: g_i = a_1 \cdot a_2 + \overline{a_1} \cdot \overline{a_2}$$

becomes $(a_1 + a_2 + g_i)(\overline{a_1} + \overline{a_2} + g_i)(\overline{a_1} + a_2 + \overline{g_i})(a_1 + \overline{a_2} + \overline{g_i})$.

### 6.3.2 Implementation notes:

In practice, the implementation deals with only NAND, NOR gates (as these are the gates used in the implementations considered for the listed benchmark circuits). To reduce the complexity of the algorithm, the two translation steps ($C$ to $C_A$, $C_A$ to CNF) are combined, i.e. the original circuit is translated directly into a CNF formula.

The CNF formula produced for a $k$-input NAND gate is as follows:

$$CNF = (f' + \sum_{i=1}^{k} a_i')(L' + X + \sum_{i=1}^{k} l_i')(f + X + \sum_{i=1}^{k} F_i')(f' + L)$$
$$\prod_{i=1}^{k} [(a_i + f)(l_i + X')(L + F_i')(a_i' + l_i)(l_i + R_i')(a_i + l_i' + R_i)]$$

Similarly, the CNF formula produced for a $k$-input NOR gate is as follows:

$$CNF = (f + \sum_{i=1}^{k} a_i)(L' + X + \sum_{i=1}^{k} l_i')(f' + X' + \sum_{i=1}^{k} R_i)(f + L)$$
$$\prod_{i=1}^{k} [(a_i' + f')(l_i + X')(L + R_i')(a_i + l_i)(l_i + F_i')(a_i' + l_i' + F_i)]$$

Where the notation used is:

- The inputs to the gate are labeled $a_i, i \in 1 \ldots k$.
- The output of the gate is labeled $f$.
- Temporary variables $l_1, \ldots, l_k$ and $L$ may be introduced in the translation process.
- $X$ represents $\chi^{f,t}$ ($\chi$ values are explained in section 6.1.1)
- $R_i$ represents $\chi^{a_i,t-rise(a_i)}$, where $rise(a_i)$ represents the rise delay in the gate due to a falling logic value on input $a_i$.
- $F_i$ represents $\chi^{a_i,t-fall(a_i)}$, where $fall(a_i)$ represents the fall delay in the gate due to a rising logic value on input $a_i$.

### 6.3.3 Solving 2CNF with implication graphs

In this section the relationship between a 2CNF formula and an implication graph is reviewed. Given a set of binary clauses from a 2CNF formula, an implication graph is established as follows: for each variable in the formula create two nodes, one representing the true form of the variable and one representing the complemented form. For each clause of the form $(A + B)$, if $A = false$ then for the clause to be satisfied it must be the case that $B = true$. So for each clause add two directed edges, $(\overline{B}, A)$, and $(\overline{A}, B)$. Observe that the presence of a cycle in the graph means the truth of any literal in the cycle implies the truth of every literal in the cycle, and hence all the literals in the cycle may be treated as a single, new, literal. This, in turn, means that all strongly-connected components in the graph may be collapsed until only a directed acyclic graph remains. The collapsing of strongly connected components is carried out as outlined in the algorithm provided in appendix 4.

### 6.3.4 Simplifying the implication graph and $k$CNF formula

Any setting of a variable carries implications for the values of literals in both the implication graph and the $k$CNF formula. This simplification process is carried out each time a new partial assignment is made to $T$ and/or $X$.

For example, if a variable is set to *true* then any $k$CNF clauses containing the "true" literal for that variable are satisfied and may be discarded, and any kCNF clauses containing the "false" literal for that variable are reduced in size by a single literal - possibly reducing them to binary clauses in which case they may be added to the implication graph. Similarly, the values of many literals in the implication graph are implied, simplifying that graph, and allowing further simplification of the $k$CNF formula, et cetera.

## 6.4 Main Routine:

The main routine of the algorithm is summarized in step 5 of section 6.2. In this section the two approaches are considered for use within the main routine to estimate $T_c$. One is a non-constructive approach, and the other a constructive approach. Also described here is the technique for estimating the number of non-satisfying assignments to a CNF formula, as this estimation technique is critical to both the constructive and non-constructive methods. In sections 6.5 and 6.6 respectively, the non-constructive and constructive solution techniques are discussed in detail. First, the problem of estimating the number of non-satisfying assignments to a CNF formula is considered.

### 6.4.1 Estimating the number of non-satisfying assignments to $k$CNF

An important problem with respect to both the constructive and non-constructive estimation processes in the algorithm is the problem of accurately estimating the number of non-satisfying assignments to the $k$CNF formula given certain assignments to subsets of the variables.

Karp, Luby, and Madras [28] developed a randomized polynomial time algorithm for estimating the number of solutions to a DNF formula. Estimating the number of non-satisfying assignments to a CNF formula is simply the dual of that problem. Here an intuitive description is given for the estimation process, detailed proofs of the algorithm correctness and running time are provided in [28] (for the DNF case).

Observe that if the clauses in the CNF formula are ordered (any arbitrary ordering) then while each non-satisfying assignment may fail to satisfy many clauses in the formula, it has a unique first clause (under the ordering) which it fails to satisfy. It is easy to determine the number of assignments which fail to satisfy a particular clause - if the formula is of $m$ variables and the clause contains $k$ literals then the number of non-satisfying assignments is simple $2^{m-k}$. The more difficult part of the problem is to estimate how much overlap there is between the clauses.

By randomly sampling from the (overlapping) sets of assignments which fail to satisfy at least one of the clauses, but counting an assignment iff the clause from which it is selected is the first clause in which it appears under the ordering, then we can estimate the degree of overlap between the clauses.

For any given clause, $\varsigma_i$:

- Set each variable appearing in the clause to the value which does *not* satisfy the clause. That is, the complement of the literal appearing in the clause.
- Randomly assign values *true* or *false* to the remaining variables in the formula.
- If the assignment fails to satisfy any clause $\varsigma_j$ (where $j < i$ under the clause ordering) then the assignment is not counted, otherwise the assignment is counted.
- Repeatedly taking such random assignments allows us to estimate the number of non-satisfying assignments the clause contributes to the formula under the ordering. If $S$ samples are taken, and $A$ assignments are "counted", and $\mid \varsigma_i \mid$ represents the number of non-satisfying assignments to clause $\varsigma_i$, then the estimated number of non-satisfying assignments to the formula is $\frac{A}{S}\sum(\mid \varsigma_i \mid)$.

If there are $U$ non-satisfying solutions, then taking a polynomial number of sample assignments to the variables (each of which may be evaluated in polynomial time) allows us to obtain an estimator, $\nu$, for $U$ such that, with probability at least $1-\alpha$, $\frac{U}{1+\epsilon} \leq \nu \leq U \cdot (1+\epsilon)$.

It is shown in [41] that, using such techniques, the number of samples which must be taken, given $m$ variables, is

$$\frac{4m}{\epsilon^2} ln(\frac{2}{\alpha})$$

In the following two sections, this estimation technique is applied as part of the process of determining the maximum $T$ value for which the CNF formula is satisfiable.

## 6.5 Main routine: non-constructive approach:

In this approach, an approximation is made for $T_c$ without ever considering a satisfying assignment to the CNF formula. In polynomial time, with probability at least $1-\alpha$, this technique finds an approximator, $T_e$, for the desired value $T_c$ such that $T_p \leq T_e \leq T_c$. Where $T_p$ is the maximum $T$ value for which the number of satisfying assignments to the remainder of the CNF formula is at least $\frac{2^m}{p(m)}$, for some polynomial $p$. Furthermore, if the set of satisfying assignments for $T$ values in the range $T_e \ldots T_c$ forms a single cube on the input space, then the cube can be exactly identified in polynomial time, again with probability at least $1-\alpha$.

A binary search is performed on the $T$ values in the range $T_l, T_u$ to find $T_e$. As there are at most $2^m$ values of $T$ in that range, the binary search considers at most $m$ values of $T$. Therefore, if each $T$ value is considered in polynomial time then the entire search is completed in polynomial time. The problem which must be considered is how to determine in polynomial time, with probability at least $1-\alpha$, if there are any satisfying assignments to the formula for $T$ values greater than or equal to the one under consideration.

It is shown in [41] that the number of satisfying solutions to a formula in disjunctive normal form can be determined in polynomial time using the dual of the estimation technique discussed in section 6.4 provided the number of samples taken is at least

$$\left(\frac{4m}{\epsilon^2}\right) ln \left(\frac{2}{\alpha}\right)$$

The problem we encounter in estimation accuracy is that, for a set of satisfying assignments of size $S$, $\epsilon$ must be small enough to distinguish between $\frac{2^m-S}{2^m}$ and

1. If we rewrite this formula to obtain $\epsilon$, the result is:

$$\epsilon = \sqrt{\frac{4m}{N} ln \frac{2}{\alpha}}$$

Each sample requires an amount of time linear in the size of the CNF formula, therefore the estimation process can be completed in polynomial time (as desired) if the number of samples taken is polynomial. That is, if $N = p_1(m)$ for some polynomial $p_1$.

Given this specification for $N$, and the specification that $\epsilon$ is bounded by the size of the set of satisfying assignments as discussed above, we obtain the following:

$$S \geq 2^{m+1} \sqrt{\frac{4m}{p_1(m)} ln \frac{2}{\alpha}}$$

From this, it follows that if the size of $S$ is of order $\frac{2^m}{p(m)}$ for some polynomial $p$, then the estimation process can be used to determine whether or not $S > 0$ with probability at least $1 - \alpha$.

### 6.5.1 Special case: single critical cubes

In this section, the process for estimating the number of non-satisfying solutions is analyzed, given the simplifying assumption that there is only a single cube of input vectors which satisfies the formula. In such a case, the estimated number of solutions can be used to determine the exact assignment of logic values necessary to satisfy the formula, and also to identify which variables in the formula are "don't-cares" with respect to satisfying the formula.

The single-satisfying-cube assumption can reasonably be made at two levels:

1. As is described in the benchmark analysis of chapter 5, for many benchmark circuits there *is* only a single cube of input vectors which sensitizes a critical

path, and hence only a single cube which satisfies the CNF formula given $T_c$.

2. During the search for a satisfying assignment for the formula, there *must* come a point at which, out of the variables whose values have yet to be fixed, only a single cube of assignments to those variables satisfies the entire formula.

Suppose $b_1, b_2, \ldots, b_\mu$ is the set of variables used in the CNF formula, and the satisfying cube for the formula is dependent on $\mu - \rho$ of these variables, i.e. $\rho$ of the variables are "don't-cares".

Then, for each $b_i$ in the set of $\mu - \rho$ variables upon which the satisfying cube is dependent, assuming $\rho \geq 1$, the following holds:

- if $b_i$ is set to the satisfying logic value then there are $2^\rho$ satisfying assignments to the formula out of the $2^{\mu-1}$ possible assignments of logic values to the other $\mu - 1$ variables, and
- if $b_i$ is set to the non-satisfying logic value, then there are 0 satisfying assignments to the formula, out of the $2^{\mu-1}$ possible assignments.

For each $b_i$ in the set of $\rho$ don't care variables, on the other hand, the number of satisfying solutions, regardless of the logic setting of $b_i$, is $2^{\rho-1}$, again out of $2^{\mu-1}$ possibilities.

Thus, if the estimation technique for the number of satisfying assignments is of sufficient accuracy, and if only a single satisfying cube exists for the remaining unassigned variables in the formula, then we can exactly determine a satisfying solution by running the estimation algorithm for each of the two logic values on

each of the unassigned variables. If the estimated number of satisfying solutions based on setting $b_i$ to *true* is the same as the estimated number of satisfying solutions based on setting $b_i$ to *false* then $b_i$ is a non-controlling input in the cube. Otherwise, set $b_i$ to the logic value for which the estimated number of satisfying solutions is non-zero.

The question which must now be answered is:

> under what set of conditions is this estimation technique accurate enough to reliably distinguish the three enumeration levels, $0, 2^{\rho-1}, 2^{\rho}$, in polynomial time?

To answer this question, consider the number of samples required, and the estimation accuracy levels which may be attained using that number of samples given a set of satisfying assignments of size $S$. Let $S = 2^k$ be the actual number of satisfying assignments to the formula, i.e. there are $k$ don't-care variables in the satisfying cube. The number of non-satisfying assignments is $2^m - S$. Let $\epsilon$ be the desired accuracy of estimation, $0 < \epsilon < 1$. As discussed in the preceding section, the sampling technique provides an estimate which is within the range $\frac{2^m - S}{1+\epsilon} \ldots (1+\epsilon)(2^m - S)$ with probability at least $1-\delta$, where $0 < \delta < 1$, provided the number of samples taken is at least

$$\left(\frac{4m}{\epsilon^2}\right) ln \left(\frac{2}{\delta}\right)$$

Given a polynomial number of samples, $N = p_1(m)$, the $\epsilon$ value obtained is:

$$\epsilon = \sqrt{\frac{4(m-1)}{p_1(m-1))} ln \frac{2}{\delta}}$$

However, to be able to completely identify the satisfying cube after a single round

of estimations[1], we require that $\epsilon$ be sufficiently small to distinguish between the three values $2^{m-1}, 2^{m-1} - \frac{S}{2}$, and $2^{m-1} - S$. That is, $\epsilon < \frac{S}{2^{m+1}}$.

Furthermore, if the accuracy desired across all estimations is $\alpha$, it is required that the $\delta$ used for an individual sample set be "sufficiently small". If $\alpha$ accuracy is desired for the round of $2m$ sets of $N$ estimates, then the accuracy required for a single set of $N$ estimates is $\delta = \sqrt[2m]{\alpha}$.

To correctly identify the satisfying cube using only a polynomial number of samples in each estimation stage, the relationship between $\alpha, N = p(m), m$, and $k$, must therefore be

$$\frac{S}{2^{m+1}} \geq \sqrt{\frac{4(m-1)}{p_1(m-1)} ln \frac{2}{\sqrt[2m]{\alpha}}}$$

Thus, to achieve the desired accuracy levels, it must be the case that

$$S \geq \frac{2^{m-1}}{p(m-1)}$$

for some polynomial $p$.

If this condition is met, then the entire main routine (step 5 of the algorithm) can be replaced by a single round of $2m$ sets of estimates, each requiring $N$ samples. With a probability of $1 - \alpha$ this procers will correctly identify the exact satisfying cube.

## 6.6 Main routine: constructive approach:

In this section a constructive approach is provided for finding both the maximum $T$ value for which the CNF formula is satisfiable *and* the satisfying assignment

[1]That is, taking $N$ samples for each of the $2m$ settings of the input variables.

given that $T$ value. This technique is based on the assumption that the number of assignments which satisfy the formula may be less than $\frac{2^m}{p(m)}$ for polynomial $p$.

The basic process is to repeatedly select and analyze new $T$ values (step 5 of the algorithm) until either $T_l \geq \frac{T_u}{1+\epsilon}$ for some pre-determined bound $\epsilon$, or until a polynomial number of $T$ values have been considered.

The $T$ value selected must satisfy $T_l < T < T_u$. This selection implies the logic values for all inputs of the form $\chi^{x_i, T-r}$, (which equal *true* iff $T \leq r$). Make and simplify copies of both the implication graph and the $k$CNF formula for the current iteration (described below). The strategy used here to select the new $T$ value is essentially a binary search, $T = \frac{1}{2}(T_l - T_u)$. Given the new $T$ value, analyze the satisfiability of the revised CNF formula as described in sections 6.6.1 through 6.6.3 below.

### 6.6.1 Key loop within constructive approach: finding and testing satisfying assignments:

When developing solutions for the 2CNF formula, on each pass through the main loop one of three things happens:

1. A satisfying assignment, $X$, is found for the combined CNF formula. In this case $T_l = T$.

2. The formula is not satisfiable given the assignment to $T$, in which case $T_u = T$.

3. The current iteration is proceeding too slowly (too many assignments have been attempted without establishing a new bound). In this case, make no

assignment to $T_l$ or $T_u$, and proceed to the next iteration of the main loop to consider a different $T$ value.

Quickly finding a satisfying assignment (1) or a contradiction (2) is the key to this algorithm, and is carried out as follows.

### 6.6.2 Test 2CNF for contradictions:

As soon as the simplified implication graph is obtained (the simplification process is described below) the 2CNF graph is checked for any chain of implications of the form $l_1 \rightarrow l_2 \rightarrow \cdots \rightarrow \overline{l_1}$. This implies a contradiction, which in turn implies the formula is not satisfiable for this $T$ value, set $T_u = T$, and proceed to the next iteration of the main loop.

### 6.6.3 Test for satisfying 2CNF solutions:

If a contradiction is not found, then try up to a polynomial number of solutions to the 2CNF formula. If no satisfying solution to the general CNF formula is found amongst these solutions then terminate the analysis of the current $T$ value (no new bounds are set on $T_u, T_l$) and try a new $T$ value.

The 2CNF solutions to be attempted for a given $T$ value are selected as follows:

**Repeat until variables $x_1, ..., x_m$ are assigned *false* or *true* :**

**Order the variables:** For every input $x_1, ..., x_m$, estimate the number of *non-satisfying* assignments to the kCNF formula, first given that $x_i = false$, and second given that $x_i = true$. This can be done efficiently (polynomial time with certain restrictions), as described in section 6.4.1. If $p_{x_i}$ is the

probability of a non-satisfying assignment to kCNF given $x_i = true$, and $p_{\overline{x_i}}$ is the probability of a non-satisfying assignment to kCNF given $x_i = false$, then order the input variables by $|p_{x_i} - p_{\overline{x_i}}|$.

**Select next variable:** Select the next $x_i$ in the ordering amongst unassigned variables. Then, with probability $\frac{p_{x_i}}{p_{x_i}+p_{\overline{x_i}}}$ set the value of $x_i$ to *true*, otherwise set it to *false*. Simplify the implication graph and kCNF formula, and if no contradiction is implied proceed to select the next variable.

We make the following observations about the nature of $p_{x_i}, p_{\overline{x_i}}$:

- If variable $x_i$ is a don't care variable with respect to the set of satisfying assignments, then $p_{x_i} = p_{\overline{x_i}}$.
- If the formula is not satisfiable given $x_i$, then $p_{x_i} = 0$.
- If the formula is not satisfiable given $\overline{x_i}$, then $p_{\overline{x_i}} = 0$.

Thus, to within the accuracy limits of our estimator, the metric $|p_{x_i} - p_{\overline{x_i}}|$ provides a reasonable estimator for cases where the variable is a don't-care, and cases where the variable value must be fixed at one specific logic level. The case where this metric is less reliable is where the formula is satisfiable whether $x_i$ is in true or complemented form, but the number of satisfying assignments is different in the two cases.

Once all input variables have been assigned values, and no contradiction has been detected, we proceed to verify that the assignment satisfies the original CNF formula.

**Verification:**

If all input variables have been assigned values and have been determined to satisfy the simplified 2CNF/kCNF formulas then *check the solution against the original formula.* If the assignment satisfies the overall formula then one could set $T_l = T$ at this point, but if the delay of $C$ given $X$ is actually greater than $T$ then a better $T_l$ can be quickly found as follows: given that $C_A$ is known to be satisfiable given $X$ and $T$, and is not satisfiable for any input given a time value $> T_u$, perform a binary search on the range $T \ldots T_u$ to find the maximum time value which satisfies the CNF formula in conjunction with $X$. This time value is the new $T_l$. If the assignment does not satisfy the formula then a contradiction has taken place and the formula is not satisfiable, set $T_u = T$.

Finally, proceed with the next iteration of the main loop.

## 6.7 Complexity

In this section the overall running time of the algorithm is examined.

The translation from $C$ to $C_A$ is linear in the number of annotated clauses and the number of literals within those clauses, as described in section 6.3.1. The translation is not necessarily linear in the number of annotated $t$ values however. The translation from $C_A$ to the CNF formula is linear in the size of $C_A$, one binary clause is created per edge in $C_A$, plus one kCNF term per gate in $C_A$. For the two approaches, constructive and non-constructive, to estimating $T_c$ the complexity results are summarized in the following two paragraphs.

It is shown in section 6.5 that the non-constructive approach provides, with

probability at least $1-\alpha$ and in time polynomial in the size of $C_A$, an approximation in the range $\frac{T'}{1+\epsilon}, T'(1+\epsilon)$ for any prespecified $\epsilon$ and $\alpha$ the largest $T'$ value for which the number of satisfying assignments to the CNF formula for values greater than or equal to $T'$ is at least $\frac{2^m}{p(m)}$ for some polynomial $p$. If $\frac{2^m}{p(m)}$ vectors require $T_c$ stabilization time, this means $T' = T_c$. Furthermore, if the set of satisfying assignments forms a single cube on the input space then the cube can be exactly identified in polynomial time, again with probability at least $1-\alpha$.

For the constructive approach to determining both $T_c$ and a satisfying assignment for the CNF formula given $T_c$, we make the following observations. For each $T$, a polynomial number of solutions to the 2CNF formula are selected and analyzed to determine if they satisfy the $k$CNF formula. There are at most a linear number of $T$ values to be considered (the $\chi^{x_i,t}$'s), since if $\chi^{x_i,t} = true$ then this implies $\chi^{x_j,t-c} = true, \forall j, c \geq 0$. Thus all $T$'s can be attempted in polynomial time if *each* $T$ can be attempted in polynomial time. It remains to be shown that each solution can be selected and evaluated in polynomial time.

The selection process involves estimating the number of non-satisfying assignments for $k$CNF given each $x_i, \overline{x_i}$. The estimation takes polynomial time and there are a linear number of $x_i's$ to check, so the estimation, ordering, and selection can all be done in polynomial time in the size of $C_A$. Setting the new implications in 2CNF and kCNF takes linear time, as does the simplification process. Checking the assignment of each solution against the original formula is also linear time, so for each attempted solution we can select and evaluate in polynomial time in the size of $C_A$.

In polynomial time, we establish bounds $T_u, T_l$ such that $T_l \leq T_c \leq T_u$, where $T_c$ is the critical delay of the circuit. The overall running time of this algorithm is polynomial in the size of the analysis circuit. What is not known, however, is

how close bounds $T_u$ and $T_l$ come to the desired result of $T_c$. Can we establish a probability $P$ and bound $\epsilon$ such that $0 \leq P, \epsilon \leq 1$ and make the claim that, given polynomial time, $\frac{T_l}{1+\epsilon} \leq T_c \leq (1+\epsilon)T_u$ with probability greater than $P$? Currently the analysis is not sufficiently advanced to prove or disprove this claim in the general case of the constructive method.

## 6.8 Shortest sensitizable paths

As mentioned in chapter 2 and discussed in detail in chapter 7, when estimating circuit delays for the purposes of setting clock speeds it is desirable to know the lengths of either the longest sensitizable path or shortest destabilizing path in the circuit. For the latter, the shortest sensitizable path functions as a useful lower bound.

Similar techniques to those applied in this chapter for finding longest sensitizable paths could be applied to find shortest sensitizable paths - the key difference being the use of an alternative version of the $\chi$ values, as described below.

In the existing algorithm, the definition for $\chi^{g_i,t}$ is that, given a circuit, $C$, an input vector to the circuit, $X$, a gate in the circuit, $g_i$, and a time, $t$, then $\chi^{g_i,t}$ is *true* iff the output of gate $g_i$ takes time at least $t$ to stabilize.

In a shortest-path version, the definition would call for $\chi^{g_i,t}$ to be *true* iff the output of gate $g_i$ takes time *at most* $t$ to stabilize. This, as in the longest-path version, can be encoded as a logic function of the gates of $C$ and an encoding of time, $T$. The translation rules are as follows:

$$NOR, OR\ gates: \chi^{g_i,t} = \prod_{i=1}^{k} \left(\chi^{a_i,t-1}\right) + \sum_{i=1}^{k} \left[\chi^{a_i,t-1} \cdot a_i\right]$$

$$NAND, AND\ gates : \chi^{g_i,t} = \prod_{i=1}^{k} \left(\chi^{a_i,t-1}\right) + \sum_{i=1}^{k} \left[\chi^{a_i,t-1} \cdot a_i'\right]$$

$$XNOR, XOR\ gates : \chi^{g_i,t} = \prod_{i=1}^{k} \chi^{a_i,t-1}$$

$$NOT\ gates : \chi^{g_i,t} = \chi^{a,t-1}.$$

However, there is an interesting alternative suggested by the nature of the annotated CNF formula developed for use in bounding the length of the longest sensitizable path in the circuit. This alternative is discussed in the following section.

**De-randomizing the problem:**

We are given an (annotated) CNF formula that is satisfiable for some encoding of time $T$ if and only if the original circuit takes at least time $T$ to stabilize for some input vector $X$.

Given a specific $T$, under what conditions is the formula satisfiable for *every* value of $X$? If the formula is satisfiable for every value of $X$ (given some specific $T$) then every input to circuit $C$ requires at least time $T$ to stabilize. That is, $T$ is a lower bound on the length of the shortest sensitizable path in the circuit.

The largest $T$ value for which the formula is satisfiable for all $X$ values thus provides the length of the shortest sensitizable path in the circuit.

If the CNF formula is satisfiable for all $X$ values given a specific $T$, this means that no variables of $X$ appear in the CNF formula after $T$ is applied and the formula is simplified. The problem of determining the shortest sensitizable path length thus becomes the problem of determining the largest $T$ value for which the CNF formula unravels without placing restrictions on any $X$ values. Note that, *given the CNF formula*, this is an easier problem than that of determining the maximum $T$ value for which there exists a satisfying $X$ value.

In the latter case (longest path), to determine an upper bound[2] it is necessary to prove, in some manner, that all assignments are non-satisfying. In the former case (shortest path), to determine a lower bound it is necessary to show that no non-satisfying assignment exists. However, in a CNF formula, the presence of any non-empty clauses whose logic values are not already implied *true* by virtue of containing at least one *true* variable immediately present a non-satisfying assignment. Thus, for a specific $T$ value, it is only necessary to apply the $T$ value, unravel the formula without requiring a set logic value for any $X$ variable, and show that at least one non-empty clause has not already been fixed at logic value *true*.

There are two other useful observations in this respect:

- If the formula unravels with respect to $X$ for a specific $T$ value, then it also unravels for all lesser $T$ values.
- Similarly, if the formula does *not* unravel for a specific $T$ value then it does not unravel for any larger $T$ values.

[2]Remembering that it is more desirable with respect to clock speed to determine an upper bound on the length of the longest sensitizable path and a lower bound on the length of the shortest sensitizable path.

Thus again the problem of finding a maximum $T$ value is approachable by a binary search on a range of $T$ values. The binary search produces at most $O(log(T)) = O(m)$ iterations. For each of these iterations a $T$ value is applied to the CNF formula and the formula is simplified. The complexity of the process relative to the size of the original circuit is dependent on two factors:

- the computation required to simplify the annotated CNF formula given a $T$ value, and
- the size of the annotated CNF formula relative to the size of the original circuit $C$.

Furthermore, this approach, used in conjunction with the alternative $\chi$ value rules discussed above, suggests a possible deterministic solution to the long sensitizable path problem. Bringing these problems out of the randomized domain is seen as a key point for future work.

In chapter 7, long sensitizable paths together with short destabilizing paths are used to determine circuit clock speeds, to identify circuit components which are sensitive to delay increases and/or decreases, and to target circuit components for gate resizing as part of circuit timing optimization.

## 6.9 Benchmark results

In this section, the results of applying an implementation of the techniques of this chapter to a set of benchmark circuits are discussed. The comparisons are based on the CPU times required by the algorithm and the upper/lower bounds established for $T_c$ for each circuit in those times.

Table 6.1: Time comparisons with existing techniques

| | CNF trans. | | Topological | functional analysis techniques | | | | |
|---|---|---|---|---|---|---|---|---|
| circuit | CPU | memory | search | [40] | [26] | | [9] | [36] |
| C6288 | 5338.4s. | 13.6M. | 722.9 | 33.6 | 69.3 | (96%) | 25.7 | 196.9 |
| C1908 | 773.8s. | 10.0M. | 8.3 | 8.2 | 9.4 | (100%) | 31.6 | 18.1 |
| C432 | 24.0s. | 3.8M. | 0.4 | – | – | | 1.3 | 2.4 |
| C499 | 334.4s. | 8.0M. | 1.0 | – | – | | 1.9 | 2.9 |
| C1355 | 282.6s. | 8.8M. | 0.3 | – | – | | 4.8 | 7.4 |
| C3540 | 1198.0s. | 16.9M. | 459.1 | 7.2 | 14.0 | (59%) | 39.4 | 58.4 |
| C880 | 121.8s. | 6.1M. | 9.9 | – | – | | 2.7 | 5.0 |
| C5315 | 5791.5s. | 13.2M. | 118.8 | 10.5 | 20.6 | (42%) | 35.8 | 91.4 |
| C7552 | • | • | 93.7 | 6.8 | 49.6 | (44%) | 26.1 | 133.4 |
| C2670 | 1218.1s. | 23.1M. | 552.8 | 22.4 | 8.8 | (35%) | 11.5 | 14.1 |

• insufficient memory to complete.

The comparison algorithms considered are the non-enumerative techniques discussed in chapter 5, and are again briefly summarized:

- [40] - McGeer et al, 1991: uses circuit analysis and path recursive functions.
- [26] - Huang et al, 1993: uses an heuristic approach with polynomial cut-off time, the percentages given are the authors' own rated accuracy of the output.
- [9] - Chang et al, 1993: uses VIPER, a path extraction tool.
- [36] - Maurer, 1992: using compiled circuit simulation.

Unfortunately, from a practical standpoint the results of the initial implementation are disappointing. While the results are still superior in some cases to the earlier enumerative techniques, they are clearly not comparable with the non-enumerative techniques cited above.

The area in which most of the computation time is absorbed in this implementation is in fact the variable-ranking scheme. While the size of the CNF formula, measuring by the number of literals and clauses, is linear in the size of the original

circuit[3] and the running time of the variable ranking scheme is polynomial in the size of the CNF formula, the resulting polynomial ($\simeq 100G^2$, for $G$ gates in the original circuit) does not yield practical results for circuits in which the number of gates is in the tens of thousands or higher.

The number of variables and clauses associated with the initial formula renders a complete ranking of the variables impractical - the *true / false* estimations can be carried out for only a subset of these variables. In the implementation considered here, the subsets are chosen based on the number of clauses a variable appears i, and the number of estimations carried out for each clause in linear in the number of circuit inputs. Even given these restrictions however, the required CPU time for the estimation process remains unacceptably high.

There are several other possible areas for future work with respect to operational parameters of the algorithm. Specifically, the impact of the choice made for the number of $T$ vectors searched, how many vectors are used to estimate the number of non-satisfying assignments to the $k$CNF formula for different variable settings, and how often these estimates are updated. Such parameters are overshadowed by the need for an alternative ranking scheme however.

It is expected that significant improvements in both the time and memory requirements can be obtained through improvements to the data structures used in the initial implementation, but it is *not* expected that these improvements are enough to result in performance generally superior to the non-enumerative techniques discussed above. To use the CNF-translation based technique to obtain performance comparable to those techniques, it is expected that a "de-randomized" approach, such as that outlined in section 6.8, must be adopted. Such a technique

[3]In fact, the number of literals and the number of clauses is consistently in the range of ten to twelve times the number of gates in the original circuit.

could be used to eliminate the estimation process entirely. As pointed out in the closing paragraphs of that section, this is seen as an essential part of future work in the area.

## 6.10 Chapter summary

This section contains a summary of the work presented on the CNF-translation technique for determining bounds on the length of the longest sensitizable path in a circuit.

The summary is performed in four parts. The first part summarizes the properties of the analysis circuit used for the algorithm. The second part summarizes the algorithm itself. The third part of the summary focuses on the theoretical analysis performed on the algorithm, and the fourth part focuses on comparisons of the algorithm with previous existing works.

### 6.10.1 The analysis circuit

The analysis circuit, $C_A$, is a circuit constructed based on an original circuit $C$. $C$ takes as input an $m$-bit vector $X$, and $C_A$ takes as input $X$ and $T$, where $T$ is an encoding of time, $t$. $C_A$ is a single output circuit, and the output of $C_A$ given $X, T$ is *true* iff the outputs of circuit $C$ require at least time $t$ to stabilize if $C$ is given vector $X$ as input at time 0.

The importance of $C_A$ with respect to the new algorithms is as follows. For any $T$, if there exists some $X$ which satisfies $C_A$ given $T$, then $T$ defines a lower bound on the length of the longest sensitizable path in $C$. If no such $X$ exists, then $T$ defines an upper bound.

By repeatedly choosing $T$ values between the current known upper and lower bounds, then determining if $C_A$ is satisfiable given $T$, we can continually improve the known bounds on the length of the longest sensitizable path in $C$. The key contribution of our algorithm is a fast probabilistic means of determining whether or not $C_A$ is satisfiable given a specific $T$ value.

### 6.10.2 Algorithm summary

The algorithm works by taking the analysis circuit, $C_A$, and observing that for a given $T$ we can bound $T_c$ further if either (1) $C_A$ can be proven to be unsatisfiable under $T$, or (2) if a satisfying assignment can be found in the input space of $X$. The maximum $T$ value for which the CNF formula is satisfiable is sought in two different ways, one constructive, and one non-constructive.

First, a non-constructive estimation technique is used. This technique provides a correct result within $\frac{T_c}{1+\epsilon}, T_c(1+\epsilon)$ with probability at least $1-\alpha$ provided the number of satisfying assignments to the formula given $T_c$ is at least $\frac{2^m}{p(m)}$ for some polynomial $p$. Furthermore, if the set of satisfying assignments forms a single cube on the input space, then the cube of satisfying assignments can be exactly identified in polynomial time, again with probability at least $1-\alpha$. If the number of satisfying assignments does *not* match the criteria specified above, then the result of this technique is to produce the maximum value, $T'$, for which the number of satisfying assignments for $T$ values greater than or equal to $T'$ is at least $\frac{2^m}{p(m)}$ for some polynomial $p$.

In the second method a constructive technique is used. A satisfying solution is sought for the CNF formula by splitting it into two parts - a set of binary clauses (solvable in linear time), and a much smaller set of clauses each of which

contains at most $k$ literals. Estimations are obtained for the probability of each input setting resulting in an unsatisfiable kCNF formula, and use these estimates are used to guide the search of the solution space for the 2CNF formula, seeking a solution which extends to a general solution for the entire formula.

Note that in one special case - where all the AND/OR gates have fan-in *exactly* $k$, then the Lovasz local lemma holds. Thus in the kCNF formula if each variable appears in at most $2^{\frac{k}{100}}$ clauses then the formula *is* satisfiable.

In section 6.8, it is shown that the length of the shortest sensitizable path in the circuit is also obtainable in either of two ways:

- Shortest sensitizable paths are obtainable using a similar approach to CNF translation for longest sensitizable paths, but based on the development of an annotated CNF formula in which the $\chi^{g,t}$ values are based on the premise of being *true* iff $g$ takes time *at most* $t$ to stabilize, rather than *at least* $t$.
- Shortest sensitizable path lengths can also be derived deterministically, using the annotated CNF formula discussed for the main algorithm of this chapter. This is accomplished by searching for the largest $T$ value for which the formula is never non-satisfiable.

The union of these two approaches, the alternative formulation of the CNF formula combined with the search for the largest $T$ value which is never non-satisfiable, also suggests a technique for de-randomizing the sensitization algorithms discussed here. This de-randomization process is seen as a key part of future work.

### 6.10.3 Theoretical analysis

In section 6.1.4 the size of the analysis circuit is analyzed relative to the size of the original circuit, and it is shown that the number of annotated clauses is linear in the size of the original circuit. The number of literals associated with each annotated clause is bounded by a constant if bounded fanin is assumed for gates in the original circuit. The number of $t$ values associated with each annotated clause, however, has the potential to grow exponentially with the depth of the original circuit, up to a maximum value bounded by the number of distinct topological pathlengths found in the original circuit.

In section 6.6 the running time of both the constructive and non-constructive solutions is analyzed, and it is shown that:

1. In its current form, the constructive solution runs in time polynomial in the size of the analysis circuit, but without confidence levels on the accuracy levels of the upper and lower bounds produced.

2. Given that the number of satisfying assignments is at least $\frac{2^m}{p(m)}$ for some polynomial $p$, then with probability at least $1 - \alpha$ the value $T_c$ can be estimated to within the range $\frac{T_c}{1+\epsilon}, T_c(1+\epsilon)$ in time polynomial in the size of the formula and for any pre-specified values of $\epsilon$ and $\alpha$. Furthermore, if the satisfying assignments form a single cube on the input space, then that cube can be exactly identified in polynomial time, again with probability at least $1 - \alpha$.

### 6.10.4 Comparison to existing works

In this section comparisons are made between published results for other solutions to the false path problem and an implementation of the CNF translation algorithm. These results are in the form of timing comparisons between the implementation and the non-enumerative techniques discussed in chapter 5, sections 5.6.3-5.6.4. The path-enumeration and checking schemes discussed in those sections are not repeated here, as they are considered too inefficient for large circuits.

As is the case for the topological search algorithm, the benchmark results obtained for the CNF translation algorithm are still not comparable with existing deterministic techniques. While it is expected that memory and time requirements can be significantly reduced by improvements to the implementation, it is *not* expected that the results obtained from these improvements will be superior to those obtained in some existing techniques.

It is recommended that future work on the CNF translation technique focus on "de-randomizing" the algorithm, following the techniques suggested in section 6.8.

### 6.10.5 Future work

The key areas for future work are to study the suggested derandomization procedure of section 6.8, to complete the complexity analysis for the general constructive solution with respect to the accuracy of the upper and lower bounds after a specified number of $T$ values have been attempted, and to refine the guidance techniques for the ordering of variables in the search of the 2CNF solution space.

# Chapter 7

# Applications in optimization, fault testing

In this chapter, the techniques developed in chapters 5 and 6 are applied to the problems of setting the circuit clock speed, timing optimization by gate resizing, and identification of delay-fault-sensitive components in the circuit. All these results assume purely combinational logic and a floating delay mode (the clock estimates obtained are not guaranteed robust under monotone speedup models).

In section 7.1 an overview of these problems is provided, and introduce the notation and definitions used in the chapter are introduced. In section 7.2 the use of long sensitizable paths and short topological paths to determine circuit clock settings is discussed. In section 7.3, the path lengths and clock settings are used to identify component sensitivity to delay increases and delay decreases, and in section 7.4, the use of component sensitivity is discussed as a guide to circuit timing optimization using gate resizing. The steps which can be taken to produce a practical design tool for timing optimization, based on the techniques presented in this dissertation, are summarized in section 7.5, and in section 7.6 all the results presented in this chapter are summarized.

## 7.1 Introduction

In this section an overview is provided for a component classification process. Each stage in this process is discussed in greater detail in later sections.

It has been shown [13] that knowledge of the maximum difference between the longest sensitizable path and the shortest topological path provides a good upper bound on minimum clock speed for a circuit. Once the clock speed has been set, circuit components are classified by the length of the long sensitizable paths and short topological paths which pass through them.

Using this information, the components in the circuit can be classified according to whether they are:

a intolerant to delay decreases: this implies a delay decrease in the component increases the required clock speed of the circuit,

b intolerant to delay increases: this implies a delay increase in the component increases the required clock speed of the circuit,

c intolerant to any delay alterations: this implies any delay change in the component increases the required clock speed of the circuit.

Components which satisfy any of [a], [b], or [c] render the circuit sensitive to delay faults, and components which satisfy conditions [a] and [b] but *not* [c] can be targeted for optimization, as is shown in section 7.5.

### 7.1.1 Notation and definitions

In this section the notation and definitions used in this chapter are provided. Some definitions originally provided in chapter 2 are repeated here for clarity.

$C$: the circuit under analysis

$m$: the number of primary inputs to $C$

$n$: is the number of primary outputs to $C$

$X_i =< x_1, \ldots, x_m >$: is an input vector to $C$, $i \in 1 \ldots 2^m$

$T_s(X_i)$: the stabilization time for the circuit given $X_i$, is the amount of time taken for all $n$ outputs of $C$ to stabilize after input vector $X_i$ is applied to the circuit. For our purposes sensitization is used with floating delay values prior to the application of $X$.

$T_c$: the critical time of the circuit, is the maximum $T_s$ value, taken across all $2^m$ input vectors

$T_{top}$: the length of the topologically-longest path in $C$, note again that $T_{top} \geq T_c$.

$T_{short}$: the length of the topologically-shortest path in $C$, noting $T_{short} \leq T_c$.

$T_{clock}$: the clock setting used for $C$ - the minimum time for which the values of input vector $X$ are held constant once applied to $C$, and the time elapsed between successive samplings of the primary outputs of $C$.

## 7.2 Path lengths and clock settings

In this section the relevance of path lengths is shown to the problem of determining optimal clock speeds for combinational logic blocks.

The length of the longest topological path in a circuit clearly provides an upper bound on the optimal clock speed of a combinational circuit. If the longest topological path length is given by $T_{top}$, and an input vector is applied to the circuit at time $t_0$, then clearly by time $t_0 + T_{top}$ all circuit outputs have stabilized. However, since not all paths in the circuit are sensitizable, $T_{top}$ is a pessimistic estimator of stabilization time.

$T_c$ represents the length of the longest sensitizable path in the circuit, and $T_c \leq T_{top}$, therefore $T_c$ is a better estimator of maximum output stabilization time and, hence, a better estimator of the optimal clock speed.

However, $T_c$ does not necessarily represent a tight bound on the optimal clock speed for a circuit. Let us consider an example. Suppose $T_s(X_a) = T_c$, and we have chosen to use $T_c$ to set the clock speed for the circuit under analysis, i.e. $T_{clock} = T_c$. At time $t_0$ apply an input vector, $X_i$, then at time $t_0 + T_{clock}$ the outputs of $C$ are sampled and a new input vector, $X_j$, is applied to $C$.

If the value $T_{short}$ is known, then it is also known that the outputs of $C$ cannot change in the time period $t_0 + T_{clock}$ to $t_0 + T_{clock} + T_{short}$. In fact, if input vector $X_j$ is applied at time $t_0 + T_s(X_i) - T_{short}$, the outputs of $C$ still do not change prior to time $t_0 + T_s(X_i)$. Thus an improved estimate for the optimal clock speed is obtained using the $X_i$ input vector which maximizes $T_{clock} = T_s(X_i) - T_{short}$, out of all $2^m$ possible input vectors.

The sensitization value, $T_s(X_i)$, can be found efficiently using the methods described in chapters 5 and 6. The shortest topological path can easily be determined in polynomial time using the dual of the longest-topological-path algorithm.

## 7.3 Circuit sensitivity to component delays

Given that the method of section 7.2 is used to establish $T_{clock}$, we now introduce techniques to evaluate the sensitivity of the circuit to changes in the delay associated with individual gates and lines in the circuit.

The circuit sensitivity to the delay of each component (line *or* gate) in the circuit is approximated by taking a set of vectors for which $T_s(X_i) - T_{short} = T_{clock}$, and for each vector in the set propagating the logic values and associated stabilization and shortest topological path length forward through the circuit.

If $X_i$, is applied to the circuit inputs at time $t_0$ and the circuit outputs are sampled at time $t_0 + T_{short} + T_{clock}$, and the next vector in the input sequence applied to the circuit inputs at time $t_0 + T_{short}$, then the following observations hold:

1. At least one primary output has just stabilized at time $t_0 + T_{short} + T_{clock}$.
2. At least one primary output may destabilize immediately after time $t_0 + T_{short} + T_{clock}$.
3. The two sets of primary outputs covered by cases 1 and 2 above may or may not have elements (primary outputs) in common.
4. Any primary output which stabilizes before time $t_0 + T_{short} + T_{clock}$ can tolerate an increased stabilization time equal to the amount of time by

which the output stabilizes prior to the output sampling taking place.

5. Any primary output which can destabilize after time $t_0 + T_{short} + T_{clock}$ can tolerate a decrease in shortest topological path length equal to the amount of time by which the output can destabilize (judged by shortest topological paths) following the output sampling taking place.

These windows of tolerance for stabilization-time increases and shortest topological path decreases can now be propagated "backwards" from primary outputs to primary inputs, and the tolerance of each component in the circuit for delay increases and decreases can be determined *for that input vector.* By selecting input vectors which maximize the required $T_{clock}$ value, we guarantee that we find the components for which any stabilization time increase, or any shortest topological path decrease, increases the required clock period of the circuit.

For components identified as having no tolerance for delay decreases, any decrease in delay results in an increase in the required $T_{clock}$, and for components identified as having no tolerance for delay increases any increase in delay results in an increase in the required $T_{clock}$. For components having a tolerance greater than 0 in these areas, a delay increase/decrease within the tolerance bounds does not increase the "current" critical path's delay beyond the requirements of $T_{clock}$, but there are no guarantees that a different path - which was previously *not* critical - is not increased beyond the limits of $T_{clock}$. This is due to the fact that the tolerance bounds are here only computed for vectors which maximize $T_s(X_i) - T_{short}$. If correctness is desired for all tolerance windows of range $0 \ldots \beta$ then vectors which produce $T_s(X_i) - T_{short} \geq \beta$ must be applied.

To summarize, the results presented are guaranteed to detect all components with a zero tolerance for delay decreases or a zero tolerance for delay increases.

Tolerance levels greater than zero are a useful metric for evaluating the sensitivity of a component, but are not guaranteed.

## 7.4 Sensitivity and fault tolerance

It is clear that fault tolerance must be expanded to include tolerance of delay-altering defects [51]. Using clock speeds which are calculated on both stabilization times and destabilization bounds (via short topological path lengths) means delay defects which increase *or decrease* the delay associated with a specific component (line or gate) can result in a decrease in circuit performance.

Practical limitations on the size of the test set which may be applied force the circuit designer to either design a circuit with delay testing in mind [18] or to try and identify key delay faults to test for [21] [42] [45] [48] The sensitivity criteria discussed in section 7.3 are very useful in the area of delay fault tolerance, in that they identify the set of components which are intolerant to delay faults under optimal clock setting. Even more, they identify specifically whether the component is sensitive to delay increases, or decreases, or both.

In the section below, the sensitivity algorithms are applied to a set of benchmark circuits. The circuits are evaluated with respect to the effect delay increases and decreases in individual gates has on circuit operation.

### 7.4.1 Benchmark results: gate sensitivity

In this section, the sensitivity approaches discussed above are applied to a set of benchmark circuits The tables in this section summarize, for each circuit, the number of gates for which delay defects render $T_{clock}$ invalid. The gates are

subdivided into those which are only tolerant of delay decreases, those which are only tolerant of delay increases, and those which are tolerant of neither delay increases *nor* delay decreases. (The number of gates which tolerate both decreases and increases is obtainable by the difference between these totals and the total number of gates in the circuit.)

Table 7.1: Circuit sensitivity to changes in gate delays

| | total | Number of gates in which circuit tolerates | | |
|---|---|---|---|---|
| circuit | gates | delay increases only | delay decreases only | no delay changes |
| b1 | 22 | 2 | 4 | 10 |
| cm42a | 43 | 1 | 0 | 32 |
| majority | 17 | 3 | 4 | 3 |
| C17 | 20 | 3 | 4 | 7 |
| decod | 21 | 1 | 0 | 28 |
| cm82a | 39 | 5 | 5 | 23 |
| cm138a | 36 | 3 | 0 | 20 |
| z4ml | 81 | 12 | 12 | 35 |
| f51m | 159 | 18 | 8 | 53 |
| ldd | 131 | 16 | 13 | 79 |
| 9symml | 178 | 4 | 5 | 24 |
| x2 | 69 | 4 | 5 | 24 |
| alu2 | 376 | 6 | 76 | 54 |
| cm152a | 45 | 5 | 18 | 9 |
| cm85a | 56 | 14 | 1 | 22 |
| cm151a | 47 | 7 | 10 | 11 |
| cm162a | 63 | 7 | 6 | 24 |
| cu | 83 | 2 | 7 | 41 |
| pm1 | 82 | 4 | 2 | 45 |
| cmb | 70 | 17 | 0 | 51 |
| cm163a | 76 | 9 | 6 | 17 |
| parity | 93 | 15 | 1 | 62 |

Gates in this case include primary inputs and outputs.

As is evident from the data in table 7.1, many gates in the circuits under analysis are tolerant (at least to a limited extent) of either delay increases, or delay decreases, or both. When seeking to develop a small set of test vectors for delay faults, small delay increases due to defects are less likely to cause a fault in

components which are tolerant of delay increases, hence it is "less vital" to test for faults due to delay increases in these components. Similarly it is less important to test for faults due to delay decreases in those components which are categorized as being tolerant of delay decreases. Thus the designer of a test set can concentrate first on detecting the critical faults - faults for which the component is known to be sensitive - and then gradually expand the testing to include other components and fault types as time and resources permit.

In table 7.1 it is also shown that the set of gates which are intolerant to any delay change can represent a substantial portion of the circuit. For example, more than half the gates in each of 9 different circuits in the benchmark set are intolerant of any delay change at the estimated optimal clock settings. This is significant in that these gates need not be considered when targeting gates for resizing, a fundamental part of the timing optimization approach discussed in section 7.5.

## 7.5 Timing optimization through gate resizing

In this section we consider the use of the sensitivity classification scheme discussed in section 7.3 for circuit timing optimization. When dealing with custom or semi-custom IC's[1], it is sometimes possible to alter the delay associated with a logic gate by resizing the gate [50]. For instance, in the case of CMOS implementations, gate delays can be reduced by increasing the size of a gate. The drawbacks associated with this approach being the increased size of the circuit, consumption, and the difficulty of identifying an optimal set of gates for resizing.

Determining optimal sizings of the gates is known to be an NP-hard problem

[1]These options are less practical when the choice is to be made from a library of standard gates.

[32]. Other attempts to provide fast resizing techniques have recently been made [20], but the computation time involved in these techniques is prohibitively high. Our sensitivity classifications quickly provide a good metric for targeting gates as follows:

1. If a gate is identified as having zero tolerance for delay increases and a zero tolerance for delay decreases then we do not target the gate for resizing.

2. If a gate has a wide tolerance range for both delay increases and delay decreases, then, while resizing is unlikely to be harmful, it is also unlikely to markedly improve the operation of the circuit.

3. If a gate has a wide tolerance range for delay decreases but a zero tolerance for delay increases, then increasing the size of the gate has a strong potential to improve the operation of the circuit, and minimal potential to be detrimental.

4. If a gate has a wide tolerance range for delay increases, but a zero tolerance for delay decreases, then decreasing the size of the gate has a strong potential to improve the operation of the circuit, and minimal potential to be detrimental.

In the following section we summarize the results of some experiments applying these targeting metrics to timing optimization of a set of benchmark circuits.

### 7.5.1 Benchmark results: gate resizing

The metrics listed in the preceding section are here applied to a set of benchmark circuits. The circuit is repeatedly fed through an optimization loop in which:

1. The topological search algorithm is applied to determine $T_{clock}$ and a set of vectors $S = \{X_i : T_{clock} = T_s(X_i) - T_{short}\}$.

2. The sensitivity algorithm, using $S$, is applied to determine the tolerance of each component in the circuit to delay increases and decreases.

3. Out of the components with zero tolerance for delay decreases, the component with the maximum tolerance for delay increases is chosen.

4. Out of the components with zero tolerance for delay increases, the component with the maximum tolerance for delay decreases is chosen.

5. The component with the greater tolerance out of [3] and [4] above is chosen as the targeted component, and is resized (larger or smaller, as appropriate).

The resizing criteria here are very simple: a gate may only be targeted for resizing once, and the effect of resizing is to either increase or decrease the delay associated with the gate by 20%.

The effect of the optimization process on the set of benchmark circuits is summarized in table 7.2. As can be seen from these results, even a very simple timing optimization scheme can improve circuit performance by modifying a small set of gates targeted by our sensitization scheme. In section 7.6 we describe the steps necessary to use the results described in this dissertation to produce a practical design tool for timing optimization.

Table 7.2: Timing optimization through gate resizing

| benchmark circuit | total gates | gates resized | timing improvement | gates resized | timing improvement |
|---|---|---|---|---|---|
| b1 | 15 | 1 | 9.8 % | 2 | 18.0% |
| cm42a | 29 | 3 | 0.7 % | 8 | 0.9% |
| majority | 11 | 1 | 0.8 % | – | – |
| C17 | 13 | 1 | 0.8 % | – | – |
| decod | 25 | 0 | 0.0 % | – | – |
| cm82a | 31 | 3 | 12.1% | 5 | 16.2% |
| cm138a | 22 | 0 | 0.0 % | – | – |
| z4ml | 70 | 6 | 2.9 % | 14 | 5.1% |
| f51m | 143 | 11 | 3.5 % | 22 | 4.8% |
| ldd | 103 | 14 | 1.2 % | 16 | 2.0% |
| 9symml | 168 | 6 | 0.2 % | – | – |
| x2 | 52 | 3 | 1.0 % | 7 | 3.1% |
| alu2 | 360 | 18 | 1.9 % | 24 | 7.0% |
| cm152a | 33 | 0 | 0.0 % | – | – |
| cm85a | 42 | 2 | 2.8 % | 8 | 7.5% |
| cm151a | 33 | 2 | 2.9 % | 8 | 5.0% |
| cm162a | 54 | 17 | 2.9 % | – | – |
| cu | 58 | 20 | 1.6 % | – | – |
| pm1 | 53 | 5 | 2.9 % | – | – |
| cmb | 54 | 7 | 3.5 % | 12 | 5.0% |
| cm163a | 55 | 14 | 2.6 % | – | – |
| parity | 76 | 3 | 2.9 % | 7 | 4.5% |
| C1908 | 718 | 4 | 4.4 % | 18 | 5.8% |
| C1355 | 619 | 7 | 1.9 % | – | – |
| C432 | 209 | 2 | 0.6 % | 21 | 2.1% |
| C499 | 579 | 8 | 2.6 % | – | – |
| C3540 | 1161 | 5 | 0.1 % | – | – |
| C2670 | 1100 | 12 | 3.2 % | 22 | 11.2% |
| C6288 | 2400 | 12 | 0.2 % | 15 | 0.4% |
| C5315 | 1938 | 17 | 1.7 % | 22 | 2.6% |
| C880 | 394 | 6 | 2.5 % | 15 | 4.5% |
| C7552 | 3087 | 0 | 0.0 % | – | – |
| tcon | 72 | 8 | 12.9 % | 26 | 16.7 % |
| pcle | 86 | 1 | 1.3 % | 13 | 2.8% |
| sct | 119 | 1 | 5.6 % | 14 | 7.9% |
| mux | 90 | 2 | 1.0 % | 15 | 2.7% |
| cc | 59 | 23 | 3.0 % | – | – |
| alu4 | 720 | 5 | 2.5 % | 26 | 8.0% |
| cm150a | 70 | 7 | 1.9% | 14 | 5.0% |

– no further improvement made in first 32 gates gates considered

## 7.6 The next step

As discussed in chapters 5 and 6, estimates for the longest sensitizable path in a circuit can be obtained more quickly using circuit analysis techniques than when using the topological search. However, the circuit translation problem becomes more complex when trying to maximize $T_s(X_i) - T_{short}$.

The next logical step in the development of component sensitivity identification and timing optimization is to develop a set of translation formulas from $C$ to $C_A$ to $CNF$ formula that account for stabilization and shortest topological path lengths simultaneously. Once this is done, the timing optimization results should be obtainable more quickly.

The timing optimization model presented above also requires improvement. In the present form, it is assumed gate delays can be increased or decreased by 20%, once only. A more realistic model might be to provide a set of several sizes of each gate, such as would be found in a standard cell library, and allow the timing optimizer to select both the gate to resize, and the new size of that gate. Estimates on area changes, based on the size of gates before and after resizing, would also be useful information for the optimizer to produce.

The combination of analysis circuit-based timing optimizer and more accurate gate-size information should result in an extremely practical design tool for circuit optimization.

## 7.7 Chapter summary

In this chapter, the applicability of our timing analysis techniques is shown in three areas: determining optimal clock speeds, circuit timing optimization, and the identification of circuit components which are sensitive to delay faults.

A metric is provided for obtaining good estimates of optimal clock speed: maximizing $T_{clock} = T_s(X_i) - T_{short}$ over all possible input vectors $X_i$.

A method is described which uses this metric, plus a set of input vectors which require the maximized value of $T_{clock}$, to estimate the sensitivity of the circuit to increases and decreases in the delay associated with each gate in the circuit. These sensitivity measures are guaranteed to identify components for which the circuit is intolerant of any delay increase, and components for which the circuit is intolerant of any delay decrease. Approximations of sensitivity for all other components are also given, but the results are only guaranteed to be accurate with respect to the current critical path in the circuit. The sensitivity measures are useful in two respects, described in the following two paragraphs.

Firstly, the measures directly indicate components in the circuit in which no delay faults can be tolerated, and hence the components which most need to be tested when developing delay fault test sets. Furthermore, the measures indicate whether it is delay increases, decreases, or both, to which the component is sensitive. The sensitivity measures for a set of benchmark circuits are summarized in section 7.4.1.

Secondly, the measures can be used in circuit timing optimization for the design of custom and semicustom ICs as described in section 7.5. By selecting components which have zero associated tolerance for delay increases but a large

associated tolerance for delay decreases (or vice versa), resizing of the component has a good chance of resulting in improved overall timing performance of the circuit, and a minimal chance of degrading overall timing performance. The effectiveness of the targeting scheme for component resizing is tested over a set of benchmark circuits in section 7.5.1.

Finally, suggestions are made with respect to developing an analysis-circuit-based timing optimizer which should, in conjunction with more accurate gate size and time information, provide the basis for a practical new design tool for circuit optimization.

# Chapter 8

# Conclusions

This dissertation contains a series of results on the studies of timing behaviour in combinational logic circuits. The primary results are two new randomized algorithms to determine the length of the longest sensitizable path in a circuit, and their applications to circuit timing analysis, optimization, and delay fault testing. In addition, there is a study of the impact the choice of delay model has on analysis of expected circuit behaviour.

The algorithmic results are presented in chapters 5 and 6, for the topological search algorithm and CNF satisfiability algorithm respectively. These results are also summarized in this chapter in sections 8.2 and 8.3. The delay modeling results are presented in chapter 3, and summarized in section 8.1 below, and the applications of the algorithms in the areas of optimal clock settings, timing optimization, and delay fault testing are presented in chapter 7 and summarized in section 8.4.

## 8.1 Delay models

In chapter 3, a set of gate delay models is analyzed with respect to the impact on modeled circuit delay behaviour. The simplest model considered is the unit delay model commonly used in research by the academic community, and the most detailed model considered is the model provided in cell library 2 with the MCNC benchmark sets. The other four "intermediate" models considered are simplifications of the most detailed model.

The models are analyzed by considering, over a set of standard benchmark circuits, the longest sensitizable path in the circuit under the model, the longest topological path in the circuit under the model, the number of input vectors which sensitize maximum length paths under the model, and the grouping of these vectors in the input space.

With respect to path sensitivity, specifically the identification of the longest true path in a circuit and the identification of long false paths in a circuit, the key factor in choosing a model appears to be the distinction between rise delays and fall delays. Models which make no distinction between rising and falling delays, such as the unit delay model, tend to classify longest paths in a circuit as sensitizable when the longest path in the circuit is *not* sensitizable under a more realistic model. Thus many algorithms which use path enumeration/path checking to determine the longest sensitizable path in a circuit are achieving artificially good results, with respect to efficiency, when applied using the unit delay model. Under more realistic model these algorithms are forced to search many more long false paths before finding a true path.

Given the sophisticated analytical techniques being employed to determine true circuit delays, the continued use of such overly simplistic models as 'unit

delay' seems inherently flawed.

## 8.2 Topological search solutions

The topological search algorithm is the first of two algorithms developed in detail in this dissertation for the purpose of determining the maximum true delay in a circuit. The algorithm works by performing a directed search of the circuit input space, simulating the application of input vectors to the circuit with the goal of finding at least one input vector which sensitizes a maximum length sensitizable path in the circuit.

For both this algorithm, discussed in chapter 5, and the CNF satisfiability algorithm discussed in chapter 6, delays are calculated assuming floating values (unknown values on internal lines prior to the application of input vectors) and using the detailed delay calculation model supplied with the MCNC benchmark set.

The basic operation of the algorithm is as follows: select a random starting vector, and simulate it on the circuit to calculate the delay associated with the vector. Following this, simulate each adjacent input vector. If one of the adjacent input vectors has greater associated delay then make that vector the "current" input vector, and continue the search by checking input vectors adjacent to the new current vector. In this manner, the input space is searched until the locally maximal vectors (maximal with respect to associated delay) are found. The goal is to find a global maximal vector by repeating this process from a number of different starting points.

The worst case performance of the algorithm is shown to be exponential in

the number of circuit inputs, and this worst case arises from the situation in which all vectors adjacent to the global maximum are local minima. Fortunately, the performance of the algorithm over the benchmark set indicates much better performance than that suggested by the worst case. More accurate performance estimates are obtained from an adjusted worst case estimator, which takes into account the number of vectors which sensitize maximum length paths in the circuit, though this is information not ordinarily available prior to the application of a timing analysis tool.

Expected case behaviour is modeled using an assumption that no two input vectors produce identical circuit delays, and applying a set of analysis techniques first proposed by Harper for minimizing errors in data transmission[23].

While the predicted behaviour of the algorithm under the model is poor, exhibited behaviour over a benchmark set is much better than that of many existing delay estimation techniques. In particular, the algorithm is generally superior to techniques based on path enumeration or path checking, especially when applied to large designs which may contain many long false paths. The algorithm is less efficient than recent analysis techniques which are not path-based, however. Advanced analysis techniques, such as those described by McGeer in [38] and those discussed in chapter 6 of this dissertation, are expected to be generally more efficient than the topological search algorithm - particularly if applied to those large designs in which only a small number of input vectors sensitize longest paths.

## 8.3 CNF satisfiability solutions

In chapter 6, a second technique for estimating the length of the longest sensitizable path in a circuit is developed. This technique is based upon the ideas of

three other authors:

- McGeer and Brayton, in [38], proposes a method by which a circuit can be translated into an analysis circuit. In addition to containing information about the logic state of each internal line of the original circuit, the analysis circuit also incorporates information about the stabilization time of each line segment in the original circuit.
- Larrabee, in [31], proposes a method by which a circuit can be translated into a boolean expression in conjunctive normal form in which the majority of the clauses contain only two literals.
- Karp, Luby, and Madras, in [28], describe a technique by which the number of satisfying assignments to a formula in disjunctive normal form can be estimated to within $\epsilon$ in polynomial time with probability at least $1 - \alpha$.

In our algorithm, we take the analysis circuit proposed by McGeer, translate it into a CNF formula as suggested by Larrabee, and apply a dual of Karp's estimation techniques to search for either the maximum delay value for which the formula is satisfiable (via a non-constructive technique), or, a satisfying assignment which maximizes the delay value (via a constructive search technique).

The non-constructive technique functions by performing a binary search on the range of $T$ values, seeking to improve known upper and lower bounds on the critical path length, $T_c$. For each $T$ value checked, the estimation technique is applied to determine if the number of satisfying solutions to the formula given that $T$ value is greater than zero. For this technique, we describe the conditions under which the length of the critical path can be determined, with probability at least $1 - \alpha$, in polynomial time in the size of the analysis circuit. The key condition is that the number of assignments which satisfy the formula given time

$T_c$ must be at least $\frac{2^m}{p(m)}$ for some polynomial $p$. If the conditions do *not hold*, then with probability at least $1-\alpha$ in polynomial time the non-constructive technique produces an estimate $T_e$, such that $T_p \leq T_e \leq T_c$, where $T_p$ is the maximum time value such that at least $\frac{2^m}{p(m)}$ vectors satisfy the formula for $T$ values greater than or equal to $T_p$. Furthermore, it is shown that the if the set of satisfying vectors for the $T$ value identified by the non-constructive algorithm form a single cube on the input space, then that cube can be exactly identified in polynomial time, with probability at least $1-\alpha$.

The constructive algorithm operates as follows: the number of satisfying assignments to the formula is estimated twice for each variable - once given that the variable is assigned logic value *true*, and once given that the variable is assigned logic value *false*. This information is used to determine a probable setting for one of the variables in the formula. Given this variable setting, the formula is simplified, and a new variable is chosen. This process is completed until either the formula unravels (all remaining variable values are implied), or a contradiction is reached. If a contradiction is reached then a new attempt is necessary to find a satisfying assignment, and if a satisfying assignment is reached then a new lower bound on $T_c$ is established. If the formula is proven unsatisfiable for a given $T$ value then a new upper bound on $T_c$ is established. It is shown that the algorithm as described in chapter 6 terminates in polynomial time in the size of the analysis circuit, but the accuracy bounds achieved in that time are not yet known.

It is shown in section 6.8 that a similar approach, using a variant on the CNF formula can also be used to determine the length of the shortest sensitizable path in the circuit. In this variant the $\chi^{g,t}$ values are defined to be *true* iff $g$ takes time *at most* $t$ to stabilize, rather than the *at least* $t$ definition used earlier in the chapter. It is also shown that the annotated CNF formula in its present form can be used as part of a deterministic search for the shortest sensitizable path in a

circuit, by searching for the largest $T$ value for which there is no non-satisfying solution to the formula. Furthermore, the combination of the variant $\chi$ values and searching for largest $T$ values which have no associated non-satisfying solutions could provide the basis for a de-randomized solution to the longest sensitizable path problem. This is seen as a key area for future work.

## 8.4 Applications

In chapter 7, the use of long sensitizable path lengths and shortest topological path lengths is discussed for optimal setting of circuit clock speeds, circuit timing optimization through gate resizing, and as a development aid in the creation of test sets for delay faults.

The topological search algorithm is adapted for use in a timing sensitivity algorithm, in which each gate in a circuit is classified as tolerant or intolerant to delay increases, delay decreases, neither, or both. This information is used in targeting gates within the circuit for resizing, with the goal of improving the optimal operating speed of the circuit. A very simple timing optimization routine is implemented, and it is shown that the sensitivity criteria are very effective at identifying key components in the circuit. The implementation of a more sophisticated timing optimization loop, in conjunction with a sensitivity tool based on the CNF satisfiability algorithm of chapter 6, is seen as a key area for future work.

Also discussed in chapter 7 is the use of the timing sensitivity criteria for the efficient development of test sets for delay faults. The criteria give the test developer very clear indications as to which components must be tested for defects causing increased delays, and which components must be tested for defects causing

decreased delays.

## 8.5 Future work

There are a number of interesting areas for future work introduced in this dissertation. These areas fall into five general categories:

- improvements and analysis for the algorithms developed in this dissertation,
- development of the alternative randomized algorithms proposed in this dissertation,
- de-randomization of the algorithms,
- more sophisticated application of the algorithms in circuit optimization and delay fault test set development, and
- the continued improvement of delay modeling techniques.

Of the two algorithms presented for the determination of maximum length sensitizable paths, the CNF satisfiability algorithm is judged to have greater long term potential. With respect to this algorithm, improvements in the variable selection process and improvements in the analysis of the algorithm are the two most immediate issues.

In terms of alternative randomized solutions, in this dissertation there are three suggested possible approaches to developing randomized solutions to the maximum sensitizable path problem:

- propagation of implications,

- path tracing, and
- integer programming.

Each of these approaches is based on the use of the analysis circuit introduced by McGeer[38], and each is worth pursuing further.

With respect to de-randomization of the algorithms, potential fast deterministic solutions to both the longest and shortest sensitizable path problems are suggested by the material presented in section 6 8.

In terms of applications, the author sees the development of a sophisticated timing optimization tool, using the CNF satisfiability approach, as being the next logical step for this work. The development of a set of circuit-to-CNF translation rules which incorporate the destabilization time estimates as well as the sensitization times should create a very effective means of rapidly categorizing the components of a circuit, and the use of this categorization within a detailed timing optimization loop is expected to produce excellent results. The applicability of the sensitivity criteria in the development of delay fault test sets is also readily apparent, and seems a logical target for further development.

Finally, the research and design community must be made aware of the implications of the unit delay model with respect to circuit behaviour. While most researchers accept, on a common sense level, that altering the delay model alters the expectations of behaviour, it is the author's opinion that the severity of the distorting effects of unit delay models is widely underestimated.

# Bibliography

[1] MCNC benchmarks and cell libraries. In *1989 International Workshop on Logic Synthesis.*

[2] J. Beck. An algorithmic approach to the Lovasc local lemma. *Random Structures and Algorithms*, pages pages 343–365, 1991.

[3] J. Benkoski, E. Vanden Meersh, L. Claesen, and H. De Man. Timing verification using statically sensitizable paths. *IEEE Transactions on Computer Aided Design*, volume 9 #10:pages 1073–1084, October 1990.

[4] D. Brand and V. Iyengar. Timing analysis using functional relationships. In *International Conference on Computer Aided Design*, pages 126–129, 1986.

[5] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *1985 International Symposium on Circuits and Systems*, 1985.

[6] R. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler. COSMOS: a compiled simulator for mos circuits. In *Design Automation Conference*, pages pages 9–16, 1987.

[7] V. Chaiyakul, A. Wu, and D. Gajski. Timing models for high level estimation. In *European Design Automation Conference*, pages pages 60–65, 1992.

[8] H. Chang and J. Abraham. CHAN: and efficient critical path analysis algorithm. In *IEEE Proceedings of the European Conference on Design Automation*, pages pages 444–448, 1993.

[9] H. Chang and J. Abraham. Viper - an efficient vigourously sensitizable path extractor. In *Design Automation Conference*, pages pages 112–117, 1993.

[10] H. Chen and D. Du. Path sensitization in critical path problem. In *International Conference on Computer Aided Design*, pages pages 208–211, 1991.

[11] H. Chen, D. Du, and L. Liu. Critical path analysis for performance optimization. In *Design Automation Conference*, pages 547–550, 1991.

[12] S. Cheng, H. Chen, D. Du, and A. Lim. The role of long and short paths in circuit performance optimization. In *Design Automation Conference*, pages pages 542–548, 1992.

[13] S. Cheng, H. Chen, D. Du, and A. Lim. The role of long and short paths in circuit performance optimization. *IEEE Transactions on Computer Aided Design*, volume 13, #7:pages 857–864, July 1994.

[14] S. Devadas, K. Keutzer, and S. Malik. Delay computation in combinational logic circuits: Theory and algorithms. In *International Conference on Computer-Aided Design*, pages 176–179, 1991.

[15] S. Devadas, K. Keutzer, and S. Malik. Computation of floating mode delay in combinational circuits: theory and algorithms. *IEEE Transactions on Computer Aided Design*, volume 12 #12:pages 1913–1923, December 1993.

[16] S. Devadas, K. Keutzer, and S. Malik. Computation of floating mode delay in combinational circuits: practice and implementation. *IEEE Transactions on Computer Aided Design*, volume 12 #12:pages 1924–1935, December 1993.

[17] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Certified timing verification and the transition delay of a logic circuit. In *Design Automation Conference*, pages pages 549–555, 1992.

[18] S. Devedas and K. Keutzer. Synthesis of robust delay-fault-testable circuits: Practice. *IEEE Transactions on Computer Aided Design*, volume 11 #3:pages 277–300, March 1992.

[19] D. Du, S. Yen, and S. Ghanta. On the general false path problem in timing analysis. In *Design Automation Conference*, pages 555–560, 1989.

[20] C. Fang and W. Jone. Timing optimization by gate resizing and critical path optimization. *IEEE Transactions on Computer Aided Design*, volume 14 #2:pages 210-217, February 1995.

[21] C. Glover and M. Mercer. A method of delay fault test generation. In *Design Automation Conference*, pages 90–95, 1988.

[22] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, volume 30, #3:pages 215–222, March 1981.

[23] L. H. Harper. Optimal assignment of numbers to vertices. *Journal Society of Industrial and Applied Mathematics*, volume 12:pages 131–135, March 1964.

[24] L. H. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, volume 1:385–393, 1966.

[25] S. Huang, T. Parng, and J. Shyu. A new approach to solving false path problem in timing analysis. In *International Conference on Computer-Aided Design*, pages 216-219, 1991.

[26] S. Huang, T. Parng, and J. Shyu. A polynomial time heuristic approach to approximate a solution to the false path problem. In *Design Automation Conference*, pages pages 118–127, 1993.

[27] Y. Ju and R. Saleh. Incremental techniques for the identification of statically sensitizable critical paths. In *Design Automation Conference*, pages 541–546, 1991.

[28] R. Karp, M. Luby, and N. Madras. Monte carlo approximation algorithms for enumeration problems. *Journal of algorithms*, vol. 10:pages 429 – 448, 1989.

[29] D. Knuth. *The Art of Computer Programming*, volume vol. I. Addison-Wesley Publisher Company, Reading Massechusetts, 1973.

[30] W. Lam, R. Brayton, and A. Sangiovanni-Vincentelli. Circuit delay models and their exact computation using timed boolean functions. In *Design Automation Conference*, pages pages 128–134, 1993.

[31] T. Larrabee. Efficient generation of test patterns using boolean difference. In *International Test Conference*, pages pages 795–801, 1989.

[32] W. Li. Strongly np-hard discrete gate-sizing problems. *IEEE Transactions on Computer Aided Design*, volume 13, #8 pages 1045–1050, August 1994.

[33] L. Liu, D. Du, and H. Chen. An efficient parallel critical path algorithm. In *Design Automation Conference*, pages pages 535–540, 1991.

[34] L. Liu, D. Du, and H. Chen. An efficient parallel critical path algorithm. *IEEE Transactions on Computer Aided Design*, volume 13 #7:pages 909 – 919, July 1994.

[35] R. Llopis, L. Xirgo, and J. Bordell. Short destabilizing paths in timing verification. In *International Conference on Circuit Design*, 1994.

[36] P. Maurer. Two new techniques for unit-delay compiled simulation. *IEEE Transactions on Computer Aided Design*, volume 11 #9:pages 1120–1130, September 1992.

[37] P. McGeer and R. Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *Design Automation Conference*, pages 561–567, 1989.

[38] P. McGeer and R. Brayton. *Integrating Functional and Temporal Domains in Logic Design.* Kluwer Academic Publishers, Boston, 1991.

[39] P. McGeer, R. Brayton, A. Sangiovanni-Vincentelli, and S. Sahni. Performance enhancement through the generalized bypass transform. In *International Conference on Computer Aided Design*, pages 184–187, 1991.

[40] P. McGeer, A. Saldanha, P. Stephan, and R. Brayton. Timing analysis and delay-fault test generation using path-recursive functions. In *International Conference on Computer Aided Design*, pages 180–183, 1991.

[41] R. Motwani and P. Raghavan. *Randomized Algorithms.* 1993. preliminary draft.

[42] E. Park and M. Mercer. Robust and nonrobust tests for path delay faults in a combinational circuit. In *International Test Conference*, pages 1027–1034, 1987.

[43] S. Perremans, L. Claesen, and H. De Man. Static timing analysis of dynamically sensitizable paths. In *Design Automation Conference*, pages 568–573, 1989.

[44] C. Ramachandran and F. Kurdahi. Combined topological and functionality-based delay estimation using a layout-driven approach for high level applica-

tions. *IEEE Transactions on Computer Aided Design*, volume 13 #12:pages 1450–1460, December 1994.

[45] S. Reddy, C. Lin, and S. Patil. An automatic test pattern generator for the detection of path delay faults. In *International Conference on Computer Aided Design*, pages 284–287, 1987.

[46] J. Silva, K. Sakallah, and L. Vidigal. FPD - an environment for exact timing analysis. In *International Conference on Computer Aided Design*, pages 212–215, 1991.

[47] S. Silva and K. Sakallah. Concurrent path sensitization in timing analysis. In *European Design Automation Conference*, pages pages 196–199, 1993.

[48] G. Smith. Model for delay faults based upon paths. In *International Test Conference*, pages 342–349, 1985.

[49] R. Stewart and J. Benkoski. Static timing analysis using interval constraints. In *International Conference on Computer Aided Design*, pages 308–311, 1991.

[50] J. Wakerly. *Digital Design: principles and practices.* Prentice Hall, Inc., Englewood Cliffs, N.J., 1994.

[51] D. Walker. Tolerance of delay faults. In *The IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 207–216, 1992.

[52] D. Wallace and M. Chandrasekhar. High-level delay estimation for technology-independent logic equations. In *International Conference on Computer Aided Design*, pages 188–191, 1990.

[53] D. Wessels and J. Muzio. Probabilistic identification of critical components for circuit delays. In *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 215–222, 1993.

# Appendix 1

# Benchmark circuits

Table 1.1: ISCAS circuit notes

| | | |
|---|---|---|
| Circuits with no long false paths | | |
| C432 | Priority encoder | |
| C499 | ECAT | |
| C880 | ALU & control | |
| C1355 | ECAT | same as C499 but replace XORs with 4-NANDS |
| Circuits with long false paths | | |
| C1908 | ALU & control | |
| C2670 | ALU & control | direct paths input-to-output<br>contains redundant inputs with no fanout |
| C5315 | ALU & select | direct paths input-to-output |
| C6288 | 16 by 16 multiplier | > 46000 long false paths<br>also contains several stuck-at-fault redundancies |
| C7552 | ALU & control | direct paths input-to-output<br>contains redundant inputs with no fanout |

This appendix contains a list of the characteristics of the benchmark circuits used in this dissertation. First, detailed information about the 9 ISCAS [5] benchmark circuits is included, then a brief summary of all 64 circuits (the ISCAS plus MCNC [1] benchmarks) is provided. The second set of tables provide some characteristics of the circuits in the ISCAS benchmark set, commonly used to compare delay algorithm behaviour. The path and delay values given here assume a unit delay model with no fanout cost, and is drawn from [5] [13] and [37].

Table 1.2: List of benchmark circuits

| circuit | inputs | outputs | gates | lines |
|---|---|---|---|---|
| b1 | 3 | 4 | 15 | 31 |
| cm42a | 4 | 10 | 29 | 61 |
| majority | 5 | 1 | 11 | 21 |
| C17 | 5 | 2 | 13 | 21 |
| decod | 5 | 16 | 25 | 91 |
| cm82a | 5 | 3 | 31 | 54 |
| cm138a | 6 | 8 | 22 | 56 |
| z4ml | 7 | 4 | 70 | 134 |
| f51m | 8 | 8 | 143 | 285 |
| ldd | 9 | 19 | 103 | 264 |
| 9symml | 9 | 1 | 168 | 403 |
| x2 | 10 | 7 | 52 | 118 |
| alu2 | 10 | 6 | 360 | 753 |
| cm152a | 11 | 1 | 33 | 55 |
| cm85a | 11 | 3 | 42 | 85 |
| cm151a | 12 | 2 | 33 | 64 |
| cm162a | 14 | 5 | 54 | 98 |
| cu | 14 | 11 | 58 | 136 |
| alu4 | 14 | 8 | 720 | 1449 |
| pm1 | 16 | 13 | 53 | 120 |
| cmb | 16 | 4 | 54 | 106 |
| cm163a | 16 | 5 | 55 | 97 |
| parity | 16 | 1 | 76 | 122 |
| vda | 17 | 39 | 926 | 2265 |
| tcon | 17 | 16 | 72 | 120 |
| pcle | 19 | 9 | 86 | 150 |
| sct | 19 | 15 | 119 | 260 |
| mux | 21 | 1 | 90 | 177 |
| cc | 21 | 20 | 59 | 145 |
| cm150a | 21 | 1 | 70 | 132 |

| circuit | inputs | outputs | gates | lines |
|---|---|---|---|---|
| tt2 | 24 | 21 | 215 | 507 |
| lal | 26 | 19 | 149 | 321 |
| pcler8 | 27 | 17 | 103 | 191 |
| frg1 | 28 | 3 | 108 | 228 |
| c8 | 28 | 18 | 189 | 390 |
| comp | 32 | 3 | 176 | 309 |
| C6288 | 32 | 32 | 2400 | 4784 |
| my_adder | 33 | 17 | 224 | 401 |
| C1908 | 33 | 25 | 718 | 1369 |
| term1 | 34 | 10 | 391 | 879 |
| count | 35 | 16 | 144 | 287 |
| C432 | 36 | 7 | 209 | 428 |
| unreg | 36 | 16 | 113 | 241 |
| too_large | 38 | 3 | 748 | 1746 |
| b9 | 41 | 21 | 123 | 263 |
| C499 | 41 | 32 | 579 | 1025 |
| C1355 | 41 | 32 | 619 | 1169 |
| k2 | 45 | 45 | 1975 | 4724 |
| cht | 47 | 36 | 231 | 470 |
| apex7 | 49 | 37 | 269 | 539 |
| C3540 | 50 | 22 | 1161 | 2448 |
| x1 | 51 | 35 | 356 | 796 |
| C880 | 60 | 26 | 394 | 766 |
| example2 | 85 | 66 | 366 | 708 |
| x4 | 94 | 71 | 443 | 1049 |
| apex6 | 135 | 99 | 831 | 1596 |
| x3 | 135 | 99 | 763 | 1875 |
| rot | 135 | 107 | 689 | 1423 |
| frg2 | 143 | 139 | 1137 | |
| pair | 173 | 137 | 1674 | 3407 |
| C5315 | 178 | 123 | 1938 | 4138 |
| C7552 | 207 | 108 | 3087 | 5823 |
| C2670 | 233 | 140 | 1100 | 2116 |
| des | 256 | 245 | 4679 | 10315 |

# Appendix 2

# Delay model comparison data

This appendix contains the raw data used to derive the results in chapter 3. The data is provided in 6 tables, each of which is summarized below.

1. Table 2.1 contains the length of the longest topological path for each circuit under each of the six delay models used in chapter 3.
2. Table 2.2 contains the length of the longest sensitizable path for each circuit under each delay model.
3. Table 2.3 contains the number of vectors which sensitize longest paths in each circuit under each delay model.
4. Table 2.4 contains the number of cubes the sensitizing vectors (of table 2.3) are divided into.
5. Table 2.5 gives the number of unidirectional errors obtained comparing the delays of adjacent vectors, where an error under a model indicates that either (a) under the DE model both vectors have the same delay, but under the comparison model the vectors have different delays, or (b) under the

DE model the two vectors have different delays but under the comparison model the two vectors have the same delay. The DE column in this table indicates the total number of adjacencies compared (one comparison per primary output for each of the $m2^{m-1}$ pairs).

6. Table 2.6 gives the number of bidirectional errors obtained comparing the delays of adjacent vectors, where an error under a model indicates that for a pair of vectors, $x$ and $y$, either (a) under the DE model the delay of $x$ is greater than the delay of $y$ but under the model of comparison the delay of $x$ is less than the delay of $y$, or (b) under the DE model the delay of $x$ is less than the delay of $y$ but under the model of comparison the delay of $x$ is greater than the delay of $y$. The DE column in this table indicates the total number of adjacencies compared (one comparison per primary output for each of the $m2^{m-1}$ pairs).

Table 2.1: Longest topological path in a circuit under each model

| circuit | inputs | DD | DE | GD | ID | SE | UE |
|---|---|---|---|---|---|---|---|
| b1 | 3 | 6.38 | 5.97 | 6.90 | 6.49 | 14.48 | 5.20 |
| cm42a | 4 | 6.93 | 6.59 | 7.01 | 6.67 | 19.79 | 6.80 |
| majority | 5 | 7.73 | 6.97 | 8.03 | 7.27 | 19.48 | 7.40 |
| C17 | 5 | 6.32 | 5.46 | 6.32 | 5.46 | 18.66 | 6.60 |
| decod | 5 | 8.51 | 8.51 | 9.03 | 9.03 | 21.22 | 6.40 |
| cm82a | 5 | 9.59 | 8.61 | 9.59 | 8.61 | 28.96 | 10.40 |
| cm138a | 6 | 9.46 | 9.46 | 9.68 | 9.68 | 23.16 | 7.40 |
| z4ml | 7 | 12.74 | 12.21 | 12.89 | 12.36 | 35.40 | 12.20 |
| f51m | 8 | 19.24 | 17.36 | 19.24 | 17.36 | 54.37 | 18.20 |
| ldd | 9 | 18.61 | 17.02 | 19.64 | 18.05 | 40.20 | 13.40 |
| 9symml | 9 | 25.73 | 24.22 | 26.61 | 25.04 | 60.81 | 20.00 |
| x2 | 10 | 11.13 | 10.61 | 11.54 | 11.18 | 28.47 | 9.80 |
| alu2 | 10 | 67.75 | 58.91 | 69.95 | 60.89 | 177.51 | 56.60 |
| cm152a | 11 | 9.22 | 8.41 | 9.22 | 8.41 | 26.72 | 10.00 |
| cm85a | 11 | 15.79 | 14.68 | 16.15 | 15.02 | 44.56 | 15.80 |
| cm151a | 12 | 14.13 | 13.30 | 14.80 | 13.91 | 37.02 | 13.80 |
| cm162a | 14 | 13.49 | 13.02 | 14.12 | 13.33 | 35.70 | 12.20 |
| cu | 14 | 11.85 | 11.40 | 12.26 | 11.81 | 28.46 | 9.20 |
| alu4 | 14 | 72.85 | 64.62 | 74.76 | 66.53 | 200.96 | 62.20 |
| pm1 | 16 | 10.31 | 9.87 | 10.69 | 10.25 | 25.41 | 7.80 |
| cmb | 16 | 13.98 | 12.16 | 14.34 | 12.52 | 35.09 | 12.80 |
| cm163a | 16 | 12.53 | 11.65 | 12.93 | 12.05 | 34.27 | 12.00 |
| parity | 16 | 13.59 | 13.32 | 13.59 | 13.32 | 43.14 | 16.20 |
| vda | 17 | 45.96 | 42.77 | 47.13 |  | 110.84 | 27.59 |
| tcon | 17 | 4.29 | 4.11 | 4.29 | 4.11 | 12.24 | 4.80 |
| pcle | 19 | 22.03 | 20.80 | 22.03 | 20.80 | 69.64 | 24.20 |
| sct | 19 | 14.69 | 13.94 | 15.10 | 14.32 | 36.83 | 11.80 |
| mux | 21 | 26.35 | 23.09 | 27.12 | 23.76 | 67.54 | 21.20 |
| cc | 21 | 13.45 | 13.33 | 14.21 | 14.09 | 33.58 | 8.60 |
| cm150a | 21 | 17.85 | 16.18 | 18.30 | 16.54 | 49.26 | 18.60 |

Table 2.2: Longest sensitizable path in circuit under each model

| circuit | inputs | DD | DE | GD | ID | SE | UE |
|---|---|---|---|---|---|---|---|
| b1 | 3 | 6.18 | 5.77 | 6.90 | 6.49 | 14.48 | 5.20 |
| cm42a | 4 | 6.72 | 6.22 | 7.01 | 6.67 | 19.79 | 6.80 |
| majority | 5 | 7.12 | 6.42 | 8.03 | 7.27 | 19.48 | 7.40 |
| C17 | 5 | 6.12 | 5.33 | 6.32 | 5.46 | 18.66 | 6.60 |
| decod | 5 | 8.51 | 8.51 | 9.03 | 9.03 | 21.22 | 6.40 |
| cm82a | 5 | 8.98 | 8.00 | 9.59 | 8.61 | 28.96 | 10.40 |
| cm138a | 6 | 9.46 | 9.46 | 9.68 | 9.68 | 23.16 | 7.40 |
| z4ml | 7 | 12.14 | 11.69 | 12.89 | 12.36 | 35.40 | 12.20 |
| f51m | 8 | 18.44 | 16.64 | 19.24 | 17.36 | 54.37 | 18.20 |
| ldd | 9 | 18.02 | 16.43 | 19.64 | 18.05 | 40.20 | 13.40 |
| 9symml | 9 | 23.12 | 21.65 | 26.61 | 25.04 | 60.81 | 20.00 |
| x2 | 10 | 10.19 | 9.68 | 11.54 | 11.18 | 28.47 | 9.80 |
| alu2 | 10 | 52.79 | 46.07 | 57.81 | 51.13 | 149.86 | 46.97 |
| cm152a | 11 | 8.41 | 7.68 | 9.22 | 8.41 | 26.72 | 10.00 |
| cm85a | 11 | 14.75 | 14.15 | 16.15 | 15.02 | 44.56 | 15.80 |
| cm151a | 12 | 12.68 | 12.15 | 13.93 | 13.40 | 37.02 | 13.80 |
| cm162a | 14 | 13.09 | 12.22 | 14.12 | 13.33 | 35.70 | 12.20 |
| cu | 14 | 11.65 | 11.20 | 12.26 | 11.81 | 28.46 | 9.20 |
| alu4 | 14 | 65.21 | 57.46 | 71.06 | 62.49 | 185.86 | 57.99 |
| pm1 | 16 | 9.57 | 9.16 | 10.69 | 10.25 | 25.41 | 7.80 |
| cmb | 16 | 13.38 | 11.59 | 14.34 | 12.52 | 35.09 | 12.80 |
| cm163a | 16 | 12.13 | 11.25 | 12.93 | 12.05 | 34.27 | 12.00 |
| parity | 16 | 12.55 | 12.30 | 13.59 | 13.32 | 43.14 | 16.20 |
| vda | 17 | 40.11 | 37.34 | 47.13 | | 110.84 | 27.59 |
| tcon | 17 | 4.09 | 3.80 | 4.29 | 4.11 | 12.24 | 4.80 |
| pcle | 19 | 21.42 | 20.27 | 22.03 | 20.80 | 69.64 | 24.20 |
| sct | 19 | 14.49 | 13.43 | 15.10 | 14.32 | 36.83 | 11.80 |
| mux | 21 | 23.77 | 20.90 | 27.12 | 23.76 | 67.54 | 21.20 |
| cc | 21 | 12.63 | 12.51 | 14.21 | 14.09 | 33.58 | 8.60 |
| cm150a | 21 | 15.49 | 14.45 | 16.56 | 15.52 | 49.26 | 18.60 |

Table 2.3: Number of vectors sensitizing longest paths

| circuit | inputs | DD | DE | GD | ID | SE | UE |
|---|---|---|---|---|---|---|---|
| b1 | 3 | 1 | 1 | 2 | 1 | 2 | 2 |
| cm42a | 4 | 4 | 4 | 8 | 4 | 8 | 8 |
| majority | 5 | 1 | 1 | 3 | 2 | 6 | 6 |
| C17 | 5 | 4 | 2 | 8 | 6 | 8 | 8 |
| decod | 5 | 16 | 16 | 32 | 32 | 32 | 32 |
| cm82a | 5 | 3 | 1 | 9 | 2 | 9 | 9 |
| cm138a | 6 | 8 | 8 | 16 | 16 | 16 | 16 |
| z4ml | 7 | 2 | 2 | 6 | 4 | 12 | 12 |
| f51m | 8 | 1 | 1 | 2 | 2 | 4 | 4 |
| ldd | 9 | 32 | 32 | 64 | 64 | 126 | 24 |
| 9symml | 9 | 2 | 1 | 3 | 3 | 3 | 3 |
| x2 | 10 | 8 | 8 | 22 | 16 | 184 | 16 |
| alu2 | 10 | 3 | 1 | 2 | 1 | 2 | 2 |
| cm152a | 11 | 512 | 128 | 768 | 192 | 768 | 768 |
| cm85a | 11 | 16 | 16 | 64 | 48 | 32 | 32 |
| cm151a | 12 | 32 | 32 | 32 | 32 | 64 | 64 |
| cm162a | 14 | 48 | 48 | 112 | 480 | 960 | 784 |
| cu | 14 | 64 | 64 | 128 | 128 | 2048 | 1024 |
| alu4 | 14 | 2 | 2 | 3 | 2 | 6 | 6 |
| pm1 | 16 | 256 | 256 | 512 | 512 | 4096 | 1024 |
| cmb | 16 | 16 | 16 | 192 | 192 | 208 | 208 |
| cm163a | 16 | 256 | 256 | 512 | 512 | 3584 | 3584 |
| parity | 16 | 12288 | 2048 | 24576 | 4096 | 24576 | 24576 |
| vda | 17 | 2048 | 2048 | 4096 |  | 2048 | 2048 |
| tcon | 17 | 58975 | 65280 | 124255 | 65535 | 131072 | 131072 |
| pcle | 19 | 256 | 256 | 1024 | 512 | 1024 | 1024 |
| sct | 19 | 5120 | 5120 | 20480 | 1280 | 1280 | 1280 |
| mux | 21 | 1024 | 1024 | 4096 | 1024 | 4096 | 4096 |
| cc | 21 | 117464 | 117464 | 248024 | 248024 | 248024 | 248024 |
| cm150a | 21 | 8192 | 8192 | 16384 | 8192 | 16384 | 16384 |

Table 2.4: Number of cubes of vectors sensitizing longest paths

| circuit | inputs | DD | DE | GD | ID | SE | UE |
|---|---|---|---|---|---|---|---|
| b1 | 3 | 1 | 1 | 2 | 1 | 2 | 2 |
| cm42a | 4 | 1 | 1 | 1 | 1 | 1 | 1 |
| majority | 5 | 1 | 1 | 2 | 1 | 4 | 4 |
| C17 | 5 | 1 | 1 | 1 | 1 | 1 | 1 |
| decod | 5 | 1 | 1 | 1 | 1 | 1 | 1 |
| cm82a | 5 | 1 | 1 | 2 | 1 | 2 | 2 |
| cm138a | 6 | 1 | 1 | 1 | 1 | 1 | 1 |
| z4ml | 7 | 2 | 2 | 4 | 2 | 8 | 8 |
| f51m | 8 | 1 | 1 | 1 | 1 | 1 | 1 |
| ldd | 9 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9symml | 9 | 2 | 1 | 2 | 2 | 2 | 2 |
| x2 | 10 | 1 | 1 | 1 | 1 | 2 | 1 |
| alu2 | 10 | 1 | 1 | 1 | 1 | 1 | 1 |
| cm152a | 11 | 4 | 1 | 4 | 1 | 4 | 4 |
| cm85a | 11 | 8 | 8 | 4 | 4 | 8 | 8 |
| cm151a | 12 | 1 | 1 | 1 | 1 | 1 | 1 |
| cm162a | 14 | 1 | 1 | 1 | 1 | 5 | 3 |
| cu | 14 | 1 | 1 | 1 | 1 | 1 | 1 |
| alu4 | 14 | 2 | 2 | 1 | 1 | 2 | 2 |
| pm1 | 16 | 1 | 1 | 1 | 1 | 1 | 1 |
| cmb | 16 | 1 | 1 | 1 | 1 | 1 | 1 |
| cm163a | 16 | 1 | 1 | 1 | 1 | 3 | 3 |
| parity | 16 | 12288 | 2048 | 2048 | 2048 | 2048 | 2048 |
| vda | 17 | 1 | 1 | 1 |  | 1 | 1 |
| tcon | 17 | 8 | 1 | 8 | 1 | 1 | 1 |
| pcle | 19 | 1 | 1 | 1 | 1 | 1 | 1 |
| sct | 19 | 1 | 1 | 1 | 1 | 1 | 1 |
| mux | 21 | 1 | 1 | 2 | 1 | 2 | 2 |
| cc | 21 | 1 | 1 | 1 | 1 | 1 | 1 |
| cm150a | 21 | 1 | 1 | 2 | 1 | 1 | 1 |

Table 2.5: Number of unidirectional errors under each model (compared to DE)

| circuit | inputs | DD | (DE) | GD | ID | SE | UE |
|---|---|---|---|---|---|---|---|
| b1 | 3 | 1 | 48 | 13 | 11 | 13 | 13 |
| cm42a | 4 | 16 | 320 | 30 | 6 | 40 | 30 |
| majority | 5 | 4 | 80 | 8 | 5 | 8 | 8 |
| C17 | 5 | 0 | 160 | 14 | 6 | 14 | 14 |
| decod | 5 | 168 | 1280 | 184 | 16 | 152 | 152 |
| cm82a | 5 | 10 | 240 | 38 | 21 | 38 | 38 |
| cm138a | 6 | 184 | 1536 | 192 | 8 | 160 | 160 |
| z4ml | 7 | 38 | 1792 | 188 | 114 | 236 | 236 |
| f51m | 8 | 156 | 8192 | 750 | 620 | 948 | 1006 |
| ldd | 9 | 1814 | 43776 | 2810 | 904 | 4398 | 4666 |
| 9symml | 9 | 118 | 2304 | 154 | 76 | 207 | 203 |
| x2 | 10 | 1468 | 35840 | 2559 | 901 | 2577 | 3093 |
| alu2 | 10 | 366 | 30720 | 1231 | 1145 | 1343 | 1428 |
| cm152a | 11 | 768 | 11264 | 1920 | 352 | 1920 | 1920 |
| cm85a | 11 | 240 | 33792 | 524 | 388 | 524 | 524 |
| cm151a | 12 | 320 | 49152 | 880 | 164 | 2496 | 2496 |
| cm162a | 14 | 22400 | 573440 | 30032 | 11952 | 44496 | 32848 |
| cu | 14 | 44544 | 1261568 | 47168 | 15008 | 55760 | 51468 |
| alu4 | 14 | 8286 | 917504 | 21331 | 21116 | 21607 | 25450 |
| pm1 | 16 | 155648 | 6815744 | 333184 | 212352 | 360064 | 390016 |
| cmb | 16 | 97872 | 2097152 | 122026 | 16048 | 107056 | 105712 |
| cm163a | 16 | 110080 | 2621440 | 156160 | 66048 | 194048 | 175104 |
| parity | 16 | 0 | 524288 | 159024 | 56964 | 159024 | 159024 |
| vda | 17 | 1224494 | 43450368 | 1801062 | 805656 | 1860670 | 2383170 |
| tcon | 17 | 393216 | 17825792 | 1486638 | 917504 | 1572864 | 1572864 |
| pcle | 19 | 1216256 | 44826624 | 1416448 | 1109760 | 1924352 | 2088192 |
| sct | 19 | 1144576 | 74711040 | 3182784 | 184352 | 3086656 | 3488064 |
| mux | 21 | 279837 | 22020096 | 410917 | 257219 | 512062 | 685025 |
| cc | 21 | 1081344 | 440401920 | 13860864 | 11960320 | 19365888 | 13271040 |
| cm150a | 21 | 90112 | 22020096 | 235520 | 39424 | 761856 | 761856 |

Table 2.6: Number of bidirectional errors under each model (compared to DE)

| circuit | inputs | DD | (DE) | GD | ID | SE | UE |
|---|---|---|---|---|---|---|---|
| b1 | 3 | 2 | 48 | 0 | 0 | 0 | 0 |
| cm42a | 4 | 0 | 320 | 0 | 4 | 0 | 0 |
| majority | 5 | 2 | 80 | 1 | 0 | 0 | 0 |
| C17 | 5 | 8 | 160 | 4 | 0 | 0 | 0 |
| decod | 5 | 0 | 1280 | 0 | 0 | 0 | 0 |
| cm82a | 5 | 19 | 240 | 12 | 6 | 12 | 12 |
| cm138a | 6 | 0 | 1536 | 0 | 0 | 0 | 0 |
| z4ml | 7 | 80 | 1792 | 90 | 38 | 58 | 58 |
| f51m | 8 | 207 | 8192 | 334 | 274 | 253 | 305 |
| ldd | 9 | 1100 | 43776 | 1580 | 1295 | 888 | 1254 |
| 9symml | 9 | 116 | 2304 | 130 | 63 | 128 | 164 |
| x2 | 10 | 310 | 35840 | 234 | 260 | 320 | 629 |
| alu2 | 10 | 217 | 30720 | 576 | 452 | 474 | 809 |
| cm152a | 11 | 656 | 11264 | 272 | 0 | 272 | 272 |
| cm85a | 11 | 200 | 33793 | 112 | 112 | 248 | 248 |
| cm151a | 12 | 464 | 49152 | 268 | 280 | 340 | 340 |
| cm162a | 14 | 128 | 573440 | 88 | 552 | 80 | 920 |
| cu | 14 | 48 | 1261568 | 2480 | 2384 | 2176 | 3480 |
| alu4 | 14 | 8526 | 917504 | 19817 | 17166 | 19621 | 28026 |
| pm1 | 16 | 0 | 6815744 | 512 | 512 | 0 | 0 |
| cmb | 16 | 32 | 2097152 | 3152 | 80 | 64 | 64 |
| cm163a | 16 | 14336 | 2621440 | 17408 | 64512 | 19546 | 21504 |
| parity | 16 | 52660 | 524288 | 528 | 1624 | 528 | 528 |
| vda | 17 | 500418 | 43450368 | 572328 | 474548 | 591208 | 1021412 |
| tcon | 17 | 131072 | 17825792 | 262144 | 0 | 0 | 0 |
| pcle | 19 | 12288 | 44826624 | 4096 | 0 | 110592 | 176128 |
| sct | 19 | 265216 | 74711040 | 415232 | 209216 | 335104 | 372224 |
| mux | 21 | 43158 | 22020096 | 109141 | 96866 | 11624 | 289378 |
| cc | 21 | 0 | 440401920 | 65536 | 196608 | 1835008 | 262144 |
| cm150a | 21 | 111104 | 22020096 | 195584 | 163840 | 162304 | 162304 |

# Appendix 3

# Software development, testing techniques

All algorithms implemented in this dissertation have been written in C++, and close attention has been paid to the design and test of the class structures used.

Discrete data types have been created for use in each of the algorithms, and testing/verification has taken place at several levels:

- Test drivers have been written to exercise the routines associated with each class of objects on two levels: one to test the class objects as distinct entities, the second to test the interaction of the class objects.
- Verification at an algorithmic level has also taken two forms: in one the operation of the algorithms on small circuits (15-20 gates) has been verified by hand, in the second the operation of the algorithms on large circuits (up to 10300 gates) has been verified against previously published results, identifying the length of longest topological paths and longest statically sensitizable paths in the circuits.

# Appendix 4

# Strongly connected components

The routines used to detect all the strongly connected components in the implication graph are based on the algorithms provided below.

```
int num_nodes, num_stacked;
graph_node *Graph[MAXNODES], *stack[MAXNODES];

void formula::search_scc()
// finds all scc's in graph, uses find_zero_dfi() and DFSSC() to do it
{  graph_node *next_node;
   int i, j;
   num_stacked = 0;
   for (j=0; j<num_nodes; j++) {
       (Graph[j])->set_stacked(FALSE);
       (Graph[j])->set_depth_index(0);
   }
   i = 1;
   next_node = find_zero_dfi();
   while (next_node != NULL) {
         DFSSC(next_node, &i);
         next_node = find_zero_dfi();
   }
}

graph_node *formula::find_zero_dfi()
{  graph_node *next_node;
   int i = 0;
```

```
   next_node = Graph[i];
   while ((i < num_nodes) && (next_node->get_depth_index() != 0)) {
         next_node = Graph[i++];
   }
   if (i >= num_nodes) return(NULL);
   else return(next_node);
}

void formula::DFSSC(graph_node *V, int *i)
{  int imp, j, rv, ru;
   graph_node *U;
   V->set_depth_index(*i);
   V->set_root_index(*i);
   (*i) += 1;
   stack[num_stacked++] = V;
   V->set_stacked(TRUE);
   imps = V->get_num_imps();
   for (j=0; j<imps; j++) {
       U = get_imp(j);
       if (U->get_depth_index() == 0) {
          DFSSC(U, i);
          rv = V->get_root_index();
          ru = U->get_root_index();
          if (ru < rv) V->set_root_index(ru);
       } else {
          if ((U->get_depth_index() < V->get_depth_index())
              && (U->get_stacked() == TRUE)) {
              rv = V->get_root_index();
              ru = U->get_root_index();
              if (ru < rv) V->set_root_index(ru);
          }
       }
   }
   if (V->get_root_index() == v->get_depth_index()) {
      // everything on stack up to and including V is in SCC with V
      while (stack[num_stacked - 1] != V) {
         U = stack[num_stacked - 1];
         num_stacked--;
         U->set_stacked(FALSE);

         // now need to rename all occurances of U by V
         // *everywhere*, in graph AND formula,
```

```
            // and simplify accordingly
        }
    }
}
```

# Curriculum Vitae

**Name** : David M. Wessels

**Place of Birth** : Victoria, British Columbia, Canada

**Educational Institutions Attended :** **Degrees Awarded**

University of Victoria (Dept. of Computer Science)

| | |
|---|---|
| Ph.D. Program, Jan. 1991 through 1995 | expected completion May 1995 |
| M.Sc. Program, May 1989 to Aug. 1990 | M.Sc., Nov. 1990 |
| B.Sc. Program, Sept. 1984 to Apr. 1989 | B.Sc., May 1989 |

**Awards and Scholarships :**

Presidents Entrance Scholarship, University of Victoria
Graduate Teaching Fellowship, Annually 1989 - 1995

**Related Training :**

Seminar: Education in Engineering and the Sciences, May 1993

**Committees etc. :**

Computer Science Acting Chair Search Committee, April - May 1995.
Computer Science Chair Search Committee, October 1994 - February 1995.
Graduate Student Social Representative, September 1993 - August 1994.

## Doctoral Research

**Randomized algorithms for VLSI design and test:** In my Doctoral dissertation [14] and several related research publications [3][12][13], I have examined and implemented (in C++) a number of randomized algorithms which rapidly approximate the maximum length sensitizable path within a combinational logic circuit. Analysis of data obtained using these algorithms has also led to a number of interesting observations about circuit delay models currently in general use [8], and about the nature of "practical" circuits [11][14]. Certain combinatoric problems encountered during this research have, as an aside, led to observations about the relationship between labelings of hypercubes and the operation of parallel networks[9][10].

## Masters Research

**Circuit partitioning and testing:** In my thesis [5] and subsequent research papers [1][4][6] I address the problem of reducing the costs normally associated with concurrent testing schemes. Cost reduction is accomplished by first partitioning the circuit into unate subcircuits, and then applying concurrent checking schemes which take advantage of that unateness. This technique allows detection of all single faults within the system, while reducing costs by permitting blocks of circuitry to be shared for use in both code and data generation.

## Other Research Interests

**Hypercube-related combinatorics:** I have recently shown [9][10] that several works from the mid-nineteen sixties on hypercube labelings have relevance in a variety of other areas in computer science, most notably the scheduling of non-parallel events in parallel architectures, the evaluation of the randomness of test pattern generators, and the modeling of delay systems.

**Multiple-valued logic systems:** These systems have long been used in fault detection and location schemes. In past work [2][5], I have shown that a number of concurrent fault detection schemes for binary circuits can be generalized for use in multiple valued circuits. I am also working on expanding the circuit analysis tools of [14] for use with multiple-valued logic systems.

**Computational complexity theory:** Significant advances have been made recently in the areas of randomized algorithms and fixed parameter complexity. However, the lack of comprehensive accessible works surveying the new results *and their relevance to common problems in other fields* prevents more widespread use of the new techniques. To help alleviate this problem, I would like to compile an up-to-date summary of the latest results, their relevance to existing problems (particularly in the VLSI area), and proven transformations between known problems.

**Switching theory:** The search for potential means of classifying the set of all possible logic functions is a challenging problem in the area of switching theory. It is clear that the set of "practical" circuits does not represent a true random sampling over all possible logic functions. Any separation of the set of potential functions into "useful" subsets could have significant impact in the realms of circuit design and test. Some of my recent investigations into circuit delay modeling [8] may shed new light on this problem.

## Publication History

**Fully refereed journal publications :**

(1) "Concurrent Checking of PLA Primary Inputs and Applications in Multiple-Component Circuits", by D. Wessels and J. Muzio, in *Journal of Semicustom ICs*, December, 1990, vol. 8-2, pp. 16-28.

**Papers published in full in refereed conference proceedings :**

(2) "Concurrent Checking and Unidirectional Errors in Multiple-Valued PLAs", by
D. Wessels and J. Muzio, in *IEEE International Symposium on Multiple-Valued Logic* , 1992, pp. 166-173.

(3) "Probabilistic Identification of Critical Components for Circuit Delays", by D. Wessels and J. Muzio, in *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, 1993, pp. 215 - 222.

(4) "PLA Decomposition to Reduce the Cost of Concurrent Checking", by D. Wessels and J. Muzio, in *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, 1992, pp. 117 - 126.

**Other Publications :**

(5) "The Cost and Fault Coverage of Unidirectional Error Detecting Codes for Concurrent Checking in PLAs", by D. Wessels, *Master's Thesis*, at the University of Victoria, July 1990.

(6) "Decompositions of PLAs and Applications to Concurrent Checking", by D. Wessels and J. Muzio, in *Fifth Technical Workshop: New Directions for Testing*, 1991.

(7) "Adding Primary Input Error Coverage to Concurrent Checking for PLAs", by D. Wessels and J. Muzio, in *Fourth Technical Workshop: New Directions for IC Testing*, 1989, pp. 135-153.

**Current journal submissions :**

(8) "The Dangers of Simplistic Delay Models", by D. Wessels and J. Muzio, submitted to *Journal of Electronic Testing: Theory and Applications.*

(9) "Optimal ordering of non-parallel events in cube-connected networks", by D. Wessels and J. Muzio, submitted to *IEEE Transactions on Parallel*

*and Distributed Systems.*

(10) "Applications of hypercube labeling", by D. Wessels and J. Muzio, submitted to *IEEE Transactions on Computers.*

**Articles in preparation :**

(11) "Black box modeling of circuit delay behaviour", by D. Wessels and J. Muzio, *in preparation.*

(12) "Randomized input space searches for maximum circuit delay", by D. Wessels and J. Muzio, *in preparation.*

(13) "Fast probabilistic solutions to CNF satisfiability, as applied to the circuit delay problem", by D. Wessels and J. Muzio, *in preparation*, at the University of Victoria.

(14) "Randomized Algorithms in Path Sensitization for Circuit Optimization and Delay Fault Tolerance", by D. Wessels, *Doctoral dissertation; in preparation*, at the University of Victoria.

(15) "Identification of components for timing optimization", by D. Wessels and J. Muzio, *in preparation.*

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my permission.

Title of Dissertation:
Randomized Algorithms in Path Sensitization for Circuit Optimization and Delay Fault Tolerance

Author:

(Signature)

Paul M Wesson

(Name in Block Letters)

May 4 1995

(Date)