NEW PROBABILISTIC MEASURES FOR ACCELERATING THE AUTOMATIC TEST PATTERN GENERATION ALGORITHM

By

BENNY PHILLIPS

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1987

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1989

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July, 1993

OKLAHOMA STATE UNIVERSITY

NEW PROBABILISTIC MEASURES FOR
ACCELERATING THE AUTOMATIC
TEST PATTERN GENERATION
ALGORITHM

Thesis Approved:

C. M. Bacon
Thesis Adviser

Richard L. Cummins

Blayne E. Mayfield

John W. Fattahi

Thomas C. Collins
Dean of the Graduate College

PREFACE

The automatic test pattern generation for single stuck-at faults in practical digital circuits is a search process which must be accelerated.

Like many other research works [Goel 1981, Fujiwara and Shimono 1983, Agrawal et al. 1985, Schulz et al. 1987, Cheng and Chakraborty 1989, and Auth and Schulz 1991], the goal of this research is to obtain efficient ways for accelerating the automatic test pattern generation algorithm.

Presently, there are few reliable acceleration tools available. Some of these tools are probabilistic measures called "signal probabilities." Unfortunately, the computation of signal probabilities is expensive because it exponentially intensifies as the number of fanout input variables and the number of logic gates in the circuit increases. As a result, many researchers have tried to find efficient ways to evaluate the signal probabilities [Savir et al. 1984, Seth et al. 1985, and Chakravarty and Hunt 1990], but the result is still far from being satisfactory.

Instead of trying to approximate the signal probabilities, this research introduces new probabilistic measures called "signal priorities," whose computation relies on the "minimum-value distributions" of fanout input variables of the circuit. The signal priorities serve the same purpose as do the signal probabilities. That is, they are used to accelerate the automatic test pattern generation algorithm. However, their computation requires much less effort.

I wish to express my sincere gratitude to the individuals who assisted me in this project and during my coursework at Oklahoma State University. In particular, I wish to thank my major adviser, Dr. Charles M. Bacon, for his intelligent guidance, inspiration, and invaluable aid. I am also grateful to the other committee members, Dr. Richard L. Cummins, Dr. John Cartinhour, and Dr. Blayne E. Mayfield, for their advisement during

the course of this work. I would also like to express my gratitude to Mr. David R. Corder of the Oklahoma City Air Logistics Center for financially supporting the project.

Special thanks are due to my brothers and sisters, and my wife, Yu-hsin Huang, for their participation in the study. My deepest appreciation is extended to my parents who provided constant support, moral encouragement, and understanding.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Motivation

In recent years, the cost and time associated with the testing of digital systems continue to increase as their complexities increase. The testing and support costs for complex electronic circuits currently account for more than 70% of the life cycle cost of these products [Hanna and Horth 1991]. A large portion of this cost is due to testing techniques. Presently, at some large testing facilities across the United States, it is not uncommon to see test engineers still manually generate sequences of input test patterns (i.e., sequences of digital signal sets or vectors to be applied to the circuit under test) to detect the faulty components of digital circuit boards. This manual test pattern generation is a very tedious and time-consuming process and, very often, a large number of faults remain undetected. As a result, it is worthwhile to seek more efficient testing techniques that can reduce the overall cost and produce high-quality sequences of input test patterns that detect a high percentage of faults.

1.2 Overview of Test Pattern Generation (TPG)

This section is intended to give the reader an overview of TPG, which includes testing objectives in different types of digital circuits, various TPG methods, the testing level, and the computer model of the physical defects in digital circuits. During the discussion, the need for this research will be pointed out.

1.2.1 TPG Objectives in Combinational and Sequential Circuits

In a combinational circuit (one with no memory elements), each input pattern or vector applied to the circuit produces exactly one output response pattern. On the other hand, in a sequential circuit (one with memory elements), a sequence of input patterns is typically required to get the circuit's output to a desired state. Consequently, for a given fault, the objective of the TPG is to obtain a single input test pattern in a combinational circuit, and a sequence of input test patterns in a sequential circuit.

1.2.2 Test Pattern Generation Methods

There are several ways sequences of input test patterns can be created. First, test engineers can manually generate these sequences for detecting faults (i.e., making fault effects observable at the external output connections of the circuit). However, as mentioned earlier, this manual approach is time-consuming and often gives a low fault coverage. This is specially true when dealing with sequential circuits that often have faults requiring long sequences of test patterns to detect.

Second, by using a computer, sequences of test patterns can be created automatically based on a fault-oriented analysis of the circuit. When the test generation is completed, the automatically created test patterns are transferred to an Automatic Test Equipment (ATE) unit for the tests to be physically carried out. This computer-based process is referred to as Automatic Test Pattern Generation (ATPG). The ATPG method, of course, is much faster than the manual one. It also provides clear information on the fault coverage (i.e., information on how many faults have been detected).

Third, with today's advanced technologies in circuit design and fabrication, a digital circuit can be built to generate sequences of pseudorandom input patterns to test itself. Such a circuit is said to have a built-in self test (BIST). More information regarding the pseudorandom testing will be given in the next section. For now, the reader should note that pseudorandom patterns work very well with a combinational circuit because any fault effect can be observed immediately after applying an input test pattern.

However, this is not true for a sequential circuit. To get the circuit to a desired state, very often a specific, and sometimes unique, sequence of patterns is required at the inputs. Such a characteristic of the sequential circuits makes it very hard for pseudorandom testing to perform efficiently. To overcome this problem, the circuit design industry has initiated a new design method called "scan design" that essentially converts a sequential circuit into a combinational one to prepare for the pseudorandom testing.

However, BIST and scan features, according to Cheng and Chakraborty [1989], have several drawbacks: a) they require additional circuitry for storing input/output patterns (for a later comparison purpose) which increases the production cost, b) this additional circuitry can slow down the circuit performance, and c) it is difficult to determine the exact fault coverage.

Although BIST and scan-based circuits are gaining acceptance in the industry, the need for the ATPG has not been eliminated. One of the main reasons is because electronics products designed for commercial and military applications in the late seventies and early eighties are still in use, and most of these products normally do not have BIST and scan features. Furthermore, these older products cannot be easily replaced and they have to be maintained and tested on a regular basis. Based upon the need for improving the performance of the ATPG methods, this research was directed toward this area.

1.2.3 Testing Level and Fault Model

Generally, TPG methods can be classified according to the component level being tested. For example, there are gate-level testing, board-level testing, system-level testing, etc.. This research is concerned with the gate-level testing of Printed Circuit Boards (PCB's), each of which may consist of a combination of both combinational and sequential circuits.

This research refers to "primary" inputs (outputs) on the PCB as those circuit lines or signals that are externally controllable (observable). Usually, the numbers of primary inputs and logic gates of a PCB are large. These two factors can complicate the TPG. Some circuit boards have more than 100 primary inputs and about 3000 gates each.

At the gate-level testing, the objective is to determine input test patterns that can identify any faulty gates on the circuit board. A gate is considered faulty when it does not function correctly due to physical defects such as broken connections at its inputs and/or output.

There are many fault models that can represent various physical defects of the circuit lines (see [Abramovici et al. 1990, pp. 93-126]); however, the single stuck-at fault model is among the most commonly assumed model in many theoretical and commercial test generators. In the single stuck-at fault model, it is assumed that there can only be one fault of either stuck-at-0 (sa0) or stuck-at-1 (sa1) type occurring on one circuit line at any given time. This research will assume the single stuck-at fault model. The justification for this assumption will be provided in Chapter 2.

1.3 Major Issues in Automatic Test Pattern Generation

1.3.1 Complexity in the ATPG

As stated earlier, a sequential circuit often requires a sequence of patterns to propagate a fault effect to the output of the circuit (this fault propagation activity is necessary for fault detection). The problem is that when the sequence is long and/or unique, the ATPG requires a lengthy fault processing time. Very frequently, the process takes so long that a large number of faults have to be abandoned. Another problem is that when a special circuit line called "fanout" is present in the circuit, signal interdependencies are introduced. These signal interdependencies make the search for input test patterns very time-consuming. Therefore, the efficiency of an ATPG algorithm is very critical when the circuit's size and the number of fanouts are large.

The effect of fanout signals will be clearly illustrated in Chapter 2. For now, an example of fanout signals is shown in Figure 1.1. In this figure, the junction formed by lines t1, t2, and t3 is a fanout point or stem, line t1 is called the fanout head, and lines t2 and t3 are called fanout branches. There is also another fanout stem formed by t4, t5, and t6. Note that when a fanout signal propagates on different paths and reconverges at some

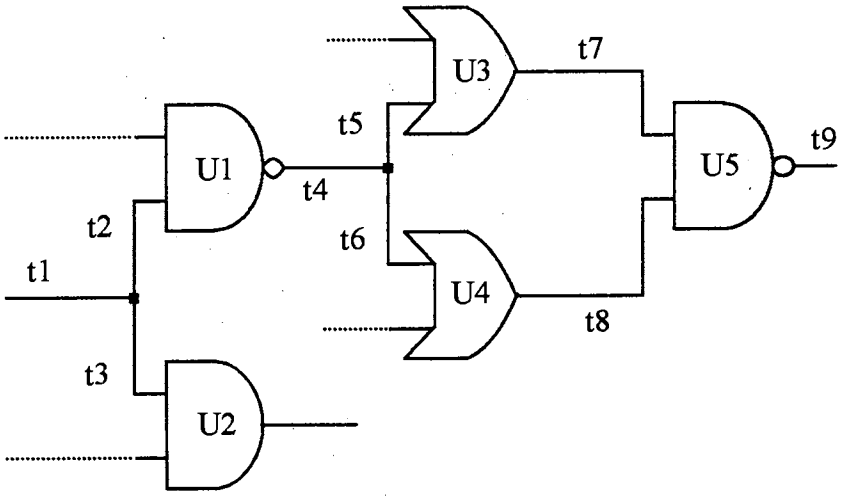


Figure 1.1. Fanouts and reconvergent fanout

point later in the circuit, it is referred to as a reconvergent fanout. The line where the fanout branches meet is called a reconvergent line. In Figure 1.1, t4 is a reconvergent fanout, and t9 is a reconvergent line. As will be seen, reconvergent fanouts create many problems in the ATPG.

1.3.2 Deterministic ATPG vs. Pseudorandom Testing

In deterministic ATPG (DATPG), information inputs to a computer are the description of the circuit and the list of faults to be checked. The fault list is usually provided by the maker of the circuit under test. The process of test generation takes place according to well-defined algorithms that ultimately generates necessary sequences of input test patterns for detecting faults on the list.

After the DATPG is completed, for each of the specified faults, the computer will store the input test patterns, the corresponding output patterns, and information regarding which faulty components are identified when each of the tests fails. These input/output patterns and other necessary information are then transferred to an ATE unit that will actually carry out the test on the circuit board.

In pseudorandom testing, as discussed in Section 1.2.2, the circuit under test is combinational. The fault-free circuit is first simulated. The resulting input/output patterns are then stored in the form of a look-up table to prepare for the actual pseudorandom testing on the ATE unit [Wagner et al. 1987].

During the actual pseudorandom testing, a computer, which is normally a built-in component of an ATE unit, applies a pre-defined number of pseudorandom input test patterns to the circuit. For each of the input patterns, the output response of the circuit is collected, and a comparison is made with the stored information to determine whether the circuit functions correctly. The circuit is considered failed when one of the comparisons fails.

The total number of pseudorandom input test patterns generated by the computer is referred as a "test length," which varies depending how high a test engineer wants the test quality to be. More faults are likely to be detected for a longer test length.

Obviously, pseudorandom testing is faster to work with compared to the deterministic technique. However, the problem with the pseudorandom testing is that the fault coverage is very difficult to evaluate. That is, the circuit under test still cannot be considered fault-free even after the pseudorandom testing is completed without any failure. This is because it is possible that a prescribed test length may not be long enough or may not include the appropriate input patterns to detect some faults in the circuit. This problem leads to the need for further statistical analysis on the quality of pseudorandom testing (see [Wagner et al. 1987]) which is beyond the scope of this research. As a result, the deterministic ATPG method, which has greater utility, will be the method of choice in this research.

1.3.3 Methods for Handling Sequential Circuits

As mentioned before, a PCB may consist of a combination of both sequential and combinational circuits. There are two methods for handling the ATPG for sequential circuits: a) simulation-based approach, and b) feedback-cutting approach.

In the first method, a circuit simulator is used to simulate the faults so that a sequence of input test patterns are determined. It is necessary that the fault simulation be guided by some criterion to prevent the process from becoming exhaustive. This simulation-based approach was recently seen in [Cheng and Chakraborty 1989 and Cheng et al. 1990]. Although the simulation-based approach can produce hazard-free input test patterns, it is very difficult to find a control criterion that can guide the simulation efficiently. More about this approach will be presented in Chapter 2.

In the second method, a sequential circuit is viewed as an array of identical combinational circuits. This array is referred to as an "iterative array model" of the original sequential circuit. The array is constructed by cutting the feedbacks in the original circuit. At the points where the cuts are made, pseudo inputs and pseudo outputs are created. Thus, the new combinational circuit will have the primary inputs and outputs of the original circuit as well as the pseudo inputs and outputs.

Each copy of the array circuit, which is identical to the original sequential circuit but with the feedbacks cut, is referred to as a "time frame" that represents a state of the circuit after one input pattern is applied. The pseudo inputs of the current time frame are the same as the pseudo outputs of the previous time frame. Figure 1.2 shows a typical model of a sequential circuit and its equivalent iterative array constructed by cutting the feedbacks.

The approach for handling sequential circuits by the iterative array model has been preferred by many researchers [Putzolu and Roth 1971, Muth 1976, Cheng and Chakraborty 1989, and Auth and Schulz 1991]. This is mainly because, whenever the iterative array model is assumed, test pattern generation algorithms designed for combinational circuits can be directly applied to sequential circuits. Using this approach, there is no need for specific algorithms for sequential circuits.

1.4 Accelerating Deterministic Automatic Test Pattern Generation

Many DATPG algorithms for combinational circuits have been developed over the years. However, because of the increasing complexity of digital systems, current research activity has focused on the acceleration of the already-existing DATPG algorithms. With this in mind, the following presents a brief look at the progress that has been made so far.

One of the very first DATPG algorithms called "D-algorithm" was designed by Paul Roth in 1966 [Roth 1966]. The D-algorithm was considered a landmark research work because it was formally proven to be a complete algorithm, which, for a given fault, guaranteed to find an input test pattern if one existed. One of the basic features of the D-algorithm was that its fault processing started at the fault's location, and advanced toward the primary outputs and inputs of the circuit. However, for certain types of combinational circuits, such a fault processing strategy is very inefficient due to an excessive number of signal value selections being repeated during the determination of the desired input test patterns. For these circuits, the D-algorithm was an exhaustive algorithm.

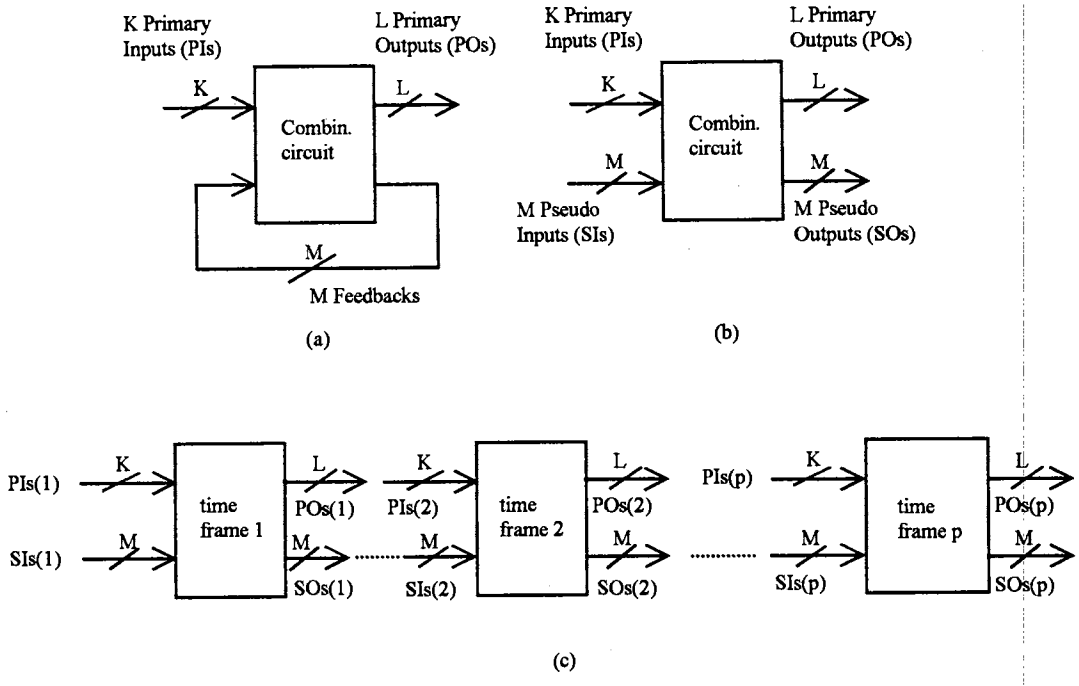


Figure 1.2. A sequential circuit and its equivalent iterative array model; (a) original; (b) after cutting the feedbacks, (c) iterative array circuit

In 1981, an algorithm called "PODEM" (path-oriented decision making) was developed to overcome the deficiency of the D-algorithm [Goel 1981]. The features that made PODEM differ from the D-algorithm were:

a) Instead of starting the fault processing at the fault's location, PODEM started at the primary inputs of the circuit and advanced toward the primary outputs of the circuit.

b) PODEM imposed several conditions for early termination of the search process whenever the fault's effect was found not observable.

By employing the above features, Goel showed that, for a certain type of combinational circuit, PODEM potentially could prevent the search for input test patterns from degenerating into an exhaustive process. Thus, the performance of PODEM was sometimes more efficient compared to that of the D-algorithm. However, PODEM still faced the same problem as did the D-algorithm for a more general class of combinational circuits.

Two years later, FAN (fanout-oriented test generation algorithm), developed by Fujiwara and Shimono [1983], introduced a new search strategy that further improved the performance of the test generation. Based on the fanout structure of the circuit, FAN's features were:

a) FAN's search mechanism could take on multiple paths in the circuit to speed up the fault processing.

b) Upon encountering a fanout signal that was fed by a fanout-free sub-circuit and not accessible by the fault effect, FAN's search process could be postponed so that the time it borrowed could be used for processing other tasks.

Basically, efforts on accelerating the already-existing DATPG algorithms discussed so far have been about how the search process can be guided so that it will not degenerate into an exhaustive procedure. These search strategies are directly based on a topological analysis of the circuit. This approach still has some drawbacks, which will be pointed out in Chapter 3.

Other search acceleration techniques were developed which relied on probabilistic measures such as the signal probabilities of the lines in the circuit. Agrawal et al. [1985]

were pioneers in promoting the probabilistic guidance for the search process in the DATPG. In his paper, titled "Probabilistically Guided Test Generation," Agrawal experimentally showed that the signal probabilities guided the search process more efficiently than did the search strategy of PODEM.

The probabilistic approach of Agrawal seemed very promising, but, another problem surfaced. The computational effort for evaluating the exact signal probabilities increased exponentially with the number of fanout input variables of the circuit. Researchers such as [Savir et al. 1984, Seth et al. 1985, and Chakravarty and Hunt 1990] have proposed ways to improve the efficiency aspect of the signal probability computation, but the results are still far from being satisfactory.

1.5 Goal and Contribution of The Research

Prior research has shown that accelerating the search process is a difficult task in the DATPG. However, a probabilistic approach has proven to be one way to accelerate the search process. The contribution of this research is that, instead of trying to approximate the signal probabilities as many other researchers have done, it introduces new parameters called "signal priorities," which serve the same purpose as do the signal probabilities, but their computation requires much less effort.

Chapter 2 reviews the major ideas and tools used in DATPG in both combinational and sequential circuits. Chapter 3 closely examines the current efforts in the acceleration of the search process in DATPG, and in the computation of signal probabilities. Chapter 4 examines the deficiencies of some existing probabilistic techniques discussed in Chapter 3, and then presents the development of signal priorities, a new probabilistic concept produced by this research. Chapter 4 also gives examples of the calculation and utility of signal priorities. Chapter 5 provides a summary of the research results and provides conclusions and ideas for future work.

CHAPTER II

CONCEPTS AND TOOLS USED IN DATPG

2.1 Single Stuck-at Fault Model and Logic Systems

2.1.1 Single Stuck-at Fault Model

The main types of faults considered in this research are the short and open. A short is a connection between two points that is not designed to be connected. An open, on the other hand, is an unintended break between two points. The short and open can modify the interconnections between the circuit components, and thus logically affect the function of the circuit. When there is either a short or an open between two points, the result will always be some fixed values at these points. If these fixed values are mapped into the two logic levels {0, 1}, these faults can be logically modeled as stuck-at-0 and stuck-at-1 faults.

Of course, stuck-at faults can occur simultaneously in a circuit. However, in most cases, multiple stuck-at faults can be detected by tests designed for single stuck-at faults. It has been shown by Jacob and Biswas [1987] that at least 99.67% of all multiple faults will be detected by single fault tests if the circuit has at least 3 primary or observable outputs. Therefore, this thesis will only consider testing for single stuck-at faults.

2.1.2 Systems of Logic Values

The commonly-known 2-value (binary) logic system is used to represent the signal values in a circuit when it is in a normal mode (fault-free) of operation. When a "don't-care" value is added to the binary logic to represent an unspecified or unknown

signal value, the result is a 3-value logic system. When a fault is present, some other logic systems must be used to represent the fault and its effect. Two of the logic systems used in this research are the 5-value and the 9-value which are described as follows.

The 5-value logic was originally introduced by Roth [1966] when he designed the DATPG algorithm. The 5-value logic uses the set of five symbols {0, 1, D, D', X} where D represents a composite signal value of a line in the circuit, D' is the complement of D, and X is a value for don't-care. Thus D can take on either a 0 or a 1 value. Its use is to simultaneously represent the good and the faulty behaviors of a line in the circuit. For example, when a line carries a symbolic value of D, and if it is physically observed to be a 1 (0), it means that the line is a good (faulty) one. The actual use of these composite values will be demonstrated in Chapter 3. Logic operations such as AND, OR, EXCLUSIVE-OR (XOR), and NOT or INVERT, etc., can be performed in the 5-value system. Some of the basic logic operations, which are also commonly referred to as the D-propagation rules, on a gate with 2 inputs are shown in Figure 2.1. The symbol "X" denotes the don't care condition.

The 5-value logic system works well for the DATPG in combinational circuits. However, for sequential circuits, it was found by Muth [1976] that the 5-value logic was not sufficient to represent the repeated effect of a fault over the time frames of the iterative array model. In order to take care of the repeated fault effect, Muth introduced the 9-value logic system to improve the performance of the D-algorithm in sequential circuits. The 9-value system and its 5-value equivalence are shown in Figure 2.2. In Figure 2.2, the corresponding binary values are shown as ordered pairs whose values on the left and on the right correspond to the "good machine" and the "faulty machine," respectively. The use of these composite values will be discussed in Chapter 3. The basic operations in the 9-value logic system are defined in Figures 2.3-4. In each of these figures, the top row and the left most column are input values, and the remaining entries are the results of the corresponding operation. It should be mentioned that these operations are component-wise. For example,

$$G0 \cap S0 = (0, X) \cap (0, 1) = (0 \cap 0, X \cap 1) = (0, X) = G0$$

where $\cap$ denotes the logic AND operation.

AND		OR		INVERTER		XOR	
IN	OUT	IN	OUT	IN	OUT	IN	OUT
0 X	0	1 X	1	0	1	0 0	0
X 0	0	X 1	1	1	0	0 1	1
1 D	D	0 D	D	D	D'	0 D	D
1 D'	D'	0 D'	D'	D'	D	0 D'	D'
D 1	D	D 0	D			1 0	1
D' 1	D'	D' 0	D'			1 1	0
D D	D	D D	D			1 D	D'
D D'	0	D D'	1			1 D'	D
D' D	0	D' D	1			D 0	D
D' D'	D'	D' D'	D'			D 1	D'
						D D	0
						D D'	1
						D' 0	D'
						D' 1	D
						D' D	1
						D' D'	0

Figure 2.1. D-propagation rules for AND, OR, INVERTER, & XOR gates

The 9-value Symbols	Corresponding Binary Values: (b_g,b_f)=(good value, faulty value)	Corresponding 5-Value Logic Used in the D-algorithm
0	(0,0)	0
G0	(0,X)	
S0	(0,1)	D'
F0	(X,0)	
U	(X,X)	X
F1	(X,1)	
S1	(1,0)	D
G1	(1,X)	
1	(1,1)	1

Figure 2.2. Definition of the 9-value logic system

2-input AND operation	0	G0	S0	F0	U	F1	S1	G1	1
0	0	0	0	0	0	0	0	0	0
G0	0	G0	G0	0	G0	G0	0	G0	G0
S0	0	G0	S0	0	G0	S0	0	G0	S0
F0	0	0	0	F0	F0	F0	F0	F0	F0
U	0	G0	G0	F0	U	U	F0	U	U
F1	0	G0	S0	F0	U	F1	F0	U	F1
S1	0	0	0	F0	F0	F0	S1	S1	S1
G1	0	G0	G0	F0	U	U	S1	G1	G1
1	0	G0	S0	F0	U	F1	S1	G1	1

Figure 2.3. AND operation for 2-input gate on 9-value logic system

2-input OR operation	0	G0	S0	F0	U	F1	S1	G1	1
0	0	G0	S0	F0	U	F1	S1	G1	1
G0	G0	G0	S0	U	U	F1	G1	G1	1
S0	S0	S0	S0	F1	F1	F1	1	1	1
F0	F0	U	F1	F0	U	F1	S1	G1	1
U	U	U	F1	U	U	F1	G1	G1	1
F1	F1	F1	F1	F1	F1	F1	1	1	1
S1	S1	G1	1	S1	G1	1	S1	G1	1
G1	G1	G1	1	G1	G1	1	G1	G1	1
1	1	1	1	1	1	1	1	1	1

(a)

Input	0	G0	S0	F0	U	F1	S1	G1	1
Output	1	G1	S1	F1	U	F0	S0	G0	0

(b)

Figure 2.4. (a) OR operation for 2-input gate on the 9-value logic system; (b) NOT operation

2.2 DATPG in Combinational Circuits

2.2.1 Fault Detection and Line Sensitization

A fault in a circuit is said to be detected whenever it is activated and propagated to at least one of the primary outputs of the circuit. Fault activation and propagation techniques vary depending on which logic system is being used. However, the basic idea is to send the fault effect to the outputs of the circuit so that the behavior of the faulty circuit can be distinguished from that of the good one. For example, if the 5-value logic is used, a sa0 (sa1) fault on a line x is said to be activated when x is set to D (D'). The fault symbol on x is then propagated according the D-propagation rules defined in Figure 2.1.

As an example, in Figure 2.5, line t6 has a sa0 fault. The first step is to activate the fault. Since the fault is a sa0, (t1, t2) is set to (0, X) to produce a 1 (a D value) on line t6 for activating the fault. The next step is to propagate the fault to t9, which is the primary output for this circuit, for external observation. In order to do this, (t7, t5) is set to (0, 1), and this implies that (t3, t8, t9) = (0, D', D'). From this example, it follows that the fault is detectable if there exists an input test sequence which can activate and propagate it to at least one of the primary outputs. Otherwise, the fault is said to be undetectable.

If a line's value changes corresponding to a situation in which a fault is present and the test for that fault is applied at the primary inputs of the circuit, it is said to be sensitized. A path that consists of the sensitized lines is called a sensitized path. This path will be designated by signal values of D or D'. In Figure 2.5, t6 is a sensitized line, and the path composed of t6, t8, and t9 is a sensitized path.

2.2.2 The Implication and Justification Processes

When a signal value is assigned to a line in the circuit, it usually results in some fixed values to be determined at some other lines. Signal implication refers to a process of computing these fixed values and, at the same time, making sure that these values are

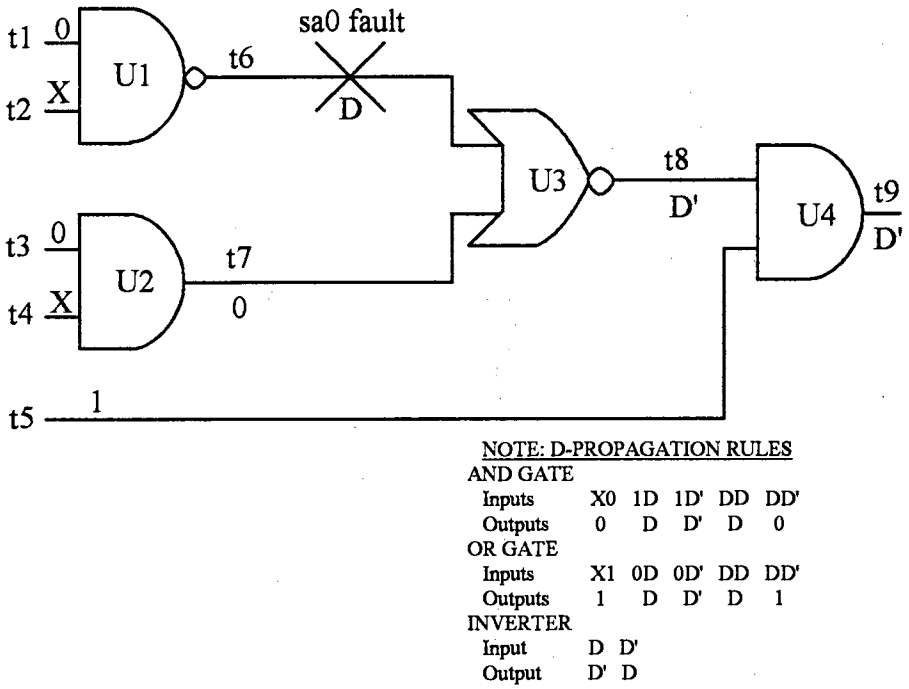


Figure 2.5. Example circuit for fault detection and line sensitization

consistent with other values that were previously determined. The signal implication process can either take on a forward direction toward the output or backward direction toward the input(s) of a logic gate. For example, when one of the inputs of an AND gate is set to 0, it implies that the gate's output is a 0. This is a forward implication. When the output of an AND gate is assigned to 1, it implies that the gate's inputs are all 1's. This is a backward implication.

Signal justification refers to a process of determining a particular input combination to a gate for achieving a given value at its output. The cost of the justification process depends upon the complexity of the logic system being used. For example, to justify a 0 at the output of a 2-input AND gate using the 5-value logic, one has to consider two choices: 0X, and X0. But if the 9-value logic is used, one has to consider four choices: (0/0, X/X), (X/X, 0/0), (0/X, X/0), and (X/0, 0/X). In general, to justify for a 0 at the output of a N-input AND gate, there will be N choices in the 5-value logic, and N^2 choices in the 9-value logic. So when the number of gates in the circuit is large, the signal justification process can be extremely lengthy.

2.2.3 Backtracking and Decision Tree

Very often there is more than one choice of input combinations to be selected during the justification of a signal. When the selection is incorrectly made, the continuing justification may lead to an inconsistency or a conflict. When this situation occurs, most DATPG algorithms will go back to the most recent decision point to remake the selection and resume the justification from there. This is referred to as a backtracking process. The backtracking process is illustrated in Figure 2.6. In Figure 2.6(a), suppose that t7 is required to be 0. Then one choice of the justified values of the primary inputs are (t1, t2, t3, t4) = (0, 0, 0, 0). Now suppose that t10 is also required to be 0. This implies that (t8, t9) should be either (0, 1) or (1, 0). However, either one of these two choices will create a conflict with the already-set primary input values. Therefore, a backtracking will take place as seen in the decision tree of Figure 2.6(b). In this case, backtracking will eventually lead to trying different values for t5 and t6.

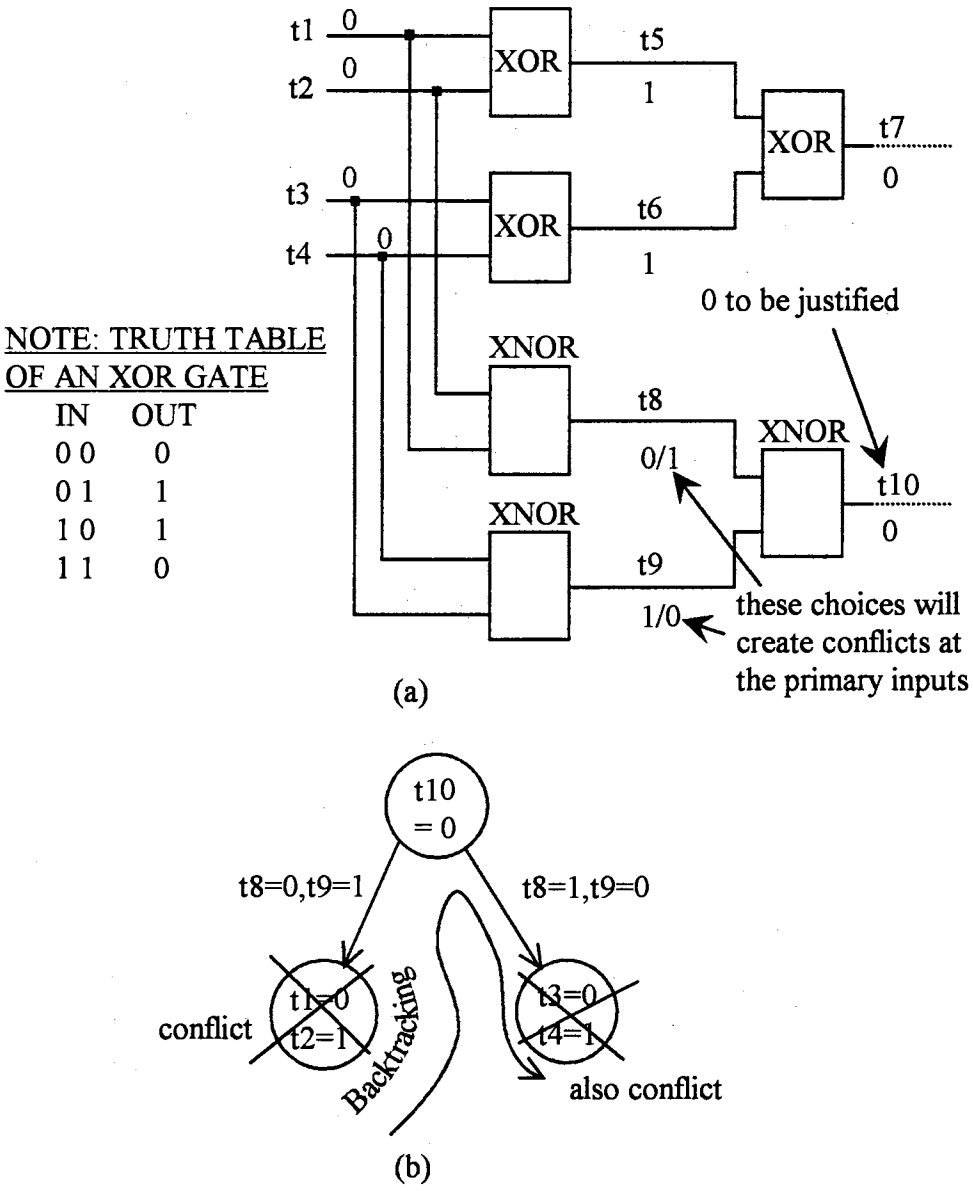


Figure 2.6. a) Example circuit illustrating backtracking process;
b) corresponding decision tree

2.2.4 Signal Probabilities

Two types of signal probabilities commonly used in DATPG are controllability, and observability.

First, the controllability of a signal line (not necessarily an output line) in a combinational circuit is defined as the probability that the line is set to a specified value (0 or 1) when an input pattern, which is randomly selected, is applied to the circuit. When the specified value is 0 (1), the controllability is referred to as 0-controllability (1-controllability). As an example, if the circuit has N primary inputs, and if there are P input combinations that can set line t1 to 0, then the 0-controllability of t1 is $P/2^N$, and the 1-controllability of t1 is $1-P/2^N$. The controllabilities play an important role in accelerating the test generation. In fact, many researchers have tried to find more efficient ways to compute the controllabilities [Savir et al. 1984, Jain and Agrawal 1985, Seth et al. 1985, and Chakravarty and Hunt 1990] so that it can be efficiently used to guide the test generation. More will be said about the signal controllabilities in Chapters 3 and 4.

Second, the observability of a signal line in a combinational circuit is defined as the probability that the line is set to a specified value, and this specified value is observable on at least one of the primary outputs when a randomly-selected input pattern is applied to the circuit. This parameter is mainly used to guide the fault propagation process. That is, when there is more than one fault propagation path, the path with highest observability will be selected.

The computation of the observability is beyond the scope of this research. For a detailed study on the use of this parameter, the reader is referred to [Jain and Agrawal 1985].

2.2.5 The Effect of Fanouts in Digital Circuits

As mentioned in Chapter 1, the DATPG for faults in a digital circuit becomes very difficult when fanout signals are present. Consider the circuit in Figure 2.7. The fault propagation path happens to be on the reconvergent fanout which, in all cases, blocks the

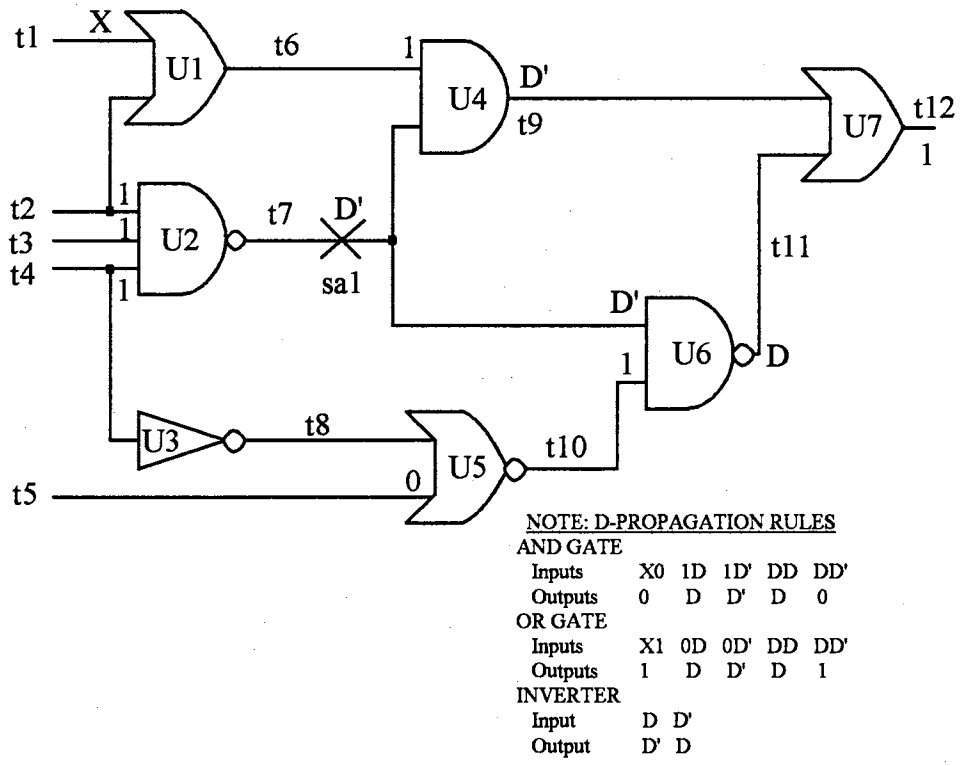


Figure 2.7. Example circuit illustrating the difficulty of fault detection when fanout is present (t12 can never be D or D')

fault effect from being seen at t12. This is because in order to activate the fault at t7 (producing t7 = 0), (t2, t3, t4) must be set to (1, 1, 1). This combination implies that t6 = 1 which naturally makes the signal t9 part of a fault propagation path. In order to propagate the fault to t12 through line t9, t11 must be 0. This requires (t4, t5) = (1, 0). But this combination makes (t9, t11) = (D', D), which cancels the fault effect on t12 (t12 = 1). Further analysis shows that t12 can never take a value of D or D'. Therefore, the fault is not detectable. In this case, the problem is that the test generator would waste its time to perform fruitless signal justification, which requires backtracking, if its search algorithm were not properly guided.

2.2.6 Implicit and Explicit Enumerations

One of the most obvious properties of a complete DATPG algorithm is the exhaustiveness. A DATPG algorithm is said to be complete when it enumerates all of the possible solutions for detecting a fault; therefore, it will definitely find a solution if it exists. However, there is a difference between explicit and implicit enumerations. The difference between the two concepts is that implicit enumeration attempts to limit the enumeration space (or search space) whereas the explicit enumeration does not. For example, when a sa0 fault at the output of a N-input AND gate is to be activated, explicit enumeration will enumerate all of the 2^N possible input combinations, but the implicit enumeration technique will start examining only the all-ones input combination because other input combinations will not lead to the fault activation. As a result, the implicit enumeration technique reduces the search space before fault processing begins. Generally, the explicit enumeration should be avoided because of its exhaustive search property.

2.3 DATPG Concepts and Tools in Sequential Circuits

2.3.1 DATPG Using Iterative Array Model

As mentioned in Chapter 1, using the iterative array model for DATPG in sequential circuits is preferred by many researchers. The main reason is that every copy of the array circuit is a combinational circuit. Hence all of the algorithms available for combinational circuits can be used.

However, the biggest problem with this approach is that the delays in the circuit are ignored due to the feedback cutting. Therefore, when testing is physically carried out on an ATE unit, the tests found may not detect anything because hazards may have been created when the circuit delays were ignored.

Fortunately, the tests found by the test generator can be very simply validated by using the Eichelberger's simulator [Eichelberger 1965] which can efficiently screen out the test patterns that create the hazards. Therefore, if the Eichelberger's simulator is used in the post-processing part of the DATPG, the tests generated can be considered reliable, provided that they pass the simulation. Note that the procedure for cutting the feedbacks and the general DATPG algorithm for handling sequential circuits were carefully provided by Putzolu and Roth [1971]. In Chapter 3, examples of feedback cutting and DATPG will be given. The Eichelberger's simulator is described next.

2.3.2 Eichelberger's Simulator

The Eichelberger's simulator was designed to detect various types of hazards and race conditions. For a detailed study on hazards and races, the reader is referred to [Mano 1984]. Basically, when the output(s) of a circuit does not behave as predicted, the circuit is said to contain a hazard, and when there are two or more feedback signals changing their values at the same time, a race condition is said to exist. Although the Eichelberger's simulator can be used to detect hazards and races, it is customary to assume that a circuit

designer would have followed simple design procedures, such as those in [Mano 1984, pp. 372-390] to prevent hazards and races from occurring. However, for the sake of reliability, the Eichelberger's simulator is included in the post-processing part of a DATPG algorithm [Putzolu and Roth 1971 and Muth 1976]. The Eichelberger's simulation procedure is described as follows.

Given a sequential circuit whose model is shown in Figure 1.2, the procedure for determining the final response of the output has two parts: a) determine all feedback signals that may be changing as a result of the input change, and b) determine whether or not these feedback signals will eventually stabilize in some predetermined states. If the final responses of the feedback signals are not stabilized (or not defined), the circuit is said to contain a hazard.

Procedure 2.1 uses a 3-value logic system in which U represents the undefined value of a signal. That is, the value of the signal cannot be determined from the circuit. Several logic operations in a 3-value logic system are shown in Figure 2.8(a).

Procedure 2.1:

Part a: Determining next states of primary outputs.

Step 1: Set the primary inputs x's and the primary outputs y_i (the current values) to 0's to get the circuit to a known state.

Step 2: Set primary inputs x's to U's, and evaluate primary outputs Y_i (the future values) to find out if one or more have changed from their current values to U.

Step 3: If one or more Y_i have changed from their current values to U, change the corresponding primary outputs y_i from their current values to U, and evaluate Y_i again. Repeat Step 3 of Part a until no additional changes in the Y_i are found.

Part b: Determining which Y_i is not defined.

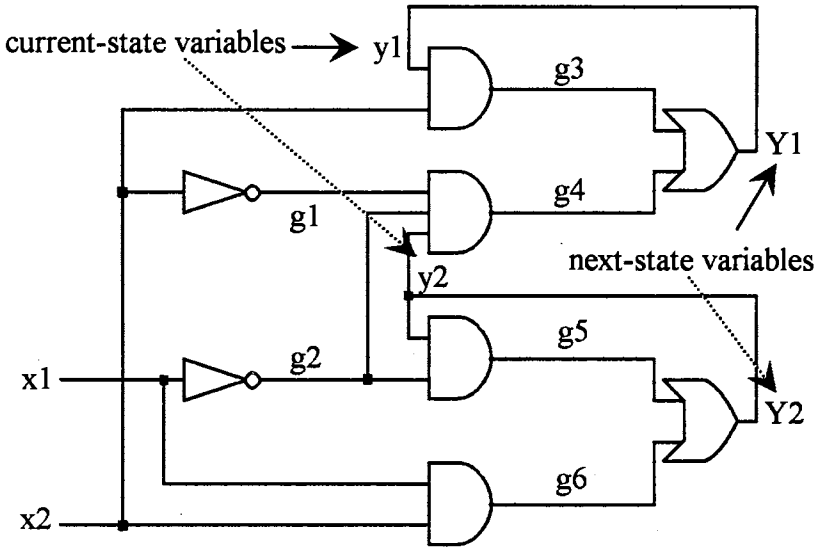
Step 1: Set primary inputs x's to new values (0 or 1), and evaluate the Y_i.

Step 2: If one or more of these Y_i changes from U to other values (0 or 1), change the corresponding y_i variables from U to 0 or 1, and

Truth Table for Three-value Logic

AND GATE				
Inputs	0X	0U	1U	UU
Outputs	0	0	U	U
OR GATE				
Inputs	1X	0U	1U	UU
Outputs	1	U	1	U
INVERTER				
Input	U			
Output	U			

(a)



(b)

Figure 2.8. a) Operations in 3-value logic system (U= undefined);
b) Asynchronous circuit for Example 2.1

evaluate Y_i again. Repeat Step 2, Part b until no additional changes in the Y_i are found.

Step 3: If any Y_i remains undefined, the circuit is said to contain a hazard.

Example 2.1: The final response of the asynchronous circuit in Figure 2.8(b) is now determined using the procedure outlined above. The result can be easily followed in Figure 2.9. Note that the final value of Y_1 (in the last column) is U. This means that this feedback signal is not stabilized, thus the circuit contains a hazard.

2.3.3 Simulation-based DATPG

Instead of using the iterative array model on sequential circuits, some DATPG algorithms rely on a fault simulator to determine the sequence of input test patterns for specified faults. Given a fault, a simulation-based DATPG algorithm simulates the circuit until the fault is detected. Of course, the simulation is not supposed to be an exhaustive process. Instead, it is controlled by a pre-defined cost function which guides the simulation into the direction that is closer to the fault detection. The simulation starts with any trial input pattern, and based on the simulated results, the cost function is computed to evaluate the "closeness" of the current trial input pattern to the fault detection. Then, based on the current result of the cost evaluation, the test generator decides whether it wants to keep the current trial pattern. The process will continue in a similar fashion. At the end, the trial patterns that meet the criteria set by the cost function will then be kept as the test sequence.

The simulation-based DATPG technique was recently used by Cheng and Agrawal [1987], and Cheng et al. [1990]. The advantage of the method is that the sequence found will naturally be free of hazards and races because the test patterns are all created through simulation. However, the time complexity of the algorithm and computation overhead intensifies since, for each of the trial input patterns, a simulation of the entire circuit and an evaluation of the cost function are required. Moreover, the criterion for selecting the

	Part a Step 1 (initialize the circuit)	Part a Step 2 (set x's = U, evaluate the next state variables Y's and other	Part a Step 3 (set y_2= Y_2 = U, re-evaluate Y's and other signals)	Part b Step 1 (set x's = 1, evaluate Y's and other signals)	Part b Step 2 (set y_2 = Y_2 = 1, re-evaluate Y's and other signals)
x1	0	U	U	1	1
x2	0	U	U	1	1
y1	0	0	0	U	U
y2	0	0	U	U	1
g1	1	U	U	0	0
g2	1	U	U	0	0
g3	0	0	0	U	U
g4	0	0	U	0	0
g5	0	0	U	U	1
g6	0	U	U	1	1
Y1	0	0	U	U	U
Y2	0	U	U	1	1

Figure 2.9. Results from Example 2.1

"good" trial input patterns has not been well defined. For example, in [Cheng et al. 1990], the selection of trial patterns depended only on a Hamming-distance heuristic which could not guarantee good guidance of the simulation.

CHAPTER III

CURRENT EFFORTS IN THE ACCELERATION OF DATPG AND IN THE COMPUTATION OF SIGNAL CONTROLLABILITIES

This chapter presents more detail of the current research work mentioned in Section 1.4. The first two sections focus on the DATPG acceleration techniques in both combinational and sequential circuits. As various algorithms are discussed, the efforts being made for accelerating the test generation process are clearly pointed out. The last section examines some of the current techniques for computing the signal controllabilities. Wherever appropriate, examples are given to clarify the basic ideas of these methods. Also, the limitations of these methods are identified.

3.1 Acceleration of DATPG Algorithms in Combinational Circuits

3.1.1 The D-algorithm

The D-algorithm [Roth 1966] is a path-sensitizing and complete algorithm which is guaranteed to find an input vector for detecting a given fault, if it exists. The algorithm uses the 5-value logic system to represent the fault effect, and the mixed process of signal implication and justification to determine input test patterns for single stuck-at faults in combinational circuits. The three main tasks performed by the D-algorithm are: a) activate the fault, b) propagate it to at least one of the primary outputs of the circuit for observation, and c) determine a primary input combination that will justify the first two actions.

Starting at the fault's location, the D-algorithm activates and propagates the fault forward toward the primary outputs by using the implication process. It then goes backward toward the primary inputs of the circuit to determine the correct primary input combination by using the justification process. The test generation is considered successful if a D or a D' is observed on at least one of the primary outputs and there is no signal conflict found during the signal justification. Procedure 3.1 gives the basic steps of the D-algorithm.

Procedure 3.1:

Step 1: Set the values of all circuit lines to don't-cares and activate the fault.

Step 2: Starting at the fault's location, propagate the fault to at least one of the primary outputs using appropriate D-propagation rules, and extend the signal justification all the way to the primary inputs.

Step 3: Continue the process in Step 2 until:

a) at least one of the primary outputs has a D or a D', and no signal conflicts are found during the justification process or

b) all possible primary input combinations have been tried and there is neither a D nor a D' observed at the outputs.

Example 3.1: The steps described in Procedure 3.1 are now applied to the circuit shown in Figure 3.1. In this figure, the stuck-at-1 fault is on line t6, thus the fault is activated by setting the value of line t6 to D' (Step 1). That is, D' implies a "good" value of 0 and a faulty value of 1 for line t6. Following Step 2, D' is propagated through U4 by using an appropriate D-propagation rule for the 2-input NAND gate (set t3 = 1). A D now appears on line t9. Since t9 is not a primary output, fault propagation continues with setting (t10, t11, t12) = (1, 1, 1). A D' is then observed at line t13 which is the primary output. The fault propagation can now stop. The next task is to perform the signal justification to determine the primary input combination will produce the current values at lines t9, t10, t11, t12, and t13. The required primary input combination is found to be (t1, t2, t3, t4, t5)

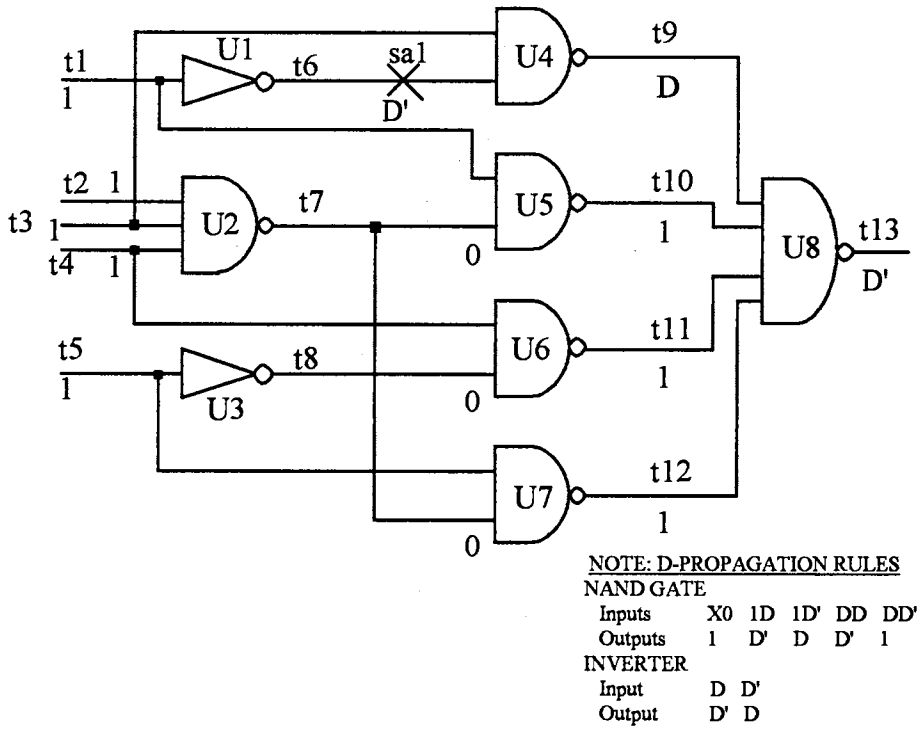


Figure 3.1. Circuit for Example 3.1 (the D-algorithm)

= (1, 1, 1, 1, 1). Note that this result is not unique. The steps performed in this example can be easily followed in Figure 3.2.

In Example 3.1, it so happens that the signal justification process was performed without any conflict. However, not every circuit is that straightforward. When working with a special type of combinational circuit that is commonly designed to implement error correction and translation functions in computer systems, the performance of the D-algorithm can become very inefficient due to a large number of decisions that have to be remade during the justification process [Goel 1981]. The following example will demonstrate the inefficiency of the D-algorithm.

Example 3.2: The objective is to find an input test pattern for the fault in Figure 3.3 by using the D-algorithm. The fault is activated by setting m = D. This implies (e, f) = (1, 1). From the D-propagation rules in Figure 2.1, k is set to 0 to propagate the fault through U8. Next, l is set to 0 to propagate the fault to the primary output z. Now the signal justification starts from line k to justify the current assignments. A primary input combination (a, b, c, d) are found to be (0, 0, 0, 0).

However, for these values of (a, b, c, d), line l will not have a value of 0 as originally requested. Therefore, the D-algorithm has to remake its decision on l by choosing another D-propagation rule for gate U9 (l = 1). The signal justification is now started all over again. The correct input test pattern for the fault is found to be (a, b, c, d, e, f) = (0, 0, 0, 1, 1, 1). The overall result for this example is shown in Figure 3.4.

As seen in Example 3.2, a conflict found during the signal justification can create a large number of backtrackings. This problem mainly accounts for the inefficiency of the D-algorithm. Additionally, when the fault is not detectable, the D-algorithm will degenerate into an exhaustive search process which becomes prohibitive as the size of the circuit increases.

ASSIGNMENTS	IMPLICATIONS	NOTES/RESULTS
t6=D'	t1=1	Activate the sa1 fault on t6 (force t6 = 0)
t3=1	t9=D	Propagate to t9
t10=1 t11=1 t12=1	t7=0 t4=1 t5=1 t2=1 t13=D'	D' is at t13. No signal conflict found. The test is (t1,t2,t3,t4,t5) = (1,1,1,1,1). This result is not unique.

Figure 3.2. Results for Example 3.1

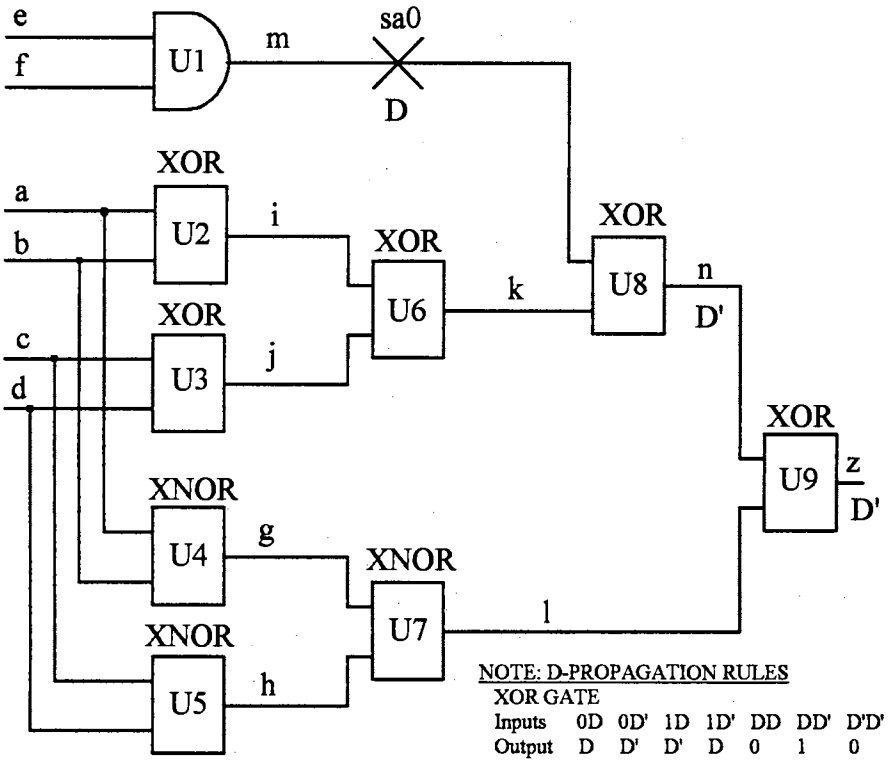


Figure 3.3. Example circuit 3.2 with XOR gates and reconvergent fanout

ASSIGNMENTS	IMPLICATIONS	RESULTS/NOTES
m=D	e=f=1	Activate the fault
k=0	n=D, i=j=0 a=b=0, c=d=0	Fault propagated to n
l=0	z=D g=0, h=1 a=0, b=1	b has a conflict
l=0	z=D g=1, h=0 c=0, d=1	Fault propagated to z d has a conflict
l=1	z=D' g=h=0 a=0, b=1 c=0, d=1	b has a conflict d has a conflict
k=1	n=D' i=0, j=1 a=b=0 c=0, d=1	Get another D-propa. rule for U8 Fault propagated to n
l=0	z=D' g=0, h=1 a=0, b=1	Fault propagated to z b has a conflict
l=0	z=D' g=1, h=0 a=b=0 c=0, d=1	Fault propagated to z A D' is observed at z. No conflict found. Stop.

Figure 3.4. Results for Example 3.2

3.1.2 Path-oriented Decision Making (PODEM)

Unlike the D-algorithm, PODEM [Goel 1981] begins the processing of a fault directly at the primary inputs rather than at the fault's location, and only propagates the fault in one direction toward the primary outputs of the circuit. When a fault is found to be neither activated nor propagated, PODEM uses a technique called "backtracing" to go back to the primary inputs and try a new input assignment. A backtrace is similar to a signal justification except that, during a backtrace, no values are assigned to internal lines. Signal values are assigned only to the primary inputs.

Recall that a backtracking takes place when a conflict is found during the signal justification. Since PODEM does not justify for values of internal lines of the circuit, it can overcome the problem of a large number of backtrackings in the D-algorithm, and can accelerate the test generation. Note that, however, PODEM may require some backtrackings during the primary signal justification. Procedure 3.2 below describes the basic steps of PODEM.

Procedure 3.2:

Step 1: Set the values of all circuit lines to don't-cares.

Step 2: Beginning at the primary inputs of the circuit, arbitrarily assign a value to one unassigned primary input at a time. For each assignment, use the forward implication to propagate the fault to at least one of the primary outputs. If the fault fails to appear at the primary outputs, backtrace to the primary inputs to choose an untried assignment. Also, stop the forward implication and backtrace to the primary inputs to choose an untried assignment if either one of the following propositions is true:

Proposition 1: The faulty line has the same value as the stuck-at level, i.e., the fault is not activated.

Proposition 2: The fault propagation path(s) becomes blocked, i.e., D or D' disappears along the propagating path(s).

Step 3: Continue Step 2 until

a) the fault effect is seen at a primary output or

b) all primary input combinations have been tried and no fault effect has been successfully propagated.

The propositions in Step 2 of PODEM can be viewed as heuristic rules which stop the forward implication and start the backtracing to the primary inputs when the fault is neither activated nor propagated. This feature prevents PODEM from performing a fruitless forward implication. From the use of the two propositions, it is clear that PODEM's enumeration technique is implicit, and its efficiency is measured by how quickly the fault is found to be neither activated nor propagated.

Example 3.3: For the circuit in Figure 3.3, PODEM is used to demonstrate its efficiency over the D-algorithm. The steps that PODEM performs can be easily followed in Figure 3.5. The required input test pattern is found to be (a, b, c, d, e, f) = (0, 0, 0, 0, 1, 1). Note that this answer is different from that in Example 3.2 because the solution is not unique.

3.1.3 Fanout Oriented Test Generation Algorithm

In PODEM, once the fault is neither activated nor propagated, the backtracing will begin on a single path back to the primary inputs to choose an untried primary input combination. This process can be improved for a faster test generation. FAN (fanout oriented test generation algorithm) [Fujiwara et al. 1983] introduces several extensions in order to accelerate PODEM. These extensions are:

a) If there is more than one propagation path found, choose the path which has the shortest distance between the fault's location to one of the primary outputs.

b) Backtracing can take on multiple paths to save time.

c) Instead of backtracing all the way to the primary inputs, the backtracing can stop at some selected internal fanout stems which are fed by fanout-free circuits and are not reachable by the fault.

ASSIGNMENTS	IMPLICATIONS	NOTES
e=0	m=0	Proposition 1 is violated. Reject e=0.
e=1	m=undefined	m is still undefined. Set f.
f=0	m=0	Proposition 1 is violated. Reject f=0.
f=1	m=D	Fault is activated.
a=0	i=g=undefined	i and g are undefined. Set b.
b=0	i=0, g=1	
c=0	j=h=undefined	j and h are undefined. Set d.
d=0	j=0, k=0, n=D, h=1, l=1, z=D'	Fault is detected.

Figure 3.5. Results for Example 3.3 with PODEM

The first extension is clearly a popular heuristic because the shorter the distance from the fault's location to the primary outputs, the more likely the fault will be propagated through. The second extension, as suggested by Fujiwara and Shimono, can be easily implemented by using the breadth-first search (see [Cormen et al. 1990] for an excellent discussion on the breath-first search algorithm). The third extension, however, requires some extra work to determine which fanout points in the circuit can be used as breaking points for the backtracing process. This last extension is discussed in further detail.

The third of FAN's extensions originates from the observation that for a signal that is fed by a fanout-free combinational sub-circuit, it is straightforward to justify for an arbitrary assignment. Therefore, a backtracing can stop at a selected internal fanout point, which must be fed by a fanout-free sub-circuit and not reachable by the fault being processed. When that fanout point is encountered during a backtracing, an assignment can be made to it, and its justification can be deferred to a later time. The time FAN borrows from such a postponement is used to concentrate on other processes.

The fanout point that is selected as a location for the backtracing process to stop is hereafter referred to as a backtrace stop. The backtrace stop is determined by a criterion that will be described after another example, which briefly shows how the FAN's third extension can be incorporated into PODEM to accelerate the test generation.

Example 3.4: Referring to the circuit in Figure 3.6, the objective is to demonstrate how FAN can accelerate PODEM by employing the backtrace stop. Assume that the sa0 fault on line h has been activated (h=D). To propagate this fault through U4, j must be set to 1. If PODEM were used, additional effort must be required to set j=1 since PODEM achieves this objective by manipulating the primary inputs (a, b, c). On the other hand, by realizing that j is a backtrace stop, FAN immediately sets signal j to 1, postpones the justification of j to a later time, and continues the fault propagation process from there.

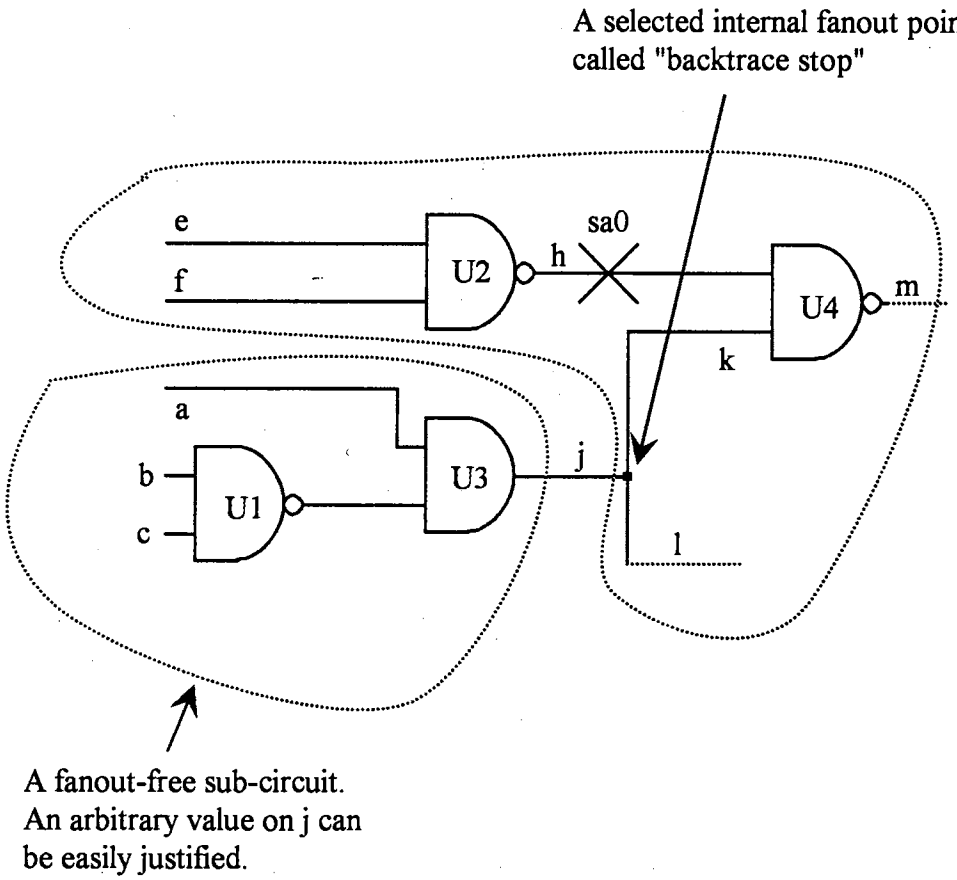


Figure 3.6. Circuit for Example 3.4

The process of finding the backtrace stop is mainly based on the controlling values of the logic gates on the paths that FAN backtraces. The controlling value of a gate is an input value that is most easily used to control the assigned output of the gate. For example, for an AND gate whose assigned output is a 0, the controlling value is a 0. For an OR gate whose assigned output is a 1, the controlling value is a 1.

During the multiple-path backtracing, the controlling values of each of the gates on each of the paths are computed and stored. When the backtracings on different paths meet at a fanout point, say F1, FAN counts how many times the 0 and 1 are requested on the fanout branches. If both 0 and 1 are requested, backtracing will continue; otherwise, backtracing will stop. Upon stopping the backtracing, FAN assigns the requested value to F1, and postpones the justification for F1 to a later time.

Example 3.5: Refer to Figure 3.7 for the computation of a backtrace stop. For any signal line x, let $n_0(x)$ denote the number of times that the value 0 is requested at x, and $n_1(x)$ denote the number of times that the value 1 is requested at x. Initially, suppose that during the multiple-path backtracing (which might have begun at some other location not shown in the figure) some signal values are requested at lines q and r. Let the notation $(q, n_0(q), n_1(q)) = (q, 0, 1)$ imply the value 1 is requested once at line q, and $(r, n_0(r), n_1(r)) = (r, 1, 0)$ imply the value 0 is requested once at line r.

Based on the two known requests at lines q and r, and the controlling values of the NAND and NOR gates (0 and 1, respectively), the values of n and p are set to 0 and 1, respectively. The corresponding notations are (n, 1, 0) and (p, 0, 1). Note that lines n and p are examined before lines j, k, l, and m because the multiple backtracing, as discussed earlier, is implemented by using the breadth-first search algorithm. Since (n, p) = (0, 1), it implies that (j, k) = (1, 1), and (l, m) = (1, 1).

At this time, FAN encounters the fanout point F1. By examining the requested values on lines k and l, FAN finds that k requests the value 1 once, and l requests the value 1 once. Therefore, $n_1(h)$ is computed by adding the $n_1(k)$ and

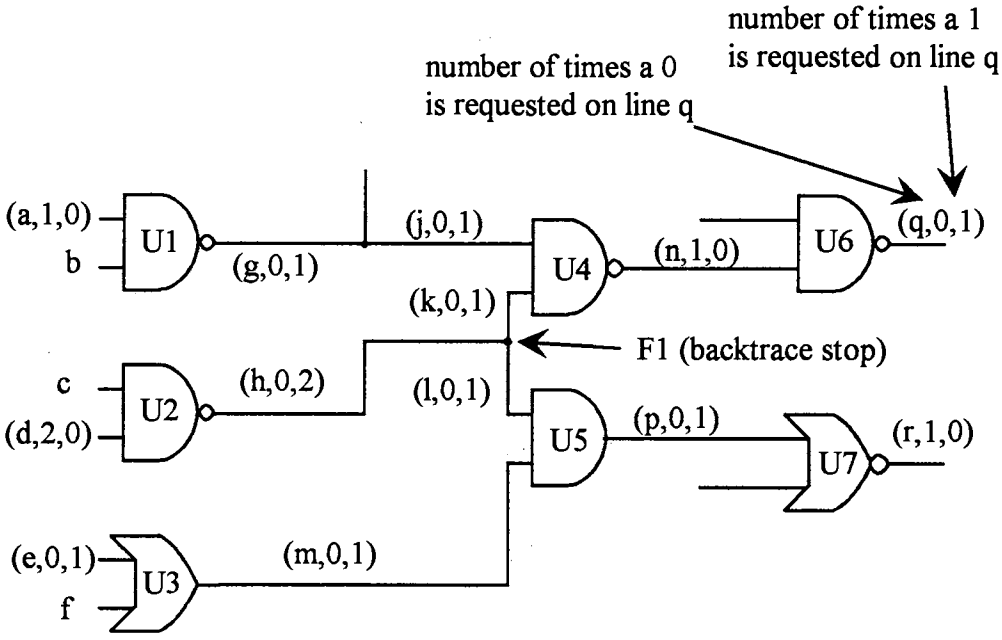


Figure 3.7. Circuit for Example 3.5

$n_1(l)$ together. The result at h is (h, 0, 2). Since only 1's are requested at h, i.e., no conflict is found, the fanout point F1 is chosen to be the backtrace stop.

The idea of using the multiple-path backtracing and the backtrace stop is quite innovative. However, the idea of the controlling value of a logic gate, in fact, is based on an assumption that the controllabilities of the inputs of a gate at any location in the circuit are equally likely. Based on such an assumption, Fujiwara and Shimono argued that the controlling value of an AND (OR) gate was 0 (1). This assumption cannot always be justified. For instance, consider an AND gate whose inputs have small 0-controllabilities ($<< 0.5$). In this situation, it is obvious that 0 is no longer a controlling value of the AND gate because it is now easier to set the gate's output to 1.

3.1.4 Probabilistically Guided Test Generation

The extensions introduced by FAN are some of the many ways used to accelerate the test generation. In this section, another technique called "probabilistically guided test generation" [Agrawal et al. 1985] is presented.

In FAN, recall that the controlling values of logic gates are used to compute the backtrace stops if they exist to reduce the number of backtrace to the primary inputs of the circuit, and that the path with the shortest distance from the fault to the primary outputs is chosen to be the propagation path. Similar ideas are employed here, but the difference is that the computed values of controllability and observability of the lines in the circuit are used to accelerate PODEM. For example, when justifying a 0 output of a 2-input AND gate, there are two choices at the inputs: 0x and x0. If the 0-controllabilities of each of the two inputs are known, the one with higher 0-controllability will be assigned a 0. During the fault propagation, when there is more than one line that the fault can be propagated through to the primary outputs, the line with the highest observability is selected.

Obviously, the use of the signal probabilities is of great help in accelerating the test generation. It was stated in Chapter 1 that this probabilistic approach would be followed

in this research. However, instead of using the signal controllabilities, this research will use newly-introduced signal priorities whose computation requires much less effort than that of the signal controllabilities. The introduction and development of the signal priorities will be presented in Chapter 4.

3.2 Acceleration of DATPG Algorithms in Sequential Circuits

3.2.1 The Heuristic Algorithm

The Heuristic Algorithm [Putzolu and Roth 1971] is the D-algorithm extended to work for sequential circuits through the use of the iterative array model. The algorithm is heuristic in the sense that it is not a complete algorithm. That is, it does not guarantee finding a test pattern for any given fault, even if one exists. The algorithm performs the test generation in three steps:

1) Cut the feedbacks in the sequential circuit to form the iterative array of an arbitrary length p (p is the number of the time frames).
2) Generate a potential test sequence T for a given fault.
3) Simulate this potential test sequence using the Eichelberger's simulator to see if the sequence creates races and hazards.

Before closely examining the Heuristic Algorithm, there are several points that should be mentioned: First, at least for asynchronous sequential circuits, there has been no known method for predicting what p will be prior to the test generation [Abramovici et al. 1990, p. 251]. Second, since every copy of the array circuit is identical, the test generator does not have to maintain all p copies of the circuit in the computer memory. Nevertheless, the signal values in each of the time frames must be separately stored.

Given a sequential circuit S and a fault F in S, the objective is to find a sequence of primary input patterns that can detect F. The basic steps of the Heuristic Algorithm are shown in the following procedure.

Procedure 3.3:

Step 1: Choose an arbitrary number of time frames p, and remodel the sequential circuit as a pseudo-combinational iterative array (see Figure 1.2).

Step 2: Choose p as the last time frame, use the D-algorithm to compute a test pattern T(p) for detecting F in the p^{th} time frame. Perform the fault propagation on only one path at a time. The shortest propagation path should be given the highest priority. T(p) can depend on primary inputs and pseudo outputs coming from the previous time frame (p-1) which are denoted as SOs(p-1).

a) If T(p) depends on only primary inputs, the test generation succeeds, and the test sequence consists of only one pattern T(p). Stop the test generation and return T(p).

b) If T(p) also depends on some SOs(p-1), then these SOs(p-1) must be justified in their own time frame. This means that the test generator now has to determine the test pattern T(p-1). The justification process is considered successful if no conflict (in the D-algorithm's sense) is found, and the pattern T(p-1) does not depend on any of the SIs(p-1) (pseudo inputs in the $(p-1)^{th}$ time frame); otherwise, the justification fails. If the justification succeeds and T(p-1) does not need further justification, i.e., T(p-1) does not depend on any of SOs(p-2), then stop the test generation and return the sequence T(p-1), T(p) as a potential test sequence T; otherwise, go to Step 3.

Step 3: If necessary, continue to justify for T(p-1) until the test generator finds a pattern T(p-r) (r is a positive integer) in some $(p-r)^{th}$ time frame that no longer depends on any of the SOs(p-r-1). The final result is the potential test sequence T which consists of the patterns T(p-r), T(p-r+1), T(p-r+2), ..., T(p-1), T(p).

Step 4: Simulate (using the Eichelberger's simulator) the potential test sequence T found in Step 2 or Step 3 to see if it is free of races and hazards. Reject the sequence if it does create races and hazards.

Example 3.6: The example sequential circuit is shown in Figure 3.8. For Step 1 of Procedure 3.3, p is arbitrarily chosen to be 2, i.e., there would be 2 copies of the original circuit, and the original circuit is remodeled as seen in Figure 3.9. Note that the numbers inside the parentheses are the time frame indicators. The input pattern that carries the largest time frame indicator (in this case, it is 2) corresponds to the last input pattern of the test sequence to be applied to the circuit. For Step 2, the D-algorithm is used and the result is shown in Figures 3.10-11. The Eichelberger's simulation (not shown here) was carried out by Putzolu and Roth, and the test sequence T in Figure 3.11 was indeed a good test for the fault.

The problems with the Heuristic Algorithm are that, first, it is based on the D-algorithm which is inefficient in handling the backtracking process. Therefore, it can be slow. Second, it is not a complete algorithm. Third, it does not address the repeated fault effect which refers to the indeterminate values at the pseudo outputs at the start of the test. That is, it is possible that some patterns of the test sequence found may be dependent on some assumed values of the pseudo outputs of the previous time frame. But one cannot really tell what these values are when the actual test begins on an ATE unit. If these values are not the same as those assumed, the test may fail. This problem is eliminated by the next algorithm.

3.2.2 The 9-value Algorithm

The 9-value Algorithm [Muth 1976] is very much the same as the Heuristic Algorithm described previously except that it uses the 9-value logic system instead of the 5-value logic one. It is a complete algorithm, and it can handle the repeated effects of the fault, which were not addressed by the Heuristic Algorithm. The incompleteness of the Heuristic Algorithm and its lack of consideration for the repeated fault effect are demonstrated in the following examples.

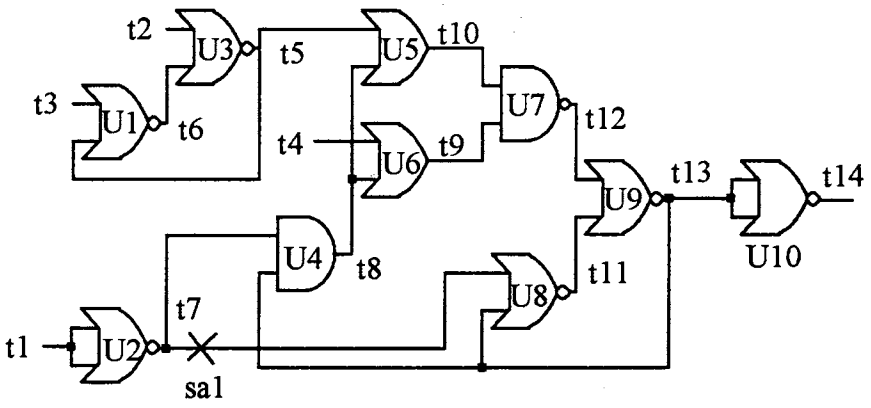


Figure 3.8. Circuit for Example 3.6

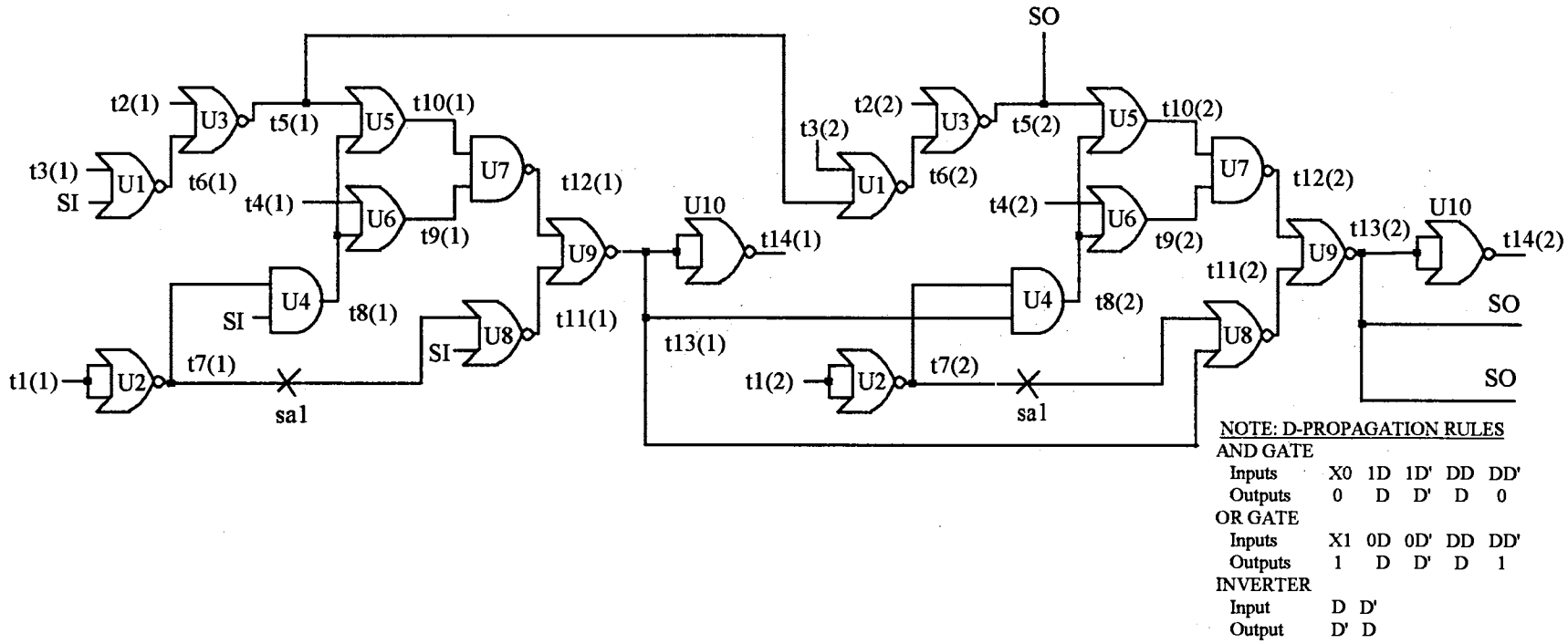


Figure 3.9. Two copies (p=2) of example circuit in Figure 3.8

ASSIGNMENTS	IMPLICATIONS	RESULTS/NOTES
t7(2)=D'	t1(2)=1	Fault is activated.
t11(2)=D	t14(1)=1, t8(2)=0, t13(1)=0	Fault is propagated through U8 because U8 is closer to primary output.
t12(2)=0	t13(2)=D', t14(2)=D, t9(2)=1, t10(2)=t4(2)= t5(2)=1 t2(2)=0, t6(2)=0	Fault is propagated through U10. Start the justification process.
t11(1)=1	SI=0	Conflict. Pseudo input SI must be a don't care.
t12(1)=1	t9(1)=0 or t10(1)=0	
t9(1)=0	t4(1)=t8(1)= t7(1)=0	Conflict. t7(1) cannot be set to 0 since it is a stuck-at-1 fault. Fault propagation through U8 is not possible. Try another propagation path.

Figure 3.10. Test generation of Example 3.6 (first trial)

ASSIGNMENTS	IMPLICATIONS AND/OR JUSTIFICATION	RESULTS/NOTES
t7(2)=D'	t1(2)=1	Fault is activated.
t8(2)=D'	t13(1)=1, t14(1)=0, t11(2)=0 t11(1)=t12(1)=0, t7(1)=1, t1(1)=0, t9(1)=t10(1)=1	Fault is now propagated through U4 because the path through U8 was blocked as seen in the first trial.
t4(2)=0	t9(2)=D'	Fault is propagated through U6.
t10(2)=1	t12(2)=D, t13(2)=D', t14(2)=D, t5(2)=1, t2(2)=t6(2)=0	Fault is propagated to the primary output. Now perform the justification process.
t4(1)=t5(1)=1	t2(1)= t6(1)=0, t3(1)=1, t3(2)=X	Justification is completed without conflict. Test sequence does not depend on any pseudo input. Test generation is successful. The desire test sequence T is (t1(1), t2(1), t3(1), t4(1)) = (0, 0, 1, 1) (t1(2), t2(2), t3(2), t4(2)) = (1, 0, X, 0)

Figure 3.11. Test generation of Example 3.6 (successful trial)

Example 3.7: The circuit shown in Figure 3.12 will demonstrate the incompleteness of the D-algorithm. The iterative array model for the circuit of Figure 3.12 is shown in Figure 3.13. Because of the incompleteness of the algorithm, the test sequence cannot be found as shown in Figure 3.14.

Example 3.8: For the same circuit in Figure 3.12, the 9-value Algorithm is applied and the test sequence is found. The definition of the 9-value logic system and its operations are shown in Figures 2.2-4. The overall result of this example is shown in Figure 3.15. Note that the result in Figure 3.15 is based on the assumption that k(1) (a pseudo output) can be a 1 (or G1 in the 9-value logic). In fact, it was set to G1 when h(2) was justified. But if it happens to be a 0 at the start of the test on an ATE unit, then in order for the test to remain valid, according the algorithm, all of the G-values on (g(1), h(1), k(1), h(2), k(2)) can be changed to S-values.

The 9-value Algorithm successfully finds the test in Example 3.8 because it simultaneously represents the good and faulty values of all signal lines in the circuit while the Heuristic Algorithm only does this for the stuck-at line. However, since the 9-value Algorithm uses the 9-value logic system, the search space for the justification process is much larger compared to that of the Heuristic Algorithm. To overcome this disadvantage, a test generator called Gentest [Cheng and Chakraborty 1989] introduced a technique to reduce the 9-value search space.

Gentest first splits the 9-value logic model into two 3-value logic models. One is for the good machine, and the other is for the faulty machine. Gentest then performs the justification process for the good and the faulty machines separately. After finishing the separate justifications, the results are then combined to obtain the final 9-value model by using the relationship between the two 3-value models. However, the search space can be further reduced if parameters such as signal controllabilities are incorporated into the test generator.

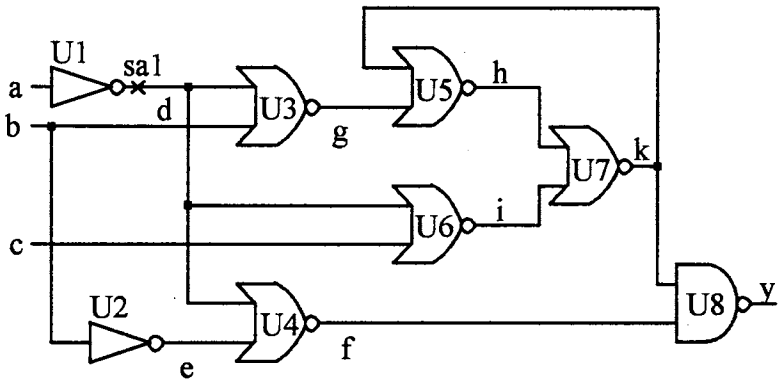


Figure 3.12. Circuit for Example 3.7

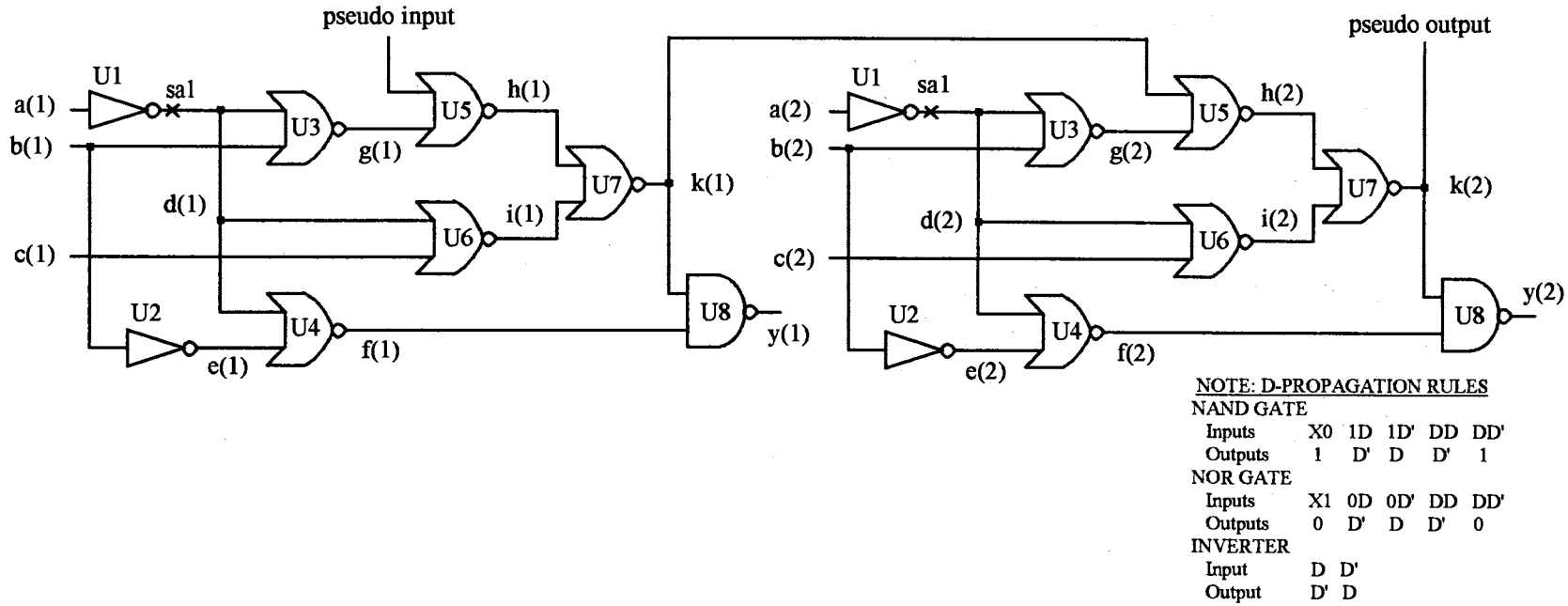


Figure 3.13. Iterative combinational circuit for Example 3.7

ASSIGNMENT	IMPLICATION AND/OR JUSTIFICATION	NOTES/RESULTS
d(2)=D'	a(2)=1	Fault is activated.
e(2)=0, c(2)=1	f(2)=D, b(2)=1, g(2)=0, i(2)=0	Fault is chosen to propagate only through U4 (shortest path).
k(2)=1	y(2)=D', h(2)=0, k(1)=1, h(1)=i(1)=0, g(1)=1, c(1)=1, b(1)=0, d(1)=0	Fault is at the primary output. Pseudo input should be don't-care whenever possible. A conflict found at d(1) since g(1)=1 requires b(1)=d(1)=0, but d(1) cannot be a 0 since it is a sa1 fault. Try the propagation path through U6.
c(2)=0, b(2)=1	g(2)=0, e(2)=0, f(2)=D, i(2)=D	
h(2)=0	k(2)=D'	(f(2),k(2))=(D,D') gives y(2)=0. The propagation path is blocked. Try the path through U3.
b(2)=0, c(2)=1	g(2)=D, e(2)=1, f(2)=0	The propagation path is again blocked due to f(2)=0. Procedure terminates with no test sequence found.

Figure 3.14. Example demonstrating the incompleteness of the Heuristic Algorithm

ASSIGNMENTS	IMPLICATION AND/OR JUSTIFICATION	NOTES/RESULTS
d(2)=S0	a(2)=G1	Fault is activated (see Figures 2.3-5 for fault activation and operations in 9-value logic system).
e(2)=G0, c(2)=G1	b(2)=G1, i(2)=G0, f(2)=S1, g(2)=0	Fault is propagated through U4.
k(2)=G1	y(2)=S1, h(2)=G0, k(1)=G1, h(1)=i(1)=G0, g(1)=G1, d(1)=b(1)=G0, a(1)=G1, c(1)=G1	Fault is propagated to the primary output y(2). No conflict is found during the justification process. The test sequence is (a(1), b(1), c(1)) = (G1, G0, G1) = (1, 0, 1), and (a(2), b(2), c(2)) = (G1, G1, G1) = (1, 1, 1).

Figure 3.15. Test generation using 9-value Algorithm

3.3 Current Techniques in the Computation of Signal Controllabilities

3.3.1 Cutting Algorithm

For a fanout-free circuit, the computation of 1-controllabilities can be performed directly using the simple probability expressions shown below.

a) An AND gate U with output g and N inputs (X_1, X_2, ..., X_N):

$$\text{Prob}\{\text{Output of U is 1}\} = \text{Prob}\{U(g) = 1\}$$
$$= (\text{Prob}\{X_1 = 1\})(\text{Prob}\{X_2 = 1\}) \ldots (\text{Prob}\{X_N = 1\}) \quad (3.1)$$

b) An OR gate U with output g and N inputs (X_1, X_2, ..., X_N):

$$\text{Prob}\{U(g) = 1\} = 1 - (1 - \text{Prob}\{X_1 = 1\}) \ldots (1 - \text{Prob}\{X_N = 1\}) \quad (3.2)$$

c) A NOT gate U with input i and output g:

$$\text{Prob}\{U(g) = 1\} = 1 - \text{Prob}\{U(i) = 1\} \quad (3.3)$$

d) XOR and XNOR gates:

The 1-controllability of the output of each of these gates is calculated in terms of the 1-controllabilities of the outputs of AND, OR, and NOT gates that implement the XOR or XNOR function.

By taking advantage of the above properties (Equations 3.1-3), the Cutting Algorithm [Savir et al. 1984] turns a general combinational circuit into a fanout-free circuit by cutting all but one of the fanout branches that reconverge, then calculating the lower and upper bounds of the 1-controllabilities of the reconvergent signals. The cutting rule is depicted in Figure 3.16.

Recall that the 1-controllability of a line is defined as the ratio of the number of input patterns that set the line to 1 to the number of all possible input patterns that can be applied to the circuit. The underlying assumption of this definition is that the line's value is controlled through all possible paths, some of which may be created by fanouts, from the primary inputs to the line's location. Therefore, the ratio of the number of the input patterns that set a line to 1 through only a single fanout path to the number of all possible input patterns can be viewed as the lower bound of the 1-controllability. Assuming that

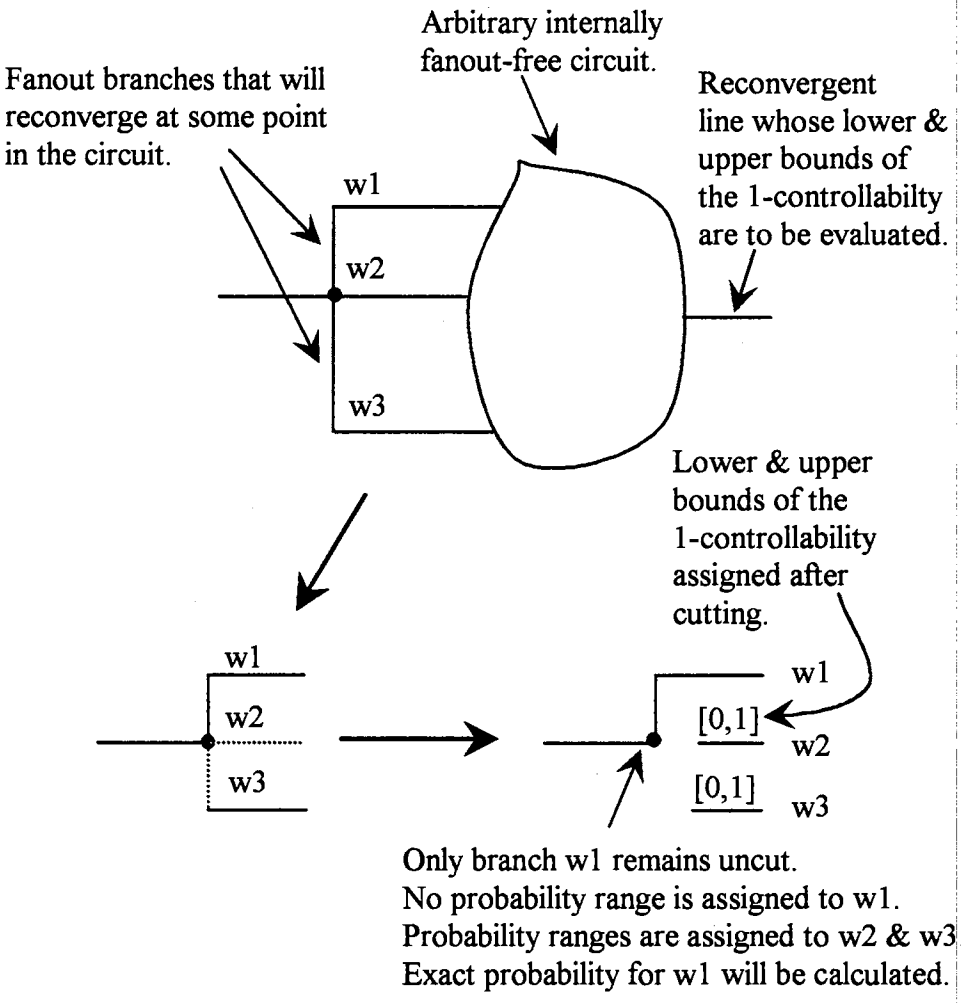


Figure 3.16. Cutting rule for the Cutting Algorithm

the signal probabilities of the primary inputs are all equal to 0.5, the basic steps of the Cutting Algorithm are described in Procedure 3.4 below.

Procedure 3.4:

Step 1: Assign a 1-controllability of 0.5 to each of the primary inputs.

Step 2: Cut the reconvergent fanout branches to turn the circuit into a fanout-free circuit. After cutting, assign a range of the 1-controllabilities, i.e., values in the continuous interval [0, 1], to each of the cut fanout branches and establish the 1-controllability of the uncut branch. Then compute the lower and upper bounds of the 1-controllabilities of all reconvergent lines using the probabilities defined in Equations 3.1-3.

Example 3.9: Referring to Figure 3.17(a), the fanout branches are w1, w2, and w3 and the reconvergent lines are r1, and r2. The objective is to compute the lower and upper bounds of the 1-controllabilities of r1 and r2. Following Procedure 3.4 step by step, one cuts all the fanout branches except w3 (see Figure 3.17(b)). Then

1-controllability for w3 = 1 - (0.5)(0.5) = 0.75

Lower bound for t1's 1-controllability = 1 - (0.5)(1) = 0.5

Upper bound for t1's 1-controllability = 1 - (0.5)(0) = 1

Lower bound for t2's 1-controllability = 1 - (0.5)(1) = 0.5

Upper bound for t2's 1-controllability = 1 - (0.5)(0) = 1

1-controllability for t3 = 1 - (0.75)(0.5) = 0.625

Lower bound for t4's 1-controllability = 1 - (0.5)(1) = 0.5

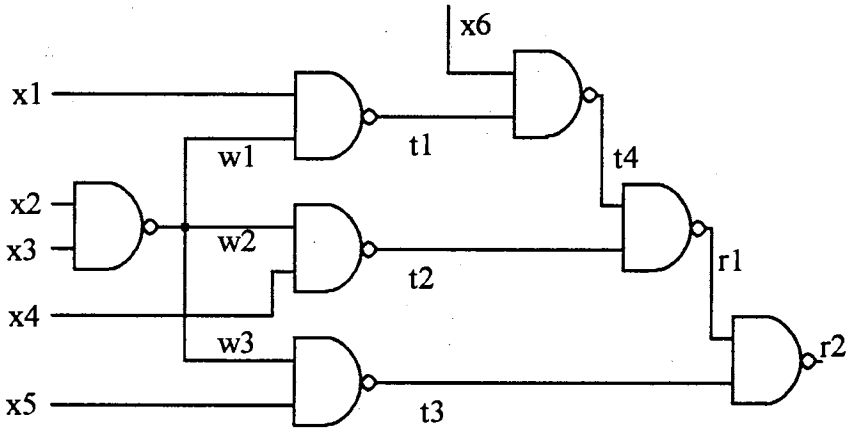
Upper bound for t4's 1-controllability = 1 - (0.5)(0.5) = 0.75

Lower bound for r1's 1-controllability = 1 - (0.75)(1) = 0.25

Upper bound for r1's 1-controllability = 1 - (0.5)(0.5) = 0.75

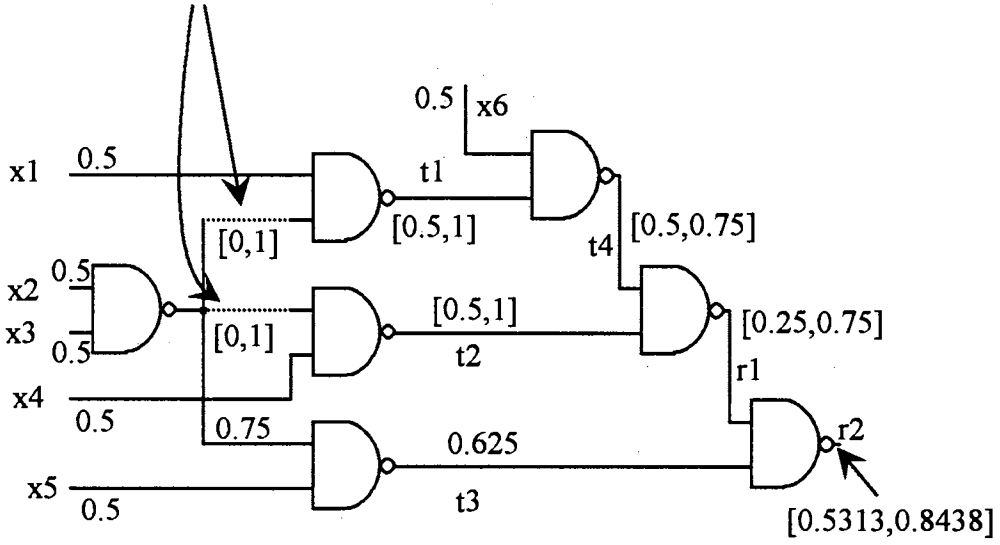
Lower bound for r2's 1-controllability = 1 - (0.625)(0.75) = 0.5313

Upper bound for r2's 1-controllability = 1 - (0.625)(0.25) = 0.8438



(a)

Assign these 1-controllability ranges after cutting the fanout branches.
Calculate 1-controllabilities of other signals in terms of these ranges.



(b)

Figure 3.17. Illustration of Cutting Algorithm (Example 3.9). a) Original circuit; b) Computation of 1-controllability ranges for r1 & r2

Note that if one forms a truth table of six inputs x1, x2, x3, x4, x5, and x6, and then determines the number of times r1, and r2 in Figure 3.17(a) are set to 1 with no fanout branches cut, one will find that the exact 1-controllabilities for r1, and r2 are 0.5938, and 0.6406, respectively. This shows that the lower bound values for r1 and r2 evaluated above are indeed less than the exact values found when the cut fanout branches are restored.

Since the Cutting Algorithm turns a general combinational circuit into a fanout-free circuit, the estimate of the lower bound of the 1-controllability of a line is performed directly. However, very often, the estimate is zero although the true value is much larger [Chakravarty and Hunt 1990].

3.3.2 The PREDICT Algorithm

PREDICT (Probabilistic Estimation of Digital Circuit Testability) [Seth et al. 1985] is a graph-based method for computing the controllabilities of the lines in a combinational circuit. PREDICT includes three different techniques. The first one is an exact computation, and the last two are approximation methods.

Seth uses the notion of a "supergate." A supergate of a signal line g, an output of some logic gate, denoted as SG(g), is defined as a sub-circuit with independent inputs feeding g. This sub-circuit, by definition, must be internally fanout-free except fanout inputs are allowed, has all of its inputs logically independent, and possesses only one output. Figure 3.18 shows the definition of several example supergates.

Basically, PREDICT divides the circuit under consideration into a number of supergates, then represents the original circuit with a directed graph whose nodes are the supergates, and whose edges are the interconnections between the supergates with the arrows pointing in the direction of flows of the signals (see Figure 3.18). Note that a supergate itself can be represented as a directed graph whose nodes are the logic gates, and whose edges are the interconnections between the gates. The controllabilities of the signal lines are then computed within one supergate at a time. The overall order of the

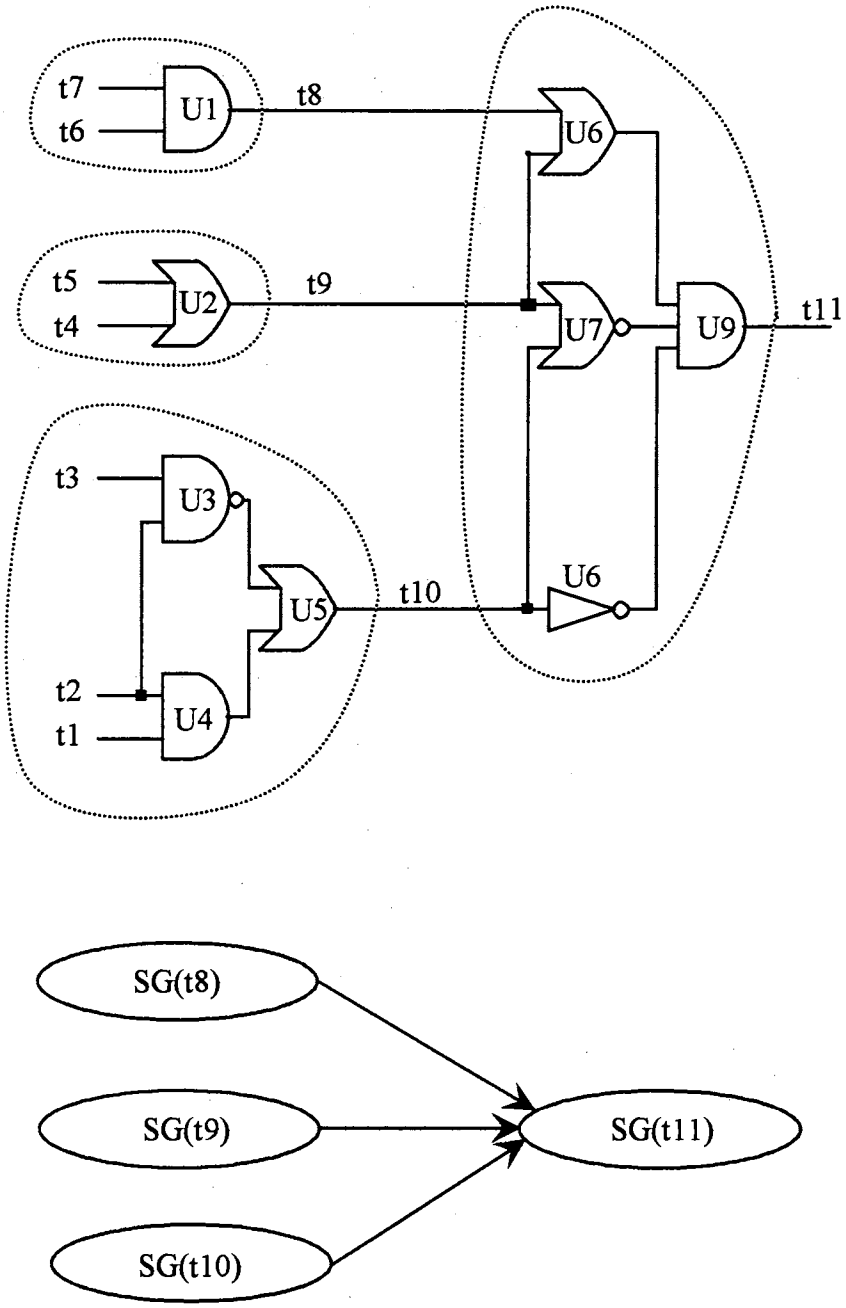


Figure 3.18. Supergates and corresponding circuit graph

computation is according to the interrelationships between the supergates, i.e., supergates with known input controllabilities are considered first. The first of the three methods (an exact method) presented by Seth is reviewed in the following.

For a supergate with no fanout inputs, the computation of the 1-controllabilities in that supergate is straightforward by using Equations 3.1-3.

For a supergate which has K fanout inputs $F_1, F_2, ..., F_K$, the exact computation of the 1-controllability, Prob{SG(g) = 1}, where g is the output line of the supergate, requires an exhaustive enumeration of 2^K K-tuples. In Figure 3.18, K = 0, 0, 1, and 2 for SG(t8), SG(t9), SG(t10), and SG(t11), respectively. The Prob{SG(g) = 1} is defined as

$$\text{Prob}\{SG(g) = 1\} = \sum_{i=1}^{2^K} \text{Prob}\{SG(g) = 1 \cap H_i\} \quad (3.4)$$

where H_i is the i^{th} K-tuple of the fanout inputs to the supergate. Note that, for each H_i, the computation of Equation 3.4 also requires the fixed controllabilities of all non-fanout inputs.

For SG(t10) in Figure 3.18,

$H_1 = (t2 = 0)$

$H_2 = (t2 = 1)$

and for SG(t11),

$H_1 = (t9 = 0, t10 = 0)$

$H_2 = (t9 = 0, t10 = 1)$

$H_3 = (t9 = 1, t10 = 0)$

$H_4 = (t9 = 1, t10 = 1)$

Using the definition of the conditional probability, Equation 3.4 is rewritten as

$$\text{Prob}\{SG(g) = 1\} = \sum_{i=1}^{2^K} \text{Prob}\{SG(g) = 1 \mid H_i\} \times \text{Prob}\{H_i\} \quad (3.5)$$

where the conditional term Prob{SG(g) = 1 | H_i} for each i can be computed directly for any gate using Equations 3.1-3, and

$$\begin{aligned} \text{Prob}\{H_i\} &= \text{Prob}\{F_1 = v \cap ... \cap F_K = v\} \\ &= (\text{Prob}\{F_1 = v\}) \ ... \ (\text{Prob}\{F_K = v\}), \ v \in \{0, 1\} \end{aligned} \quad (3.6)$$

with the assumption that $F_1, F_2, ..., F_K$ are independent.

As an example, Prob{SG(t10) = 1} in Figure 3.18 is found as follows.

Prob{SG(t10) = 1} = 1 - Prob{SG(t10) = 0}

$$\text{Prob}\{SG(t10) = 0\} = (\text{Prob}\{SG(t10) = 0 \mid H_1\})(\text{Prob}\{H_1\}) + (\text{Prob}\{SG(t10) = 0 \mid H_2\})(\text{Prob}\{H_2\})$$

where $H_1 = (t2 = 0)$, and $H_2 = (t2 = 1)$. Assuming that $\text{Prob}\{t1 = 1\} = \text{Prob}\{t2 = 1\} = \text{Prob}\{t3 = 1\} = 0.5$, one has

$$\text{Prob}\{H_1\} = 0.5$$

$$\text{Prob}\{H_2\} = 0.5$$

$$\text{Prob}\{U3 = 0 \mid H_1\} = 0$$

$$\text{Prob}\{U3 = 0 \mid H_2\} = 0.5$$

$$\text{Prob}\{U3 = 1 \mid H_1\} = 1$$

$$\text{Prob}\{U3 = 1 \mid H_2\} = 0.5$$

$$\text{Prob}\{U4 = 0 \mid H_1\} = 1$$

$$\text{Prob}\{U4 = 0 \mid H_2\} = 0.5$$

$$\text{Prob}\{U4 = 1 \mid H_1\} = 0$$

$$\text{Prob}\{U4 = 1 \mid H_2\} = 0.5$$

$$\begin{aligned}\text{Prob}\{SG(t10) = 0\} &= (\text{Prob}\{SG(t10) = 0 \mid H_1\})(\text{Prob}\{H_1\}) + \\ &\quad (\text{Prob}\{SG(t10) = 0 \mid H_2\})(\text{Prob}\{H_2\}) \\ &= (\text{Prob}\{U3 = 0 \mid H_1\})(\text{Prob}\{U4 = 0 \mid H_1\})(\text{Prob}\{H_1\}) + \\ &\quad (\text{Prob}\{U3 = 0 \mid H_2\})(\text{Prob}\{U4 = 0 \mid H_2\})(\text{Prob}\{H_2\}) \\ &= (0)(1)(0.5) + (0.5)(0.5)(0.5) = 0.125\end{aligned}$$

and

$$\text{Prob}\{SG(t10) = 1\} = 1 - 0.125 = 0.875$$

An obvious drawback with the exact method of the PREDICT Algorithm above is that the computation of Equation 3.5 increases exponentially as the number of fanout inputs increases. To avoid this problem, Seth proposed an approximation method for evaluating the signal controllabilities. The approximation is based on the topological analysis of the supergate. These exact and approximate methods will be further analyzed in Chapter 4.

In 1990, Chakravarty and Hunt argued that the intensive computation of the exact signal controllabilities can be avoided. Their argument relied on the idea that the $\text{Prob}\{H_i\}$ can be zero for some H_i. By employing such a condition, the enumeration of 2^K

K-tuples may become an implicit process. Hence the computation of Equation 3.5 may be reduced. This implicit process is what Chakravarty and Hunt referred to as the "Efficient Enumeration" Algorithm. Nevertheless, it will be shown in Chapter 4 that the argument made by Chakravarty and Hunt is not valid.

3.3.3 Matrix Notation for Computing Exact Controllabilities

For convenience in later discussion and computer simulations, the definition of Equation 3.5 is extended for all signal lines in a supergate (including the supergate's inputs), and its conditional and unconditional terms are rewritten in a matrix notation. Let g be any signal line in a supergate (a signal line can be an input to the supergate), and let C(1,g) denote a $1x2^K$ conditional 1-controllability row-vector whose elements correspond to the conditional terms in Equation 3.5. The row-vector C(1,g) for any line g in a supergate is then defined as

$$C(1,g)=\begin{bmatrix} \text{Prob}\{g=1\} \mid H_1 \\ \text{Prob}\{g=1\} \mid H_2 \\ \bullet \\ \bullet \\ \text{Prob}\{g=1\} \mid H_{2^K} \end{bmatrix}^T \tag{3.7}$$

where $[]^T$ denotes the matrix transpose operator, and H_i ($i = 1 \ldots 2^K$) is the i^{th} K-tuple. As an example, $H_1 = (F_1 = 0, F_2 = 0, \ldots, F_K = 0)$, $H_2 = (F_1 = 0, F_2 = 0, \ldots, F_K = 1)$, etc..

Using Equation 3.7, the corresponding $1x2^K$ conditional 0-controllability row-vector, C(0,g), can be easily found by subtracting every element of C(1,g) from 1.

Let H(F) be a 2^Kx1 fanout probability column-vector, whose elements correspond to the unconditional terms in Equation 3.5. H(F) is defined as

$$H(F) = \begin{bmatrix} \text{Prob}\{H_1\} \\ \text{Prob}\{H_2\} \\ \bullet \\ \bullet \\ \text{Prob}\{H_{2^K}\} \end{bmatrix} \tag{3.8}$$

where Prob{H_i} is specified in Equation 3.6. As an example, Prob{H_1} = Prob{F_1=0} x Prob{F_2=0} x ... x Prob{F_K=0}, and Prob{H_2} = Prob{F_1=0} x Prob{F_2=0} x ... x Prob{F_K=1}, etc..

By using Equations 3.7-8, the 1-controllability of any line g in a supergate is

$$\text{Prob}\{g = 1\} = C(1,g) * H(F) \tag{3.9}$$

where the symbol * denotes the vector inner product. The 0-controllability, Prob{g = 0}, is calculated by replacing C(1,g) in Equation 3.9 with C(0,g), or using the relation

$$\text{Prob}\{g = 0\} = 1 - \text{Prob}\{g = 1\}$$

Recall that the computation of Equation 3.9, for each H_i, requires the controllability values of all non-fanout inputs to the supergate to be included in the evaluation of C(1,g).

Also, the conditional controllability row-vector for the output of a logic gate in a supergate can be expressed in terms of the conditional controllability row-vectors of its input lines:

a) A T-input AND gate with output g:

$$C(1,g) = C(1, \text{input } 1) \otimes C(1, \text{input } 2) \otimes \ldots \otimes C(1, \text{input T}) \tag{3.10}$$

where the symbol ⊗ denotes array element-by-element multiplication. For example, if array A and array B have the same dimensions, then A ⊗ B denotes the array whose elements are the products of the individual elements of A and B.

b) A T-input OR gate with output g:

$$C(0,g) = C(0, \text{input } 1) \otimes C(0, \text{input } 2) \otimes \ldots \otimes C(0, \text{input T}) \tag{3.11}$$

c) A NOT gate with output g:

$$C(1,g) = C(0, \text{input}) \tag{3.12}$$

As an example, Prob{t10 = 1} in Figure 3.18 is calculated again using the matrix notation. Assuming that Prob{t1 = 1} = Prob{t2 = 1} = Prob{t3 = 1} = 0.5 as before, one has from Equation 3.8:

$$H(F) = [\text{Prob}\{H1\}\ \text{Prob}\{H2\}]^T = [0.5\ 0.5]^T$$

To correctly implement the matrix approach for non-fanout inputs such as t1 and t3 in the supergate SG(t10), the conditional 1-controllabilities for t1 and t3 must take the form of 2-dimensional vectors as follows:

$$C(1,t1) = [\text{Prob}\{t1 = 1 \mid t2 = 0\}\ \text{Prob}\{t1 = 1 \mid t2 = 1\}] = [0.5\ 0.5]$$

$$C(1,t3) = [\text{Prob}\{t3 = 1 \mid t2 = 0\}\ \text{Prob}\{t3 = 1 \mid t2 = 1\}] = [0.5\ 0.5]$$

Also,

$$C(1,t2) = [\text{Prob}\{t2 = 1 \mid t2 = 0\}\ \text{Prob}\{t2 = 1 \mid t2 = 1\}] = [0\ 1]$$

$$C(1,U3) = [1\ 1] - C(1,t3) \otimes C(1,t2)$$
$$= [1\ 1] - [0.5\ 0.5] \otimes [0\ 1]$$
$$= [1\ 1] - [0\ 0.5] = [1\ 0.5]$$

$$C(1,U4) = C(1,t2) \otimes C(1,t1)$$
$$= [0\ 1] \otimes [0.5\ 0.5] = [0\ 0.5]$$

$$C(0,t10) = ([1\ 1] - C(1,U3)) \otimes ([1\ 1] - C(1,U4))$$
$$= [0\ 0.5] \otimes [1\ 0.5] = [0\ 0.25]$$

Since $C(1,t10) = [1\ 1] - C(0,t10)$, then

$$C(1,t10) = [1\ 1] - [0\ 0.25] = [1\ 0.75]$$

Therefore, applying Equation 3.9, one has

$$\text{Prob}\{t10 = 1\} = C(1,t10) * H(F) = [1\ 0.75] * [0.5\ 0.5]^T$$
$$= 0.5 + 0.375 = 0.875$$

which is the same result as found with the non-matrix method.

CHAPTER IV

SIGNAL PRIORITIES

4.1 Exact Controllability Computation Using Matrix Notation

For the same circuit seen in [Seth et al. 1985], Example 4.1 below will compute the exact 1-controllability of every signal line in the second supergate of the circuit using the matrix notation.

Example 4.1: For the circuit in Figure 4.1, it is assumed that the supergates have been found by using the procedure described in [Seth et al. 1985]. The exact 1-controllabilities for all signal lines in the 2nd supergate are computed by using matrix notation introduced at the end of Chapter 3. Note that K=1 in the 2nd supergate.

First, assume that all the fanout-free primary inputs have probabilities of occurrence of 0.5. That is, Prob{t1=1} = ... = Prob{t6=1} = 0.5. This implies Prob{t7=0} = 0.25 and Prob{t7=1} = 0.75. Using these known quantities and Equations 3.7-10, the conditional 1-controllabilities vectors for the lines in the 2nd supergate are

$$C(1,t1) = [\text{Prob}\{t1 = 1\} \mid t7 = 0 \quad \text{Prob}\{t1 = 1\} \mid t7 = 1]$$
$$= [0.5\ 0.5]$$
$$C(1,t2) = C(1,t5) = C(1,t6) = C(1,t1) = [0.5\ 0.5]$$
$$C(1,t7) = [\text{Prob}\{t7 = 1\} \mid t7 = 0 \quad \text{Prob}\{t7 = 1\} \mid t7 = 1]$$
$$= [0\ 1]$$
$$C(0,t8) = C(1,t2) \otimes C(1,t7) = [0.5\ 0.5] \otimes [0\ 1] = [0\ 0.5]$$

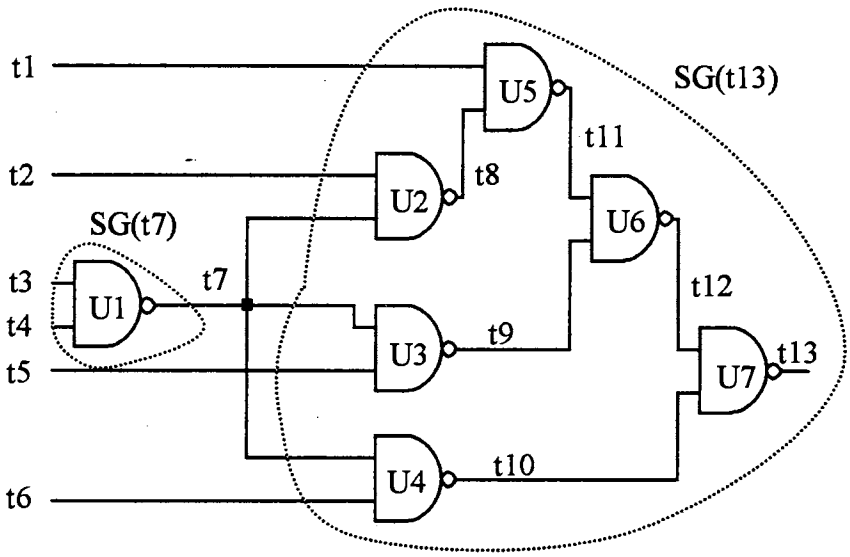


Figure 4.1. Circuit for Example 4.1

$C(1,t8) = [1\ 1] - C(0,t8) = [1\ 0.5]$

$C(1,t9) = C(1,t10) = C(1,t8) = [1\ 0.5]$

$C(0,t11) = C(1,t8) \otimes C(1,t1)$
$= [1\ 0.5] \otimes [0.5\ 0.5] = [0.5\ 0.25]$

$C(1,t11) = [1\ 1] - C(0,t11) = [0.5\ 0.75]$

$C(0,t12) = C(1,t11) \otimes C(1,t9)$
$= [0.5\ 0.75] \otimes [1\ 0.5] = [0.5\ 0.375]$

$C(1,t12) = [1\ 1] - [0.5\ 0.375] = [0.5\ 0.625]$

$C(0,t13) = C(1,t12) \otimes C(1,t10)$
$= [0.5\ 0.625] \otimes [1\ 0.5] = [0.5\ 0.3125]$

$C(1,t13) = [1\ 1] - [0.5\ 0.3125] = [0.5\ 0.6875]$.

From Equation 3.8,

$H(F) = [\text{Prob}\{t7 = 0\}\ \text{Prob}\{t7 = 1\}]^T = [0.25\ 0.75]^T$

and from Equation 3.9, the exact 1-controllabilities are

$\text{Prob}\{t8 = 1\} = C(1,t8) * H(F) = [1\ 0.5] * [0.25\ 0.75]^T$
$= 0.25 + 0.375 = 0.625$

$\text{Prob}\{t9 = 1\} = \text{Prob}\{t10 = 1\} = \text{Prob}\{t8 = 1\} = 0.625$

$\text{Prob}\{t11 = 1\} = C(1,t11) * H(F)$
$= [0.5\ 0.75] * [0.25\ 0.75]^T = 0.6875$

$\text{Prob}\{t12 = 1\} = C(1,t12) * H(F)$
$= [0.5\ 0.625] * [0.25\ 0.75]^T = 0.5938$

$\text{Prob}\{t13 = 1\} = C(1,t13) * H(F)$
$= [0.5\ 0.6875] * [0.25\ 0.75]^T = 0.6406$.

4.2 Current Methods for Approximating Controllabilities

4.2.1 Distance-based Approximation Method in PREDICT

As with the Cutting Algorithm in Section 3.3.1, the objective of the distance-based approximation method is to approximate the 1-controllability of a reconvergent signal in a supergate (i.e., output of the supergate).

Let t be the reconvergent signal whose controllabilities are to be approximated, and let L_m be the maximum distance measured backward from t toward the first fanout point in terms of signal lines. L_m is counted from the reconvergent line farthest to the first fanout point. The distance-based approximation method is described as follows:

a) Evaluate the 1-controllabilities for all non-reconvergent signals in the supergate using Equations 3.1-3. These signals carry no effect from the fanout signals.

b) Choose a value $L < L_m$, where L is counted from the reconvergent line closest to the first fanout point, and evaluate the 1-controllability of t by using either one of the two following rules:

b.1) When none of the possible paths of L distance measured from t toward the fanout point meet at a fanout point, use Equations 3.1-3 to evaluate directly the 1-controllability of t as if t carried no effect from the fanout.

b.2) When two or more paths of L distance measured from t toward the fanout point meet at a fanout point, apply Equation 3.5 to evaluate the 1-controllability of t with the assumption that the fanout effect is shared only by those paths that meet at the fanout point. Treat those paths that do not meet at the fanout point as if they carry no effect from the fanout.

Seth justified this approximation method by arguing that when the circuit's size is large, the fanout effect ignored on those paths that do not meet at the fanout point would likely become weakened due to the influence of other signals. Thus, under such a condition, those paths can be considered independent of the fanout inputs. This approximation method is now illustrated by an example.

Example 4.2: For a comparison purpose, the same circuit in Example 4.1 is used. Since the longest path (path t7-t8-t11-t12) from t13 to the fanout point is 4 signal lines long, then $L_m = 4$. Using Step a, the 1-controllabilities for all non-reconvergent signals in the second supergate are:

Prob{t1=1} = Prob{t2=1} = Prob{t5=1} = Prob{t6=1} = 0.5

Prob{t7=1} = 0.75

Prob{t8=1} = 1 - (Prob{t2=1})(Prob{t7=1})

= 1 - (0.5)(0.75) = 0.625

Prob{t9=1} = Prob{t10=1} = Prob{t8=1} = 0.625

Prob{t11=1} = 1 - (Prob{t1=1})(Prob{t8=1})

= 1 - (0.5)(0.625) = 0.6875

For L = 1, all the paths of 1 signal line away from t12 (paths t11 and t9), and from t13 (paths t12 and t10) do not meet at t7, so Step b.1 is applied and one has

Prob{t12=1} = 1 - (Prob{t11=1})(Prob{t9=1})

= 1 - (0.6875)(0.625) = 0.5703

Prob{t13=1} = 1 - (Prob{t10=1})(Prob{t12=1})

= 1 - (0.625)(0.5703) = 0.6436.

For L = 2, only one of the paths of 2 signal lines away from t12 (path t7-t9), and from t13 (path t7-t10) meet at t7, therefore, the approximate values remain the same as in the case L = 1.

For L = 3, all the paths of 3 signal lines away from t12 meet at t7, therefore, Step b.2 is applied with the assumption that paths t7-t8-t11 and t7-t9 both share the fanout effect, and the 1-controllability of t12 will be the same as that found in Example 4.1 (Prob{t12 = 1} = 0.5938).

For signal t13, not all of its paths meet at t7 (i.e., the missing path is t8-t11-t12). Therefore, Step b.2 is applied with the assumption that only paths t7-t10 and t7-t9-t12 share the fanout effect. The 1-controllability of t13 is found as follows:

As found earlier, the exact 1-controllability of t11 was 0.6875. Since the path t8-t11-t12 carries no fanout effect, then

C(1,t11) = [0.6875 0.6875]

On paths t7-t10 and t7-t9-t12, from Example 4.1, one has

C(1,t9) = [1 0.5]

$C(1,t10) = [1\ 0.5]$

also, from Example 4.1,

$H(F) = [0.25\ 0.75]^T$

Therefore,

$C(1,t12) = [1\ 1] - C(1,t9) \otimes C(1,t11)$
$= [1\ 1] - [1\ 0.5] \otimes [0.6875\ 0.6875]$
$= [0.3125\ 0.6563]$

$C(1,t13) = [1\ 1] - C(1,t10) \otimes C(1,t12)$
$= [1\ 1] - [1\ 0.5] \otimes [0.3125\ 0.6563]$
$= [0.6875\ 0.6719]$

$Prob\{t13 = 1\} = C(1,t13) * H(F)$
$= [0.6875\ 0.6719] * [0.25\ 0.75]^T$
$= 0.6758$

For L = 4, all the paths of 4 signal lines away from t12, and from t13 meet at t7. Therefore, Step b.2 is applied, and the 1-controllabilities of t12 and t13 will be the same as those found in Example 4.1.

By comparing results in Example 4.1 and Example 4.2, one can see that the approximate values are close to the exact controllabilities. However, as stated by Seth, the approximation error on a signal line does not converge to the exact value monotonically as L increases. Moreover, the error fluctuates unpredictably as seen in Examples 4.1-2 (lines t12 and t13). That is, for a signal line, there is no assurance that the approximate 1-controllability will be larger or smaller than the exact one.

4.2.2 "Efficient Enumeration" Algorithm

The "Efficient Enumeration" Algorithm in [Chakravarty and Hunt 1990] relied on a possibility that some of the probabilities, $Prob\{H_i\}$, in Equation 3.8 are zero. Using this situation as the key to control the enumeration, these researchers expected the computation of Equation 3.9 would be reduced. However, it is found by this research that such an argument is not valid for two reasons:

a) For a supergate whose fanout inputs F_1, F_2, ..., F_K are all primary inputs to the circuit, the term Prob{H_i} can never be zero for any i simply because Prob{F_k=0} = Prob{F_k=1} = 0.5 for all i (k = 1 ...K). Therefore, in this case, the performance of the "Efficient Enumeration" Algorithm is the same as that of the exact method previously reviewed, which requires an exhaustive enumeration to evaluate Equation 3.9.

b) For a supergate whose fanout inputs come from other supergates (see Figure 4.1), the probability distributions of these fanout inputs are not necessarily identical as are those in case (a) above. However, will any of those Prob{H_i} ever be zero? The answer is no since this situation implies that some supergate can output only one logic level. By definition, such a supergate is a redundant circuit that can always be replaced by a constant signal [Hayes 1976]. According to Hayes, circuit redundancy, in many situations, is unintentional and undesirable because it will make some faults undetectable. With this in mind, one can expect that a well-designed circuit will not have any redundancy; therefore, for all practical purposes, one should assume that Prob{H_i} $\neq$ 0 for all i.

4.3 Signal Priorities Development and Computation

There are two types of signal priorities: 0-priority and 1-priority. The definition and use of signal priorities are equivalent to that of signal controllabilities. That is, the priority of a signal is defined as an estimate probability of setting that signal to a specified value when an input pattern is randomly selected and applied to the circuit. When the 0-priority is higher than the 1-priority for a given signal line, it implies that the 0 value is more justifiable than is the 1 value, and vice versa.

However, this research does not introduce signal priorities as approximations to signal controllabilities. Instead, the signal priorities reflect the relationship between the 0-controllability and 1-controllability of a given signal line. That is, most of the time, for a given signal line, the 0-priority will be higher than the 1-priority whenever the

0-controllability is higher than the 1-controllability, and vice versa. As with controllabilities, the signal priorities can be used to effectively guide the justification process so that the number of backtrackings is reduced. The development and computation of signal priorities are described in the following sections.

4.3.1 Minimum-value Distributions of Fanouts

Consider a supergate with K fanout inputs $F_1, F_2, ..., F_K$, and $v \in \{0, 1\}$. Let M_v denote an event which is defined as

$$M_v = \{\text{at least one } F_k = v\} \tag{4.1}$$

where k = 1 ... K. For v = 0, one has

$$\begin{aligned} \text{Prob}\{M_0\} &= \text{Prob}\{\text{at least one } F_k = 0\} \\ &= \text{Prob}\{F_1 = 0 \cup \ldots \cup F_K = 0\} \\ &= 1 - \text{Prob}\{F_1 = 1 \cap \ldots \cap F_K = 1\} \\ &= 1 - [1 - \text{Prob}\{F_1 = 0\}] \times \ldots \times [1 - \text{Prob}\{F_K = 0\}] \end{aligned} \tag{4.2}$$

where the signal controllabilities of F_k for all k are assumed to be independent. When the probability distributions of all F_k are identical, Equation 4.2 becomes

$$\text{Prob}\{M_0\} = 1 - [1 - \text{Prob}\{F_K = 0\}]^K \tag{4.3}$$

The corresponding expressions for $\text{Prob}\{M_1\}$ can be easily obtained by changing the 0's to 1's in each of the equations 4.2 and 4.3. That is,

$$\text{Prob}\{M_1\} = 1 - [1 - \text{Prob}\{F_1 = 1\}] \times \ldots \times [1 - \text{Prob}\{F_K = 1\}] \tag{4.4}$$

and

$$\text{Prob}\{M_1\} = 1 - [1 - \text{Prob}\{F_K = 1\}]^K \tag{4.5}$$

when the probability distributions of F_k are identical. The probability distributions of M_v are termed the minimum-value distributions [Soong 1981] of the fanout input variables.

It is useful to determine the relationship between $\text{Prob}\{M_0\}$and $\text{Prob}\{M_1\}$. When $\text{Prob}\{M_0\} > \text{Prob}\{M_1\}$, from Equation 4.2 and Equation 4.4, one has

$$\begin{aligned} &1 - [1 - \text{Prob}\{F_1 = 0\}] \times \ldots \times [1 - \text{Prob}\{F_K = 0\}] > \\ &\quad 1 - [1 - \text{Prob}\{F_1 = 1\}] \times \ldots \times [1 - \text{Prob}\{F_K = 1\}] \end{aligned} \tag{4.6}$$

Subtracting 1 from both sides of Inequality 4.6 and multiplying both side by -1 yield

$$\begin{aligned} &[1 - \text{Prob}\{F_1 = 0\}] \times \ldots \times [1 - \text{Prob}\{F_K = 0\}] < \\ &\quad [1 - \text{Prob}\{F_1 = 1\}] \times \ldots \times [1 - \text{Prob}\{F_K = 1\}] \end{aligned} \tag{4.7}$$

but $[1 - \text{Prob}\{F_k = 0\}] = \text{Prob}\{F_k = 1\}$; therefore, the Inequality 4.7 can be rewritten as

$$[\text{Prob}\{F_1 = 1\}] \times \ldots \times [\text{Prob}\{F_K = 1\}] < [\text{Prob}\{F_1 = 0\}] \times \ldots \times [\text{Prob}\{F_K = 0\}] \quad (4.8)$$

From Inequalities 4.6 and 4.8, one can see that when $\text{Prob}\{M_0\} > \text{Prob}\{M_1\}$, the probability of the event that all of the fanout inputs is equal to 0 is greater than that of the event that all of the fanout inputs is equal to 1. Also, note that the greater the difference between the two sides of Inequality 4.6, the greater the difference between the two sides of Inequality 4.8. If one performs a similar analysis for the case $\text{Prob}\{M_1\} > \text{Prob}\{M_0\}$, one will find that the opposite of the above result is also true.

This research heuristically extends the above result. That is, when $\text{Prob}\{M_0\} > \text{Prob}\{M_1\}$ with K reasonably large, from Inequality 4.8, one can expect that, among the 2^K K-tuples, the probabilities of occurrences of the K-tuples (i.e., $\text{Prob}\{H_i\}$) that have a majority of 0 signals are likely larger than those of the K-tuples that have a majority of 1 signals, and vice versa. Consequently, when $\text{Prob}\{M_0\} > \text{Prob}\{M_1\}$ and as the difference between the two quantities gets larger, the sum of the $\text{Prob}\{H_i\}$ corresponding to the 0-majority tuples will likely get larger than that of the $\text{Prob}\{H_i\}$ corresponding to the 1-majority tuples, and vice versa.

As an example, suppose that a supergate has 3 fanout inputs F_1, F_2, and F_3 (i.e., K = 3). If $\text{Prob}\{M_0\} > \text{Prob}\{M_1\}$, then the probabilities of occurrences of those 3-tuples that have a majority of 0 such as 000, 001, 010, and 100 will likely be greater than those 3-tuples with a majority of 1's such as 011, 101, 110, and 111. Consequently, as the difference between $\text{Prob}\{M_0\}$ and $\text{Prob}\{M_1\}$ gets larger, the sum of the $\text{Prob}\{H_i\}$ corresponding to the tuples 000, 001, 010, and 100 will likely get larger than that of the $\text{Prob}\{H_i\}$ corresponding to the tuples 011, 101, 110, and 111.

4.3.2 Computation of Signal Priorities

As presented in Section 4.3.1, when $\text{Prob}\{M_0\} > \text{Prob}\{M_1\}$, the quantities $\text{Prob}\{H_i\}$ corresponding to the 0-majority fanout input combinations will likely be larger than those of the 1-majority fanout input combinations, and vice versa. To illustrate this point, two calculations are performed on an arbitrary supergate that has 5 fanout input

variables whose signal probabilities are randomly generated (the internal structure of this supergate is not relevant at this point).

a) The signal probabilities of the fanout inputs are generated such that $Prob\{M_0\} < Prob\{M_1\}$ as seen in Figure 4.2. In this figure, the probabilities $Prob\{H_i\}$ of the 5-tuples are listed in ascending order. The entries in the first column are listed according to the binary values of the tuples in the second column. Note that the probabilities of occurrences of 0-majority tuples are generally small compared to the others. Most (11 of 16) of the 0-majority tuples are shown in the upper half of this figure. The sum of the probabilities of these 0-majority tuples contributes a maximum of about 16% to the computation of Equation 3.9 with the assumption that each of the elements of C(1,g) is equal to 1.

b) The signal probabilities of the fanout inputs are generated such that $Prob\{M_1\} < Prob\{M_0\}$ as seen in Figure 4.3. In this figure, the probabilities $Prob\{H_i\}$ of the 5-tuples are listed in ascending order. Note that the probabilities of occurrences of 1-majority tuples are generally small compared to the others. Most of the 1-majority tuples are shown in the upper half of this figure. The sum of the probabilities of these tuples contributes a maximum of about 15% to the computation of Equation 3.9 with the assumption that each of the elements of C(1,g) is equal to 1.

In either case, one can simply ignore the rows of H(F) that correspond to the small $Prob\{H_i\}$, and the value of $Prob\{g = 1\}$ calculated from Equation 3.9 will not be significantly affected. Additional computational savings can be achieved if the probabilities of the fanout variables are used directly. For example, since $Prob\{F_3 = 0\}$ and $Prob\{F_5 = 0\}$ of Figure 4.2 are very small, tuples with $F_3 = 0$ and $F_5 = 0$ could also be ignored. This additional step cannot be implemented when the probabilities of all of the fanout variables are about 0.5.

Let $H(F)_r$ denote the row-reduced H(F), and let $C(1,g)_r$ denote the column-reduced C(1,g). When one performs the calculation of Equation 3.9 with H(F) replaced by $H(F)_r$ and C(1,g) replaced by $C(1,g)_r$, one has

H_i	$F_1F_2F_3F_4F_5$	Prob{H_i}	Notes
H_{25}	11000	3.0512844e-005	Prob{F_1=0}=.5297
H_9	01000	3.4366715e-005	Prob{F_2=0}=.6711
H_{27}	11010	4.9068790e-005	Prob{F_3=0}=.0077
H_{11}	01010	5.5266337e-005	Prob{F_4=0}=.3834
H_{17}	10000	6.2273493e-005	Prob{F_5=0}=.0668
H_1	00000	7.0138836e-005	Prob{M_0}=.9117
H_{19}	10010	1.0014422e-004	Prob{M_1}=.9999
H_3	00010	1.1279276e-004	
H_{26}	11001	4.2597762e-004	
H_{10}	01001	4.7977997e-004	
H_{28}	11011	6.8502976e-004	
H_{12}	01011	7.7155124e-004	
H_{18}	10001	8.6937534e-004	
H_2	00001	9.7918026e-004	
H_{20}	10011	1.3980734e-003	
H_4	00011	1.5746546e-003	
H_{29}	11100	3.9331278e-003	
H_{13}	01100	4.4298946e-003	
H_{31}	11110	6.3250028e-003	
H_{15}	01110	7.1238711e-003	
H_{21}	10100	8.0270986e-003	
H_5	00100	9.0409471e-003	
H_{23}	10110	1.2908663e-002	
H_7	00110	1.4539069e-002	
H_{30}	11101	5.4908826e-002	
H_{14}	01101	6.1843988e-002	
H_{32}	11111	8.8300836e-002	
H_{16}	01111	9.9453517e-002	
H_{22}	10101	1.1206312e-001	
H_6	00101	1.2621705e-001	
H_{24}	10111	1.8021268e-001	
H_8	00111	2.0297413e-001	

Figure 4.2. Relationship between Prob{M_v} and Prob{H_i}
(Prob{M_1} > Prob{M_0})

H_i	$F_1F_2F_3F_4F_5$	Prob{H_i}	Notes
H_{16}	01111	5.7517402e-004	Prob{F_1=0}=.4175
H_{32}	11111	8.0253458e-004	Prob{F_2=0}=.6868
H_{12}	01011	8.2419662e-004	Prob{F_3=0}=.5890
H_{28}	11011	1.1499933e-003	Prob{F_4=0}=.9304
H_8	00111	1.2611092e-003	Prob{F_5=0}=.8462
H_{24}	10111	1.7596131e-003	Prob{M_0}=.9992
H_4	00011	1.8071087e-003	Prob{M_1}=.8670
H_{20}	10011	2.5214407e-003	
H_{15}	01110	3.1637741e-003	
H_{31}	11110	4.4143825e-003	
H_{11}	01010	4.5335357e-003	
H_{27}	11010	6.3255972e-003	
H_7	00110	6.9367958e-003	
H_{14}	01101	7.6931560e-003	
H_{23}	10110	9.6788421e-003	
H_3	00010	9.9400937e-003	
H_{30}	11101	1.0734184e-002	
H_{10}	01001	1.1023921e-002	
H_{19}	10010	1.3869314e-002	
H_{26}	11001	1.5381568e-002	
H_6	00101	1.6867782e-002	
H_{22}	10101	2.3535448e-002	
H_2	00001	2.4170718e-002	
H_{18}	10001	3.3725162e-002	
H_{13}	01100	4.2316598e-002	
H_{29}	11100	5.9043928e-002	
H_9	01000	6.0637644e-002	
H_{25}	11000	8.4607101e-002	
H_5	00100	9.2782098e-002	
H_{21}	10100	1.2945794e-001	
H_1	00000	1.3295227e-001	
H_{17}	10000	1.8550698e-001	

Figure 4.3. Relationship between Prob{M_v} and Prob{H_i}
(Prob{M_0} > Prob{M_1})

$$P_1(g) = C(1,g)_r * H(F)_r \quad (4.9)$$

$P_1(g)$ is the definition of the 1-priority of a line g, as developed in this research.

The 0-priority, denoted as $P_0(g)$, can be obtained by replacing $C(1,g)_r$ in Equation 4.9 with $C(0,g)_r$. That is,

$$P_0(g) = C(0,g)_r * H(F)_r \quad (4.10)$$

where $C(0,g)_r$ is found by subtracting every element of $C(1,g)_r$ from 1.

Now, assuming that the supergate structure of a circuit has been determined, Procedure 4.1 below describes the basic steps for obtaining the 1-priorities in each of the supergates of the circuit.

Procedure 4.1:

Step 1: With the known probability distribution of the fanout input variables, evaluate $Prob\{M_0\}$ and $Prob\{M_1\}$ from Equation 4.2 and 4.4.

Step 2: Construct $H(F)_r$, and $C(1,g)_r$ according to following rules:

a) If $Prob\{M_0\} > Prob\{M_1\}$, form $H(F)_r$, and $C(1,g)_r$ by enumerating only fanout input combinations that carry a majority of 0's.

b) Elseif $Prob\{M_1\} > Prob\{M_0\}$, form $H(F)_r$, and $C(1,g)_r$ by enumerating only fanout input combinations that carry a majority of 1's.

c) Elseif $Prob\{M_1\} = Prob\{M_0\}$, form $H(F)_r$, and $C(1,g)_r$ by enumerating only fanout input combinations that carry about the same numbers of 0's and 1's. This last case is rarely seen except when the fanout inputs are primary inputs to the circuit.

Step 3: Use $H(F)_r$, and $C(1,g)_r$ found in Step 2 and Equation 4.9 to obtain the 1-priority for each of the lines in the supergate under consideration.

From Procedure 4.1, one can make the following observations:

a) The enumeration process required to calculate Equation 4.9, for all cases in Step 2, is an implicit one (does not require full enumeration).

b) For all cases in Step 2, the computation of Equation 4.9 (in terms of the number of multiplication and addition operations) is reduced approximately by half compared to that of the exact method.

c) For any signal line, the 1-priority (0-priority) is always smaller than the corresponding 1-controllability (0-controllability).

For any given signal line in a supergate, it is useful to develop some general relationships between the 1-controllability and the 1-priority, and between the 1-controllability and the 0-controllability.

Suppose that a supergate has 2 fanout inputs. Let x be any signal line in this supergate, let c_1, c_2, c_3, and c_4 be the elements of C(1,x), and let h_1, h_2, h_3, and h_4 be the elements of H(F). Using Equation 3.9, the 1-controllability and 0-controllability of x are

$$\text{Prob}\{x = 1\} = c_1h_1 + c_2h_2 + c_3h_3 + c_4h_4 \tag{4.11}$$

$$\text{Prob}\{x = 0\} = (1-c_1)h_1 + (1-c_2)h_2 + (1-c_3)h_3 + (1-c_4)h_4 \tag{4.12}$$

Suppose that h_3 is the quantity one omits in calculating the 1-priority of x. From Equation 4.9, one has

$$P_1(x) = c_1h_1 + c_2h_2 + c_4h_4 \tag{4.13}$$

Then, the difference between the 1-controllability and 1-priority of x is

$$\text{Prob}\{x = 1\} - P_{1,x} = c_1h_1 + c_2h_2 + c_3h_3 + c_4h_4 - \{c_1h_1 + c_2h_2 + c_4h_4\} = c_3h_3$$

This difference is maximum when $c_3 = 1$. But

$$h_3 = 1 - \{h_1 + h_2 + h_4\} = 1 - \text{sum}(H(F)_r) \tag{4.14}$$

where the notation sum(A) denotes the algebraic sum of all elements of vector A. If one performs the above the analysis for a larger number of fanout inputs, one will see that the maximum difference defined in Equation 4.14 is independent of the number of the fanout inputs. Therefore, in general, the maximum difference between the 1-controllability and 1-priority of any signal line in the supergate, denoted as D_{max}, is

$$D_{max} = 1 - \text{sum}(H(F)_r) \tag{4.15}$$

This maximum difference will get smaller as the difference between $\text{Prob}\{M_0\}$ and $\text{Prob}\{M_1\}$ gets larger (see Section 4.3.1). As an example, for any signal x in a supergate whose fanout inputs have the probability distributions identical to those in Figure 4.2, D_{max}

of x is about 0.16. If the fanout inputs have the probability distributions of Figure 4.3, where the difference between Prob{M_0} and Prob{M_1} is greater, D_{max} of x is about 0.15.

By following a similar analysis, it can be shown that the maximum difference between 0-controllability and the 0-priority of any signal is the same as that of Equation 4.15.

Let D(x) be the absolute difference between the 1-controllability and 0-controllability of the signal x, then from Equation 4.11 and Equation 4.12, one has

$$\begin{aligned} D(x) &= |c_1h_1 + c_2h_2 + c_3h_3 + c_4h_4 - \{(1-c_1)h_1 + (1-c_2)h_2 + (1-c_3)h_3 + (1-c_4)h_4\}| \\ &= |2c_1h_1 + 2c_2h_2 + 2c_3h_3 + 2c_4h_4 - (h_1 + h_2 + h_3 + h_4)| \\ &= |2c_1h_1 + 2c_2h_2 + 2c_3h_3 + 2c_4h_4 - 1| \end{aligned} \quad (4.16)$$

By using Equation 4.9 in Equation 4.16, one gets

$$D(x) = |2(P_1(x)) + 2c_3h_3 - 1| \quad (4.17)$$

If c_3 is equal to 1, then Equation 4.17 is maximum, and is denoted as $D_{max}(x)$. One can replace h_3 with the result in Equation 4.14. That is,

$$\begin{aligned} D_{max}(x) &= |2(P_1(x)) + 2(1 - sum(H(F)_r)) - 1| \\ &= |2(P_1(x)) - 2(sum(H(F)_r)) + 1| \end{aligned} \quad (4.18)$$

Clearly, as $D_{max}(x)$ gets smaller, both of the 1-controllability and 0-controllability of x approach 0.5.

When $D_{max}(x)$ is large then the difference between the controllabilities is also large, and the priorities will correctly estimate which one of the controllabilities is larger. When $D_{max}(x)$ is small it is unlikely the priorities can accurately predict such a relationship. In this case, a user-defined threshold value V can be used to distinguish between these cases. Let

$$D_{max}(x) \leq V < 1 \quad (4.19)$$

Experience shows that V = 0.3 is effective. This choice is rather heuristic since V represents the maximum difference between the controllabilities of a signal, and if Inequality 4.19 holds for V = 0.3, it means that the true difference will be more likely less than that. This point will be clarified further by examples later in this section.

Since only signal priorities are calculated for guiding the test pattern generation, one needs some way to interpret the result so that one can monitor how closely the signal priorities reflect the relationship between the two controllabilities. Procedure 4.2 below

(the corresponding flowchart is in Figure 4.4) provides useful rules for such a interpretation.

Procedure 4.2:

Step 1) If the 1-priority of signal line x, $P_1(x)$, is equal or greater than 0.5, then the corresponding 1-controllability is greater than 0.5.

Step 2) When the 1-priority $P_1(x)$ is less than 0.5, one cannot definitely state that the corresponding 1-controllability is less than 0.5. Then, perform the following test:

$$P_1(x) + D_{max} \leq 0.5 \tag{4.20}$$

where D_{max} is-defined by Equation 4.15.

2.1) If Inequality 4.20 holds, then the corresponding 1-controllability is less than 0.5. Else,

2.2) If Inequality 4.20 does not hold, then calculate the corresponding 0-priority, $P_0(x)$, from Equation 4.10, and perform the test:

$$0.5 \leq P_0(x) \tag{4.21}$$

2.2.1) If Inequality 4.21 holds, then the corresponding 0-controllability is greater than 0.5. Else, perform the test:

$$P_0(x) + D_{max} \leq 0.5 \tag{4.22}$$

2.2.2) If Inequality 4.22 holds, then the corresponding 0-controllability is less than 0.5. Else, go to Step 3.

Step 3) If Inequality 4.22 does not hold, then one can consider 2 choices:

3.1) Pick a threshold value V as discussed above, evaluate $D_{max}(x)$ in Equation 4.18. If Inequality 4.19 holds with the selected V, then one can assume that both of the controllabilities are about the same. If the inequality does not hold, go to Step 3.2.

3.2) Compare the priorities. If $P_1(x) \geq P_0(x)$, then assume that the 1-controllability is greater than the 0-controllability. Otherwise, assume that the 0-controllability is greater than the 1-controllability.

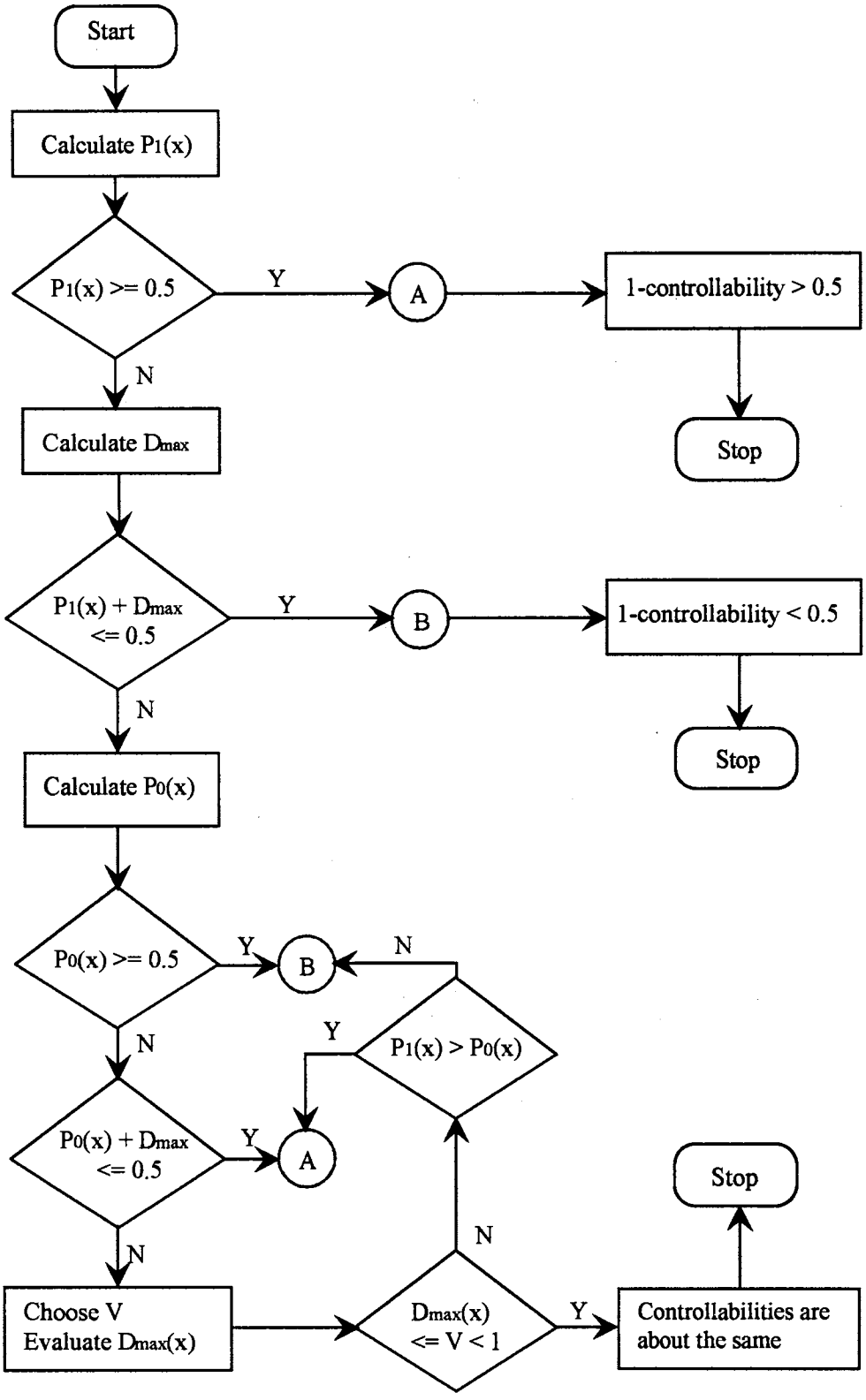


Figure 4.4. Flowchart for Procedure 4.2

In Procedure 4.2, Step 1 and Step 2.1 are rather obvious. In Step 2.2, for a given signal, the computation of the 0-priority is needed only when Inequality 4.20 does not hold. In Step 3, note that the test for Inequality 4.19 is performed only after other tests in Step 1 and Step 2 have been performed. Clearly, at this point, it becomes very likely that neither of the controllabilities is extremely high or low because, if this is not the case, the 1-controllability of x would have been correctly predicted by the first two steps of the procedure. Therefore, if Inequality 4.19 holds with selected value V (0.3 is suggested), it is likely that the controllabilities are about the same.

In the following examples, the computation of the 1-priorities is demonstrated. The controllabilities are also computed for reference purpose but these values would not be available during an actual test generation. The 0-priorities are also shown but they are computed only when needed as discussed in Procedure 4.2.

Example 4.3: Using Procedure 4.1, the 1-priority for every internal line of the circuit in Figure 4.5 is computed. The results for different randomly generated fanout input probability distributions are shown in Figures 4.6-8.

In these figures, one can see that Step 1 of Procedure 4.2 applies to the results for:

a) t1, t2, t4, t5, t6, t9, and t10 of Figure 4.6

b) t1, t2, t4, t5, t8, and t9 of Figure 4.7

c) t1, t2, t4, and t5 of Figure 4.8.

That is, the 1-controllabilities of all of these signals will be correctly predicted to be greater than 0.5.

Step 2.2.1 of Procedure 4.2 applies to the results for:

a) t7, t8, t11, and t13 of Figure 4.6

b) t6, and t13 of Figure 4.7

c) t13 of Figure 4.8

That is, the 1-controllabilities of all of these signals will be correctly predicted to be less than 0.5. The result of the above analysis is summarized in Figure 4.9.

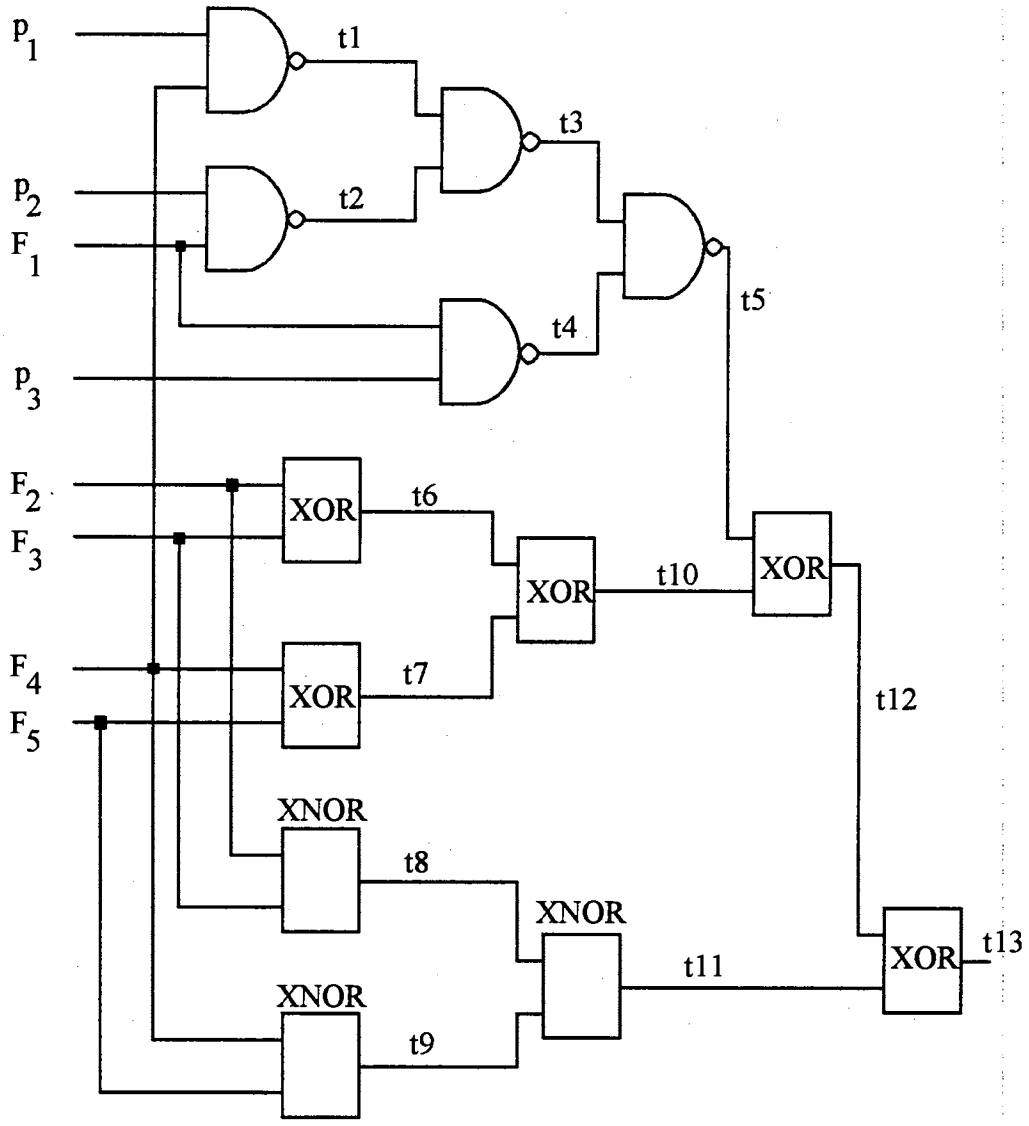


Figure 4.5. Circuit for Example 4.3

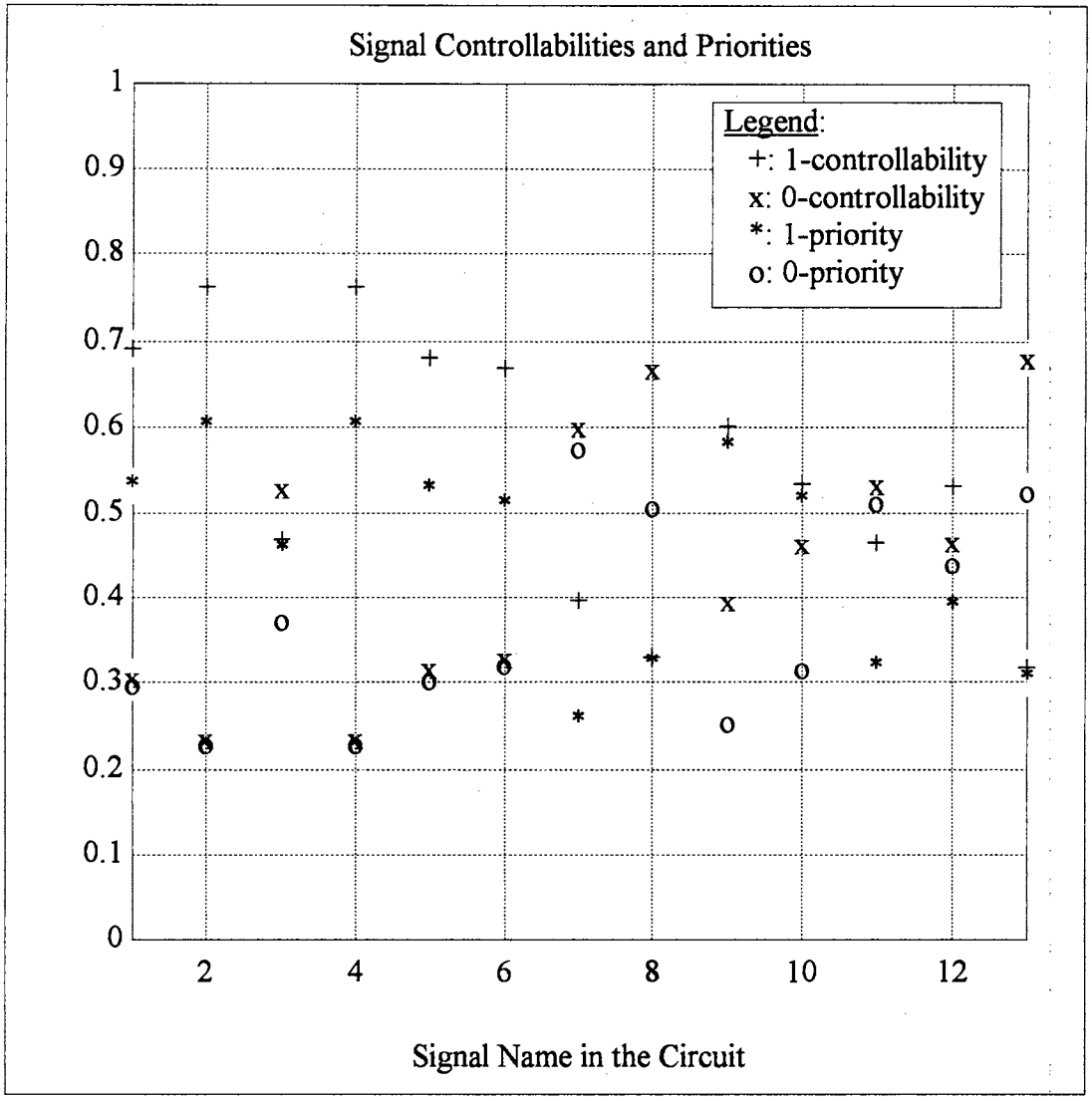


Figure 4.6. Controllabilities and priorities of Example 4.3; Prob{M0}<Prob{M1}; Prob{M0}=0.9117, Prob{M1}=0.9999, Prob{F1=0}=0.5297, Prob{F2=0}=0.6711, Prob{F3=0}=0.0077, Prob{F4=0}=0.3834, Prob{F5=0}=0.0668; D_{max} = 0.1666 .

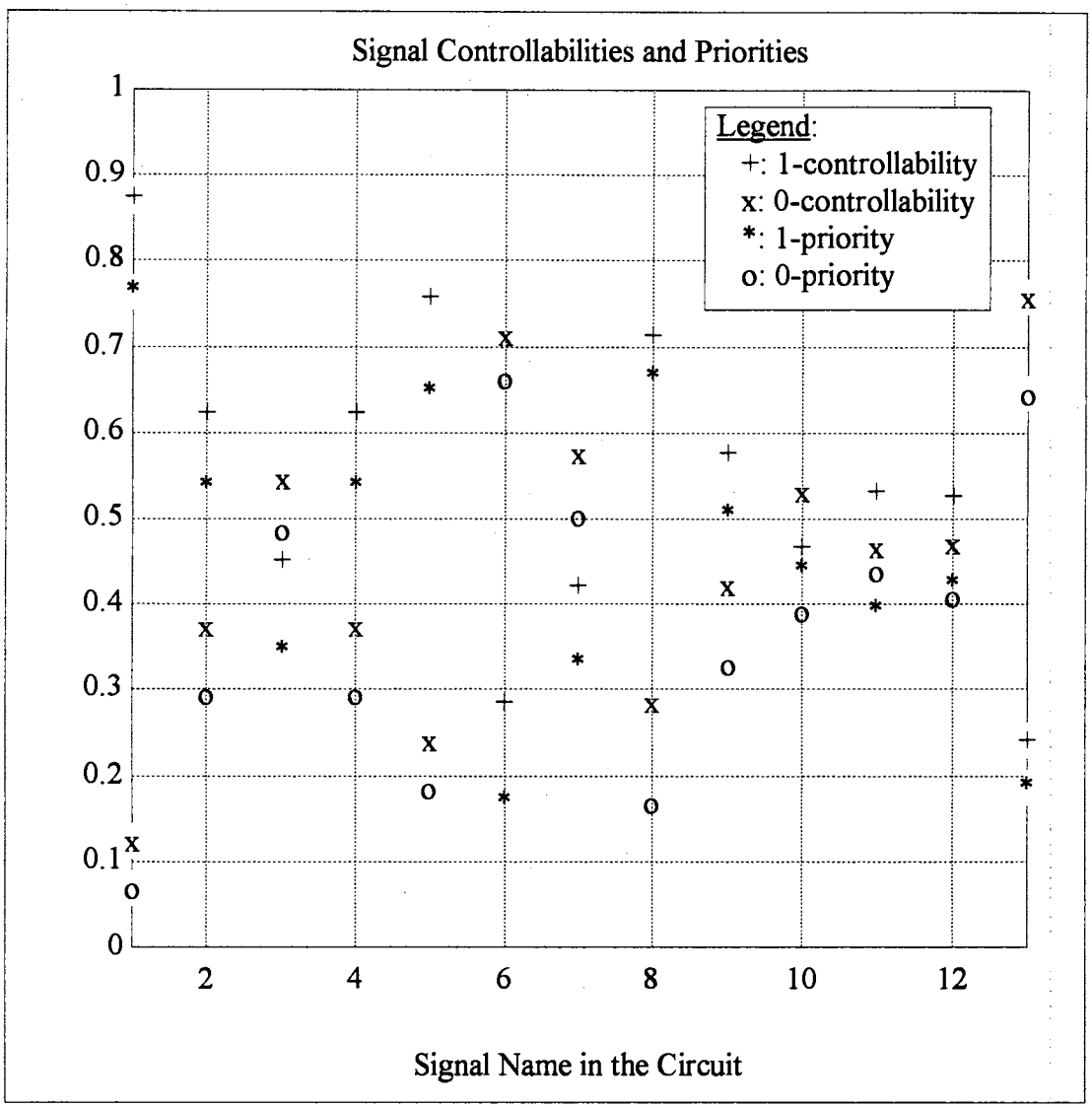


Figure 4.7. Controllabilities and priorities of Example 4.3; Prob{M0}>Prob{M1}; Prob{M0}=0.9997, Prob{M1}=0.9139, Prob{F1=0}=0.247, Prob{F2=0}=0.9826, Prob{F3=0}=0.7227, Prob{F4=0}=0.7537, Prob{F5=0}=0.651; $D_{max} = 0.1656$

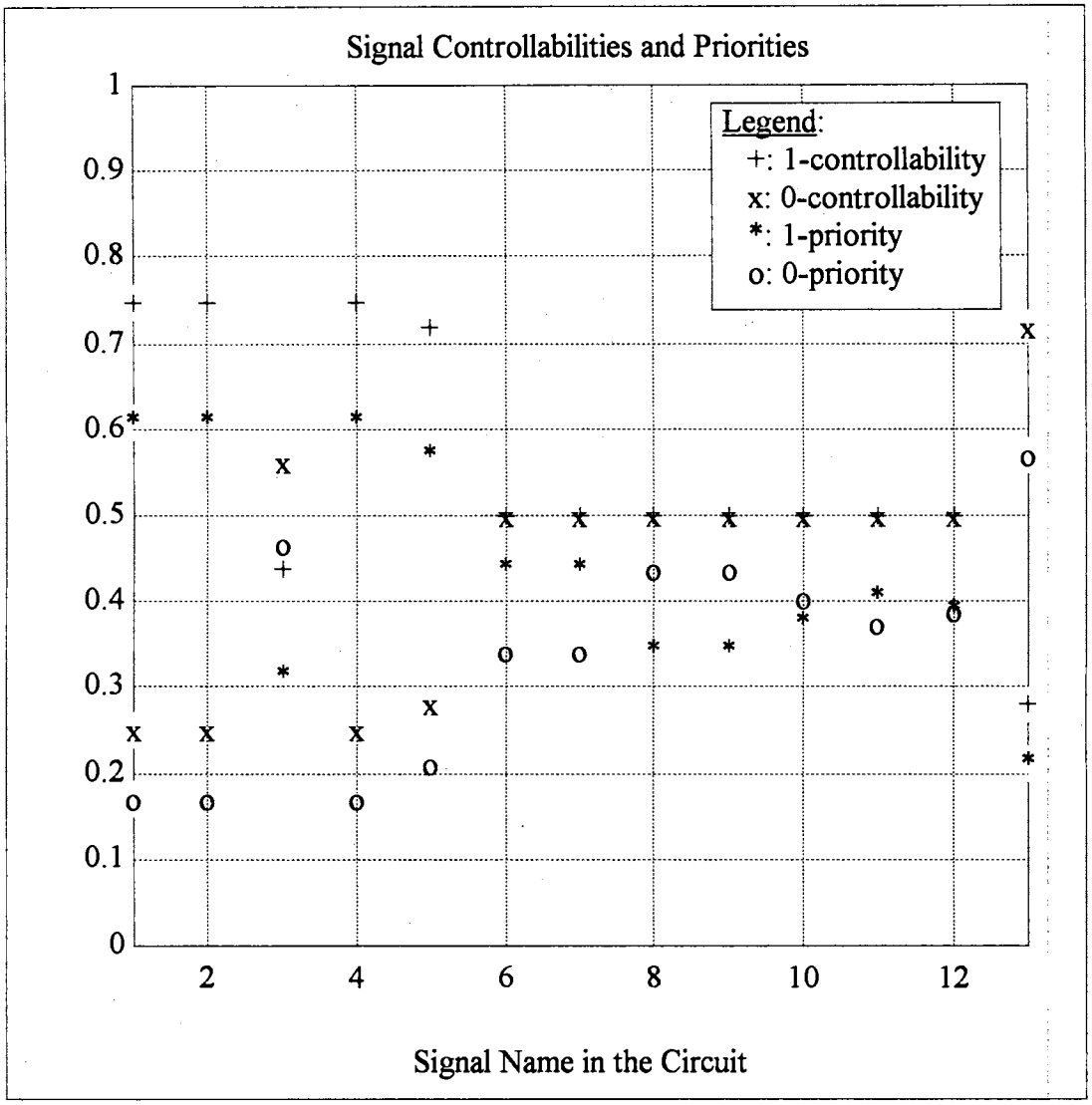


Figure 4.8. Controllabilities and priorities of Example 4.3; Prob{M0}=Prob{M1}; Prob{F1=0}=Prob{F2=0}= Prob{F3=0}= Prob{F4=0}= Prob{F5=0}=0.5; D_{max} = 0.2188

Figure #	1-controllability correctly predicted to be > 0.5 by Step 1 of Procedure 4.2	1-controllability correctly predicted to be < 0.5 by Step 2.2.1 of Procedure 4.2	$D_{max}(x)$	Notes
4.6	t1, t2, t4, t5, t6, t9, t10	t7, t8, t11, t13	$D_{max}(t3) = 0.2732$ $D_{max}(t12) = 0.1332$	Step 3.1 applies to t3, and t12.
4.7	t1, t2, t4, t5, t8, t9	t6, t13	$D_{max}(t3) = 0.0312$ $D_{max}(t10) = 0.2312$ $D_{max}(t11) = 0.1312$ $D_{max}(t12) = 0.1512$	Step 3.1 applies to t3, t10, t11, and t12.
4.8	t1, t2, t4, t5	t13	$D_{max}(t3) = 0.0576$ $D_{max}(t6) = 0.3376$ $D_{max}(t7) = 0.3376$ $D_{max}(t8) = 0.1376$ $D_{max}(t9) = 0.1376$ $D_{max}(t10) = 0.2176$ $D_{max}(t11) = 0.2576$ $D_{max}(t12) = 0.2376$	Step 3.1 applies to t3, t8, t9, t10, t11, and t12. Step 3.2 applies to t6, and t7 (signals with $D_{max}(x) > 0.3$).

Figure 4.9. Overall result for Example 4.3

Step 3.1 of Procedure 4.2 applies to some signals in these figures. $D_{max}(x)$, the left side of Inequality 4.19, is evaluated for these signals (see Figure 4.9). Note that if one chooses V = 0.3, Inequality 4.19 will hold for all signals in Figures 4.6-7, and most of the signals in Figure 4.8. This means that the controllabilities of these signals are predicted to be about the same. For t6 and t7 of Figure 4.8, $D_{max}(t6)$ and $D_{max}(t7)$ are both > V = 0.3. In this case Step 3.2 can apply, and the relative magnitudes of the priorities $P_1(x)$ and $P_0(x)$ can be used. Thus, from Figure 4.8, the 1-controllabilities of both t6 and t7 are predicted to be greater than the 0-controllabilities. Note that Procedure 4.2 fails to identify the fact that the controllabilities of both t6 and t7 are about the same. However, such a failure is not significant since it will not seriously matter which values of t6 and t7 the test generator chooses to guide the signal justification process.

Keep in mind that to accelerate a test generation, only signal priorities (not signal controllabilities) are calculated. These values will be used to guide the fault propagation and signal justification processes to prevent the test generation from becoming exhaustive. For instance, if t12=0 in Figure 4.5 is to be justified, one has two choices for (t5, t10). They are (0, 0) and (1, 1). In this case, and if the test generator is set up to examine the inputs of a logic gate based on a top-down procedure (i.e., the input signal located on the topmost of the schematic of a logic gate is examined first), and if the signal priority values shown in Figure 4.8 is used, t5=1 will be chosen because the 1-priority of t5 is greater than 0.5, which predicts that the actual 1-controllability is greater than 0.5 (see Step 1 of Procedure 4.2).

In summary, the new approach of calculating the signal priorities for predicting the controllabilities gives accurate information on all but 2 out of 39 signals (t6 and t7). If the exact controllabilities were calculated, roughly 100% more computation would have been required.

Example 4.4: This example uses a part of an actual circuit that is now maintained at the Oklahoma City Air Logistics Center, Tinker Air Force Base. The circuit has 7 fanout input variables (see Figure 4.10). The simulated results are shown in

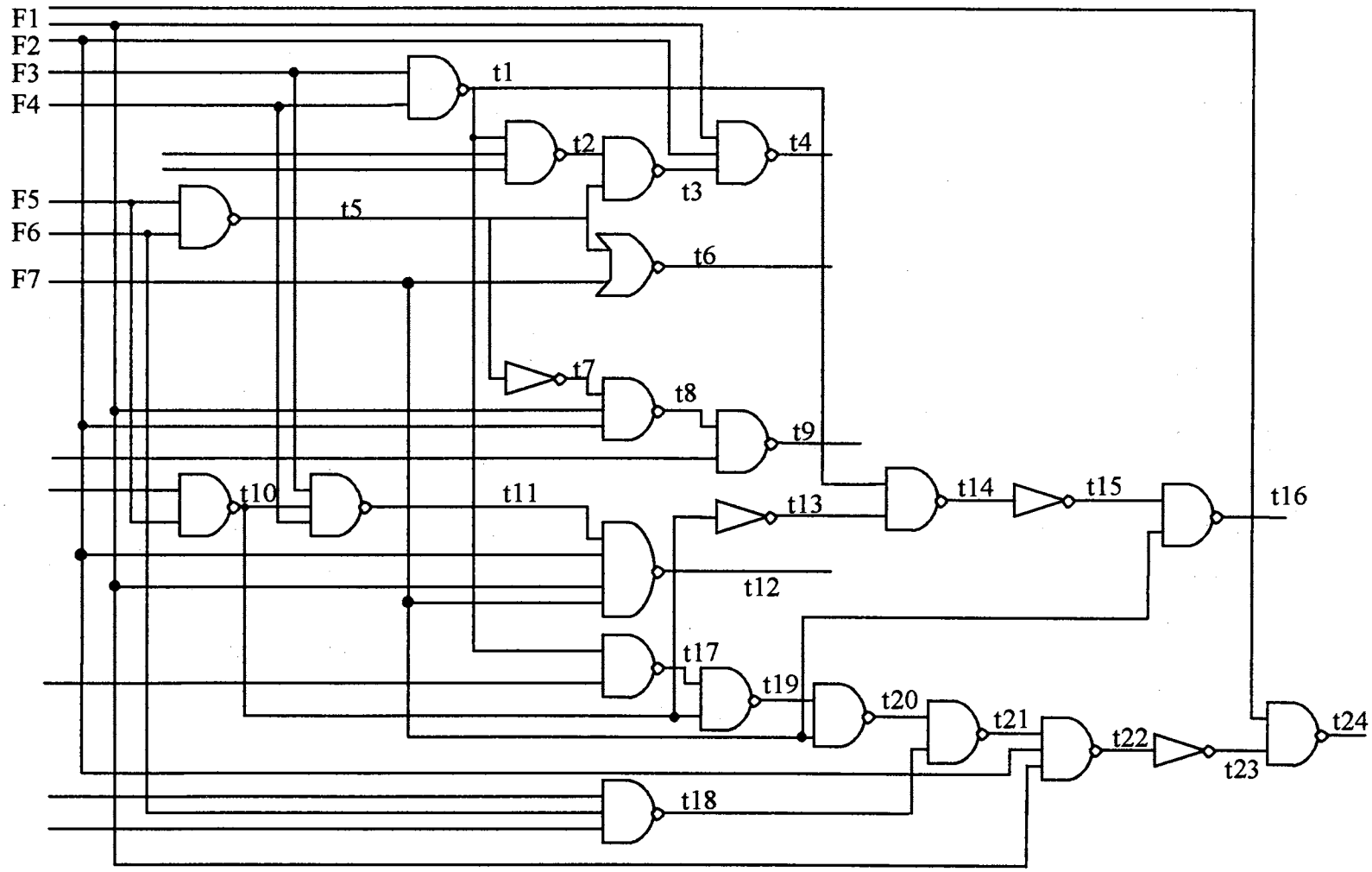


Figure 4.10. Circuit for Example 4.4

Figures 4.11-13. In these figures, similar analysis using Procedure 4.2 is performed as in Example 4.3, and the overall result is shown in Figure 4.14.

4.4 Preferred Overall Procedures of DATPG

Given a digital circuit of either combinational or sequential type and a fault list containing single stuck-at faults, the necessary steps to obtain an input test sequence to detect each of the faults are described in Procedures 4.3-4 below.

Procedure 4.3 (for a combinational circuit):

Step 1: Determine the supergate structure of the circuit, and compute the signal priorities for all internal signals for each supergate of the circuit. The necessary procedure for determining the supergates is adapted from PREDICT [Seth et al. 1985].

Step 2: Using the D-algorithm [Roth 1966], and the 5-value logic, activate and propagate the fault to at least one of the primary outputs. Control the fault propagation and signal justification processes by employing the signal priorities found in Step 1.

Procedure 4.4 (for a sequential circuit):

Step 1: Cut the feedbacks to obtain an equivalent iterative array circuit of arbitrary length p by using the procedure described by Putzolu and Roth [1971] (see Section 3.2).

Step 2: From one copy of the array, determine the supergate structure of the circuit, and compute the signal priorities for every line in this copy of the array. The necessary procedure for determining the supergates is adapted from PREDICT [Seth et al. 1985].

Step 3: Using the D-algorithm [Roth 1966], and the 5-value logic, activate and propagate the fault to at least one of the primary outputs. Control the fault

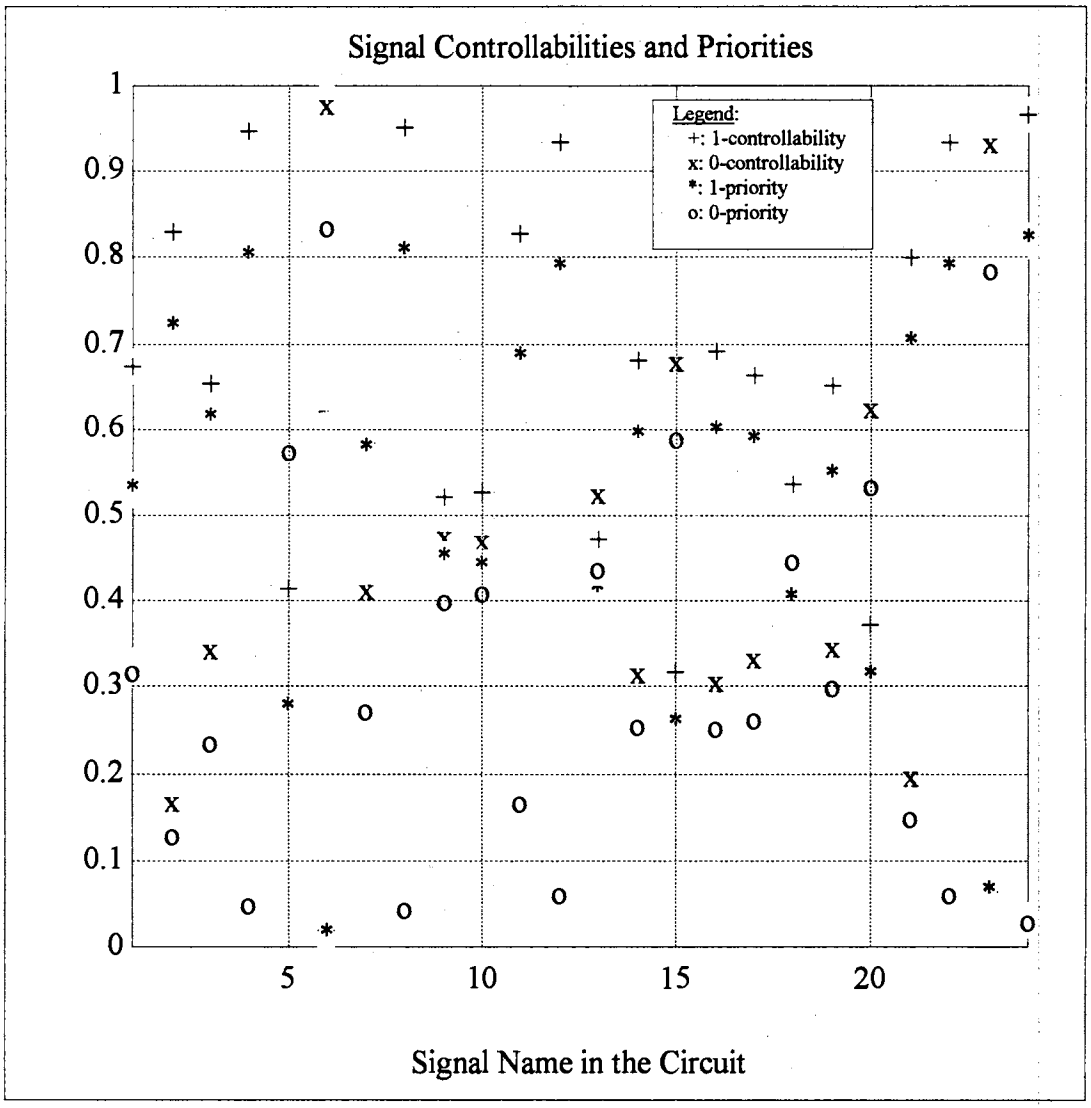


Figure 4.11. Controllabilities and priorities of Example 4.4; Prob{M0}<Prob{M1}; Prob{M0}=0.9854, Prob{M1}=1, Prob{F1=0}=.831, Prob{F2=0}= .0346, Prob{F3=0}=.0535, Prob{F4=0}=.5297, Prob{F5=0}=.6711, Prob{F6=0}=.0077, Prob{F7=0}=.3834;D_{max} = 0.1479

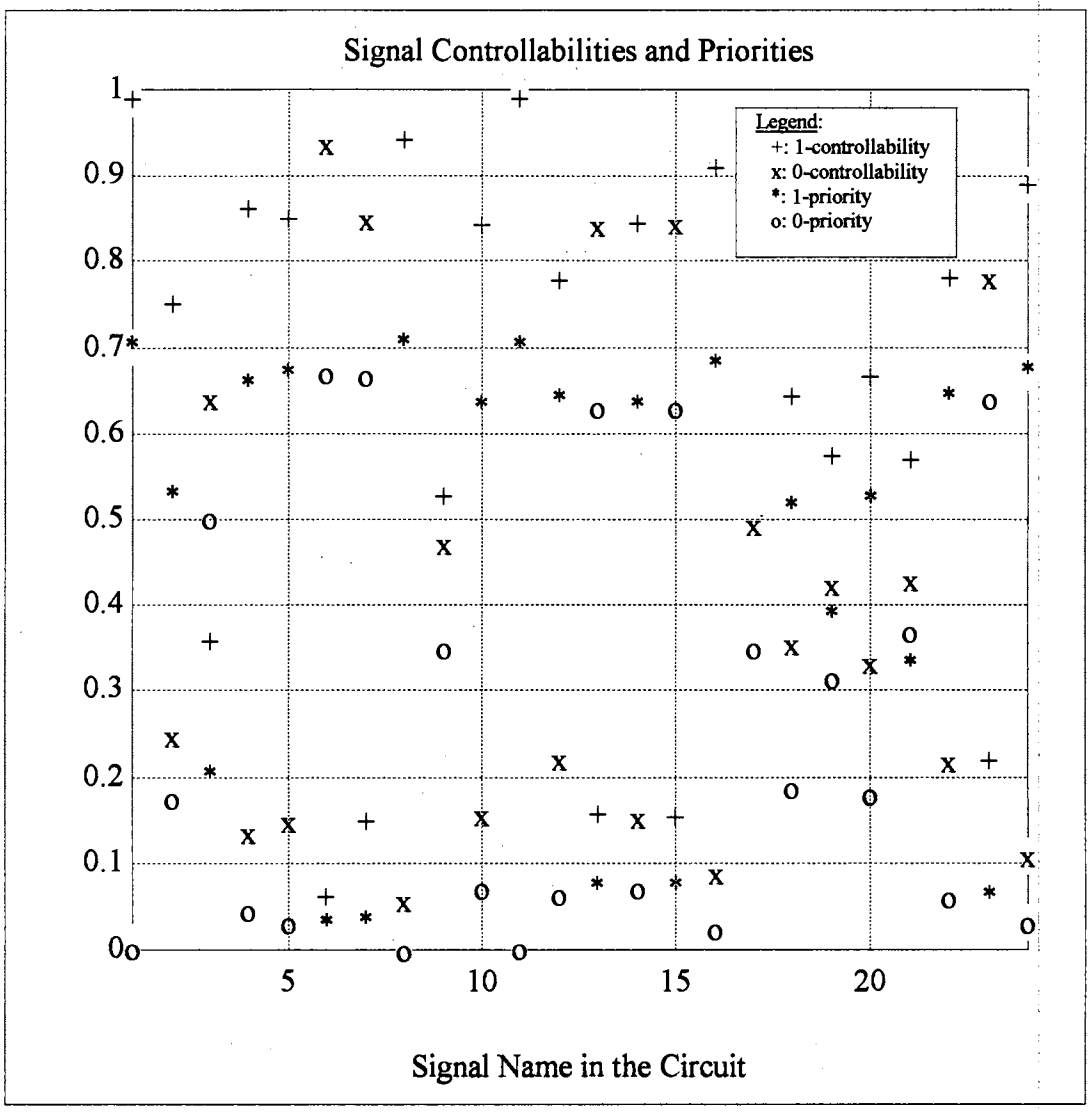


Figure 4.12. Controllabilities and priorities of Example 4.4; Prob{M0}>Prob{M1}; Prob{M0}=0.9996, Prob{M1}=.9953, Prob{F1=0}=.0668, Prob{F2=0}=.4175, Prob{F3=0}=.6868, Prob{F4=0}=.589, Prob{F5=0}=.9304, Prob{F6=0}=.8462, Prob{F7=0}=.5269; $D_{max} = 0.2967$

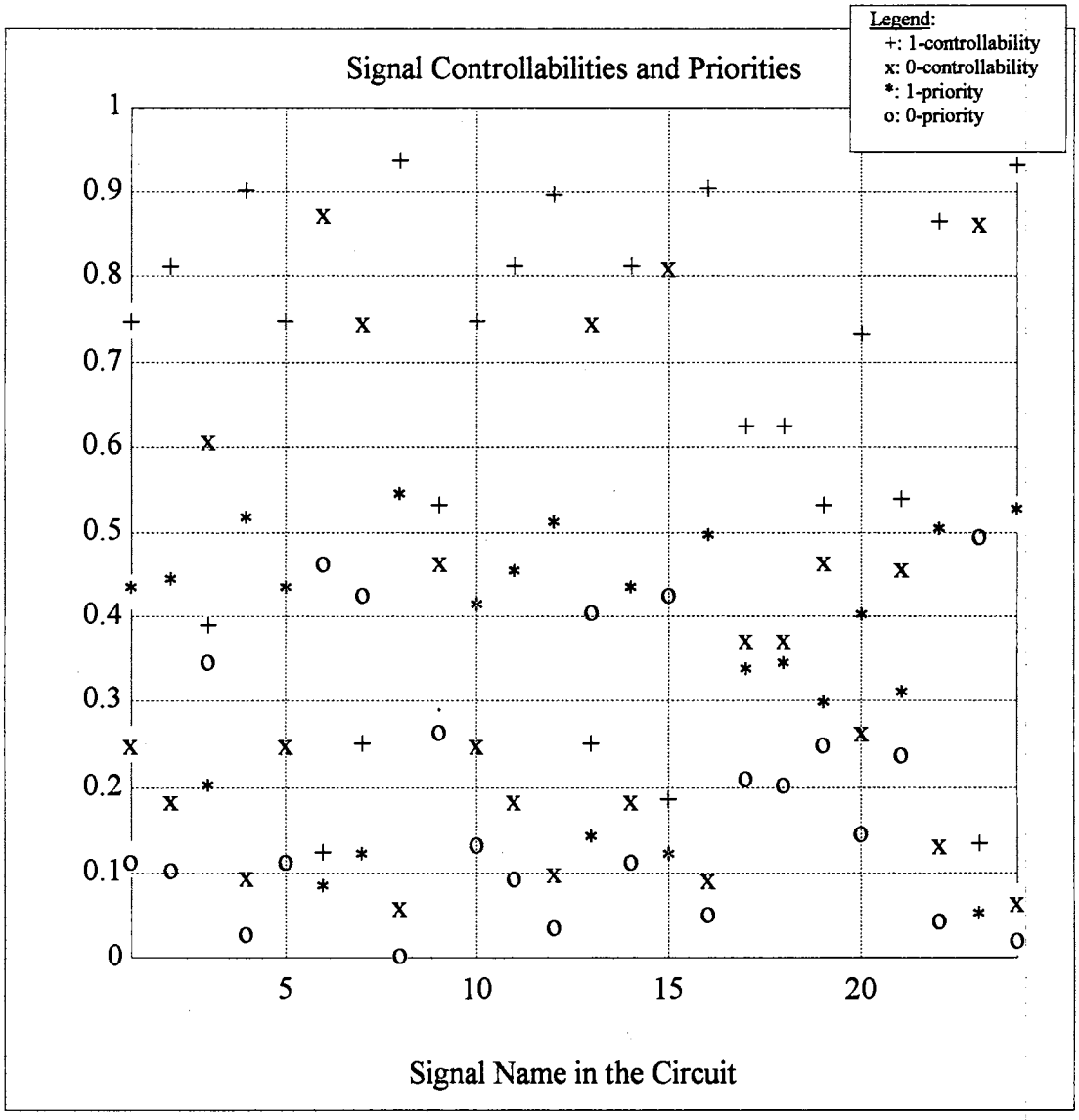


Figure 4.13. Controllabilities and priorities of Example 4.4; Prob{M0}=Prob{M1}; Prob{M0}=0.9922, Prob{M1}= 0.9922,Prob{F1=0} = Prob{F2=0} = Prob{F3=0} = Prob{F4=0} = Prob{F5=0 }= Prob{F6=0} = Prob{F7=0}= 0.5; D_{max} = 0.4531

Figure #	1-controllability correctly predicted to be > 0.5 by Step 1 of Procedure 4.2	1-controllability correctly predicted to be < 0.5 by Step 2.2.1 of Procedure 4.2	$D_{max}(x)$	Notes
4.11	t1, t2, t3, t4, t7, t8, t11, t12, t14, t16, t17, t19, t21, t22, t24	t5, t6, t15, t20, t23	$D_{max}(t9) = 0.2058$ $D_{max}(t10) = 0.1958$ $D_{max}(t13) = 0.1158$ $D_{max}(t18) = 0.1058$	Step 3.1 applies to t9, t10, t13, and t18.
4.12	t1, t2, t4, t5, t8, t10, t11, t12, t14, t16, t18, t20, t22, t24	t3, t6, t7, t13, t15, t23	$D_{max}(t9) = 0.7538$ $D_{max}(t17) = 0.7538$ $D_{max}(t19) = 0.8338$ $D_{max}(t21) = 0.7138$	Step 3.2 applies to t9, t17, t19, and t21.
4.13	t4, t8, t12, t16, t22, t24	t23	$D_{max}(t1) = 0.8062$ $D_{max}(t2) = 0.8062$ $D_{max}(t3) = 0.3062$ $D_{max}(t5) = 0.8062$ $D_{max}(t6) = 0.0862$ $D_{max}(t7) = 0.1262$ $D_{max}(t9) = 0.4462$ $D_{max}(t10) = 0.7262$ $D_{max}(t11) = 0.8062$ $D_{max}(t13) = 0.2062$ $D_{max}(t14) = 0.8062$ $D_{max}(t15) = 0.1262$ $D_{max}(t17) = 0.6062$ $D_{max}(t18) = 0.6062$ $D_{max}(t19) = 0.5062$ $D_{max}(t20) = 0.7062$ $D_{max}(t21) = 0.5062$	Step 3.1 applies to t6, t7, t13, t15. Step 3.2 applies to signals with $D_{max}(x) > 0.3$.

Figure 4.14. Overall result for Example 4.4

propagation and signal justification processes by employing the signal priorities found in Step 2.

Step 4: If necessary, perform the justification process according to justification criteria specified in [Putzolu and Roth 1971] by using the 9-value logic instead of the 5-value logic system. Use the signal priorities computed in Step 2 to guide the fault propagation and signal justification.

Step 5: Perform the Eichelberger's simulation [Eichelberger 1965] to validate the test sequences. Reject any sequence that creates races and hazards.

In the above procedures, the signal priorities are used to replace the controllabilities. For details of other steps, the reader is referred to the respective references summarized in Chapter 3.

Example 4.5: To show additional detail on how the signal priorities are used to accelerate the test generation, Procedure 4.3 was used to detect the sa1 fault on line t13 of the circuit used in Example 4.3 (see Figure 4.4). In this test pattern generation example, there are three cases to be considered:

a) The test generation is performed without using signal priorities. The backtracking effort in this case is unguided and depends only on the topology of the circuit.

b) The test generation is guided by only one signal priority and that priority incorrectly predicts the exact controllability. The signal priority for line t5 is incorrectly used in this example. The backtracking effort in this case is shown to be exhaustive and greater than Case a.

c) The test generation is guided by only one signal priority that correctly predicts the exact controllability. As in Case b, the signal priority of line t5 is used. Compared to the other two cases, the backtracking effort in this case is found to be significantly less.

In all cases, first, the backtracking effort made by the test generator is described in terms of the number of the combinations of all signal lines that the test

generator tries before it finds a primary input combination for detecting the fault. Second, it is assumed that the signal distributions of the fanout inputs are identical; therefore, the signal priority values shown in Figure 4.7 will be used. Refer to the Appendix for detailed results. In the Appendix, Figures A.1-3 show the results for three cases presented below.

Case a: The test generation is performed without using the priority, i.e., there is no guidance. The test generator first sets line t13 to 0, then performs the signal justification toward the circuit's inputs. When a conflict is found, the test generator backtracks toward the output of the circuit to reselect different values to resolve the conflict. The process continues until the test pattern that can set line t13 to 0 is found. The result of the entire process can be followed in Figure A.1 in the Appendix. The desired test pattern is (F1, F2, F3, F4, F5, p1, p2, p3) = (1, 0, 0, 0, 1, 0, 0, 1). The total number of all-signal combinations the test generator tries before the valid input test pattern above is 298. This number depends on the topology of the circuit.

Case b: In this case, t5 will be set to a fixed value to guide the test generation. Suppose that, for some reason, the priority incorrectly predicts the exact 1-controllability of line t5. Instead of using t5=1, the test generator uses t5=0 to guide the search. The result is that the test generation degenerates into an exhaustive process, and finds no input pattern for detecting the sa1 fault on line t13. For this test generation, the total number of all-signal combinations is 576 (see Figure A.2 in the Appendix).

After finding no input pattern to detect the fault on line t13, the test generator, by default, will have to backtrack, switch t5 to 1, and resume the test generation.

Case c: In this case, the test generator uses t5=1 according to the value in Figure 4.7. Since the test generation is correctly guided this time,

the total number of all-signal combinations that the generator tries is 11. This number is the smallest compared to other two cases. The input test pattern required to detect the fault is the same as that of Case a. This test generation can be followed in Figure A.3 in the Appendix.

Example 4.6: Two more test generations are performed in this example. The same circuit, fault, and signal priority values as those in Example 4.5 are used. The difference is that the signal priority of line t9 is used. This example considers two cases: a) t9=0, and b) t9=1. Refer to Figures A.4-5 in the Appendix for detailed results of the two cases presented below.

Case a: With t9=0, as the higher value of 0-priority in Figure 4.7 would suggest, the test generator tries 261 all-signal combinations before it terminates the process (see Figure A.4 in the Appendix). The desired test pattern is (1, 0, 0, 0, 1, 0, 0, 1).

Case b: With t9=1, 455 all-signal combinations are examined before the test generation is finished (see Figure A.5 in the Appendix). The desired test pattern is (1, 0, 1, 0, 0, 0, 0, 1). Note that the solution to this test generation will not be unique. Compared to Case a, more signal combinations are examined with t9 = 1 because the 1-priority of t9 is smaller than its 0-priority as shown in Figure 4.8.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

5.1 Summary and Conclusions

The goal of this research was to seek a more efficient technique for accelerating an existing Automatic Test Pattern Generation (ATPG) algorithm for detecting single stuck-at faults in digital circuits. The effort was directed at reducing the overall testing cost and producing high-quality input test patterns that would detect a high percentage of faults.

Chapters 1 and 2 provided an overview of the test pattern generation process. In Chapter 1, major issues such as deterministic versus pseudorandom ATPG, methods for handling sequential circuits, and research trends in the DATPG were discussed. In Chapter 2, important ideas and tools such as the single stuck-at fault model, various logic systems, signal probabilities, the iterative array model, and the Eichelberger's simulator were reviewed.

Chapter 3 presented a number of DATPG algorithms that had been developed over the years. Because of the increasing complexity of digital systems, current research activity has focused on the acceleration of the existing DATPG algorithms. With this in mind, Chapter 3 investigated the progress that had been made so far. This progress is briefly summarized below.

One of the very first DATPG algorithms called the "D-algorithm" was designed by Roth [1966]. The D-algorithm was considered significant because it was formally proven to be a complete algorithm, which, for a given fault, guaranteed to find an input test pattern if one existed. The basic feature of the D-algorithm was that its fault processing started at the fault's location, and advanced forward to the primary outputs and backward

to the inputs of the circuit. However, for a certain type of combinational circuit, this processing strategy became very inefficient due to the excessive number of backtrackings required during the determination of the desired input test patterns. This problem made the D-algorithm potentially an exhaustive algorithm.

In 1981, an algorithm called "PODEM" (path-oriented decision making) was developed to overcome the deficiency of the D-algorithm [Goel 1981]. Instead of starting the fault processing at the fault's location, PODEM starts at the primary inputs of the circuit and advances toward the primary outputs of the circuit. PODEM also imposes several conditions for early termination of the search process whenever the fault's effect is found not observable.

By employing the above features, Goel showed that, for a certain type of combinational circuit, PODEM potentially could prevent the search for input test patterns from degenerating into an exhaustive process. Thus, the performance of PODEM was more efficient compared to that of the D-algorithm. However, PODEM still faced the same problem as did the D-algorithm for a more general class of combinational circuits.

Two years later, FAN (fanout-oriented test generation algorithm), developed by Fujiwara and Shimono [1983], introduced a new search strategy that further improved the performance of test generation. Based on the fanout structure of the circuit, FAN's search mechanism performs on multiple paths in the circuit to speed up the fault processing. Upon meeting a fanout signal that is fed by a fanout-free sub-circuit and not accessible by the fault effect, FAN postpones the search process for this signal so that it can work on other tasks.

Efforts to accelerate existing DATPG algorithms summarized so far have focused on how to guide the search process so that it would not degenerate into an exhaustive procedure. These acceleration strategies were based on a topological analysis of the circuit. This research has shown that these search strategies also had a weakness. That is, the topological analysis wasted time examining test patterns which might never occur due to their small probabilities of occurrence.

Nevertheless, there were other search acceleration techniques available. These techniques rely on probabilistic measures such as the signal "controllabilities" of signal

lines in the circuit. Agrawal [1985] was one of the pioneers in promoting the probabilistic guidance of the search process in the DATPG. In his paper, titled "Probabilistically Guided Test Generation," Agrawal experimentally showed that signal controllabilities guided the search process more efficiently than did the search strategy of PODEM.

The major problem with Agrawal's probabilistic approach is that the computation of the exact signal controllabilities increases exponentially with the number of fanout input variables of the circuit. Research by [Savir et al. 1984, Seth et al. 1985, and Chakravarty and Hunt 1990] proposed ways to improve the efficiency of signal controllability computation, but the result was still far from being satisfactory. For example, very often, the controllability was severely underestimated by using the method proposed by Savir. Also, the method proposed by Chakravarty and Hunt relied heavily on redundant subcircuits which, as as discussed in this research, will not normally appear in well-designed circuits.

Chapter 4 presented the major contribution of this research. Instead of trying to approximate the signal controllabilities as many other researchers had proposed, this research introduced new parameters called "signal priorities" which serve the same purpose as signal probabilities (i.e., guiding the test generation process), but require significantly less computational effort.

The new signal priorities utilize the "minimum-value distributions" [Soong 1981] of the fanout input variables of the circuit. Based on these distributions, only those input test patterns which contribute a significant weight to the computation of the controllability of a signal line are enumerated; the other input test patterns are totally ignored in the process. As a result, the computation process is obviously not exhaustive.

This research shows that the amount of computation for evaluating a signal priority is about half that of the exact signal controllability. For a large digital circuit with many fanout input variables, such a reduction in the computation can significantly accelerate the overall test pattern generation process. Also, this research has produced two methods for evaluating the maximum difference between the 1-priority and the exact 1-controllability for any given signal, and the maximum difference between the

controllabilities of a signal in terms its 1-priority. These allow a more precise estimate of the exact controllabilities from the priorities.

To demonstrate the computation and utility of the 1-priority, Chapter 4 included several examples. One example is a part of a real circuit which has been maintained at the Oklahoma City Air Logistics Center, Tinker Air Force Base. The results consistently show that signal priorities can replace the traditional exact controllabilities and significantly accelerate the test pattern generation process.

5.2 Future Work

There are at least two issues that should be examined in future research:

1) Although the computation of signal priorities is far more efficient than that of the exact signal controllabilities, this research has shown that the amount of the computation can be further reduced. Experimental results have shown that as the difference between Prob{M_1 } and Prob{M_0} becomes larger, then more rows of the matrix H(F) can be neglected in the computation of signal priorities. Also, the use of the probabilities of individual fanout input variables may improve the process. Therefore, further reduction in the computation may be accomplished if the effect that the minimum-value distributions have on the construction of matrix $H(F)_r$ is more thoroughly analyzed.

2) It was shown that the enumeration of fanout input variables in the computation of the signal priorities was an implicit process in all cases. Nevertheless, it is a concern that the enumeration process may still be lengthy when the number of fanout input variables is extremely large. Therefore, it is worthwhile to look into the complexity of the computation under such a condition.

REFERENCES

[Abramovici et al. 1990] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, New York, 1990.

[Agrawal et al. 1985] V. D. Agrawal, S. C. Seth, and C. C. Chuang, "Probabilistically Guided Test Generation," *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 687-690, July 1985.

[Auth and Schulz 1991] E. Auth, and M. H. Schulz, "A Test-Pattern-Generation Algorithm for Sequential Circuits," *IEEE Design & Test of Computers*, pp. 72-85, June 1991.

[Chakravarty and Hunt 1990] S. Chakravarty, and H. B. Hunt III, "On Computing Signal Probability and Detection Probability of Stuck-at Faults," *IEEE Transactions on Computers*, Vol. 39, No. 11, pp. 1369-1377, November 1990.

[Cheng and Agrawal 1987] K. T. Cheng, and V. D. Agrawal, "Simulation-Based Directed-Search Method for Test Generation," *Proceedings of IEEE International Conference on Computer Design*, Port Chester, NY, pp. 48-51, October 1987.

[Cheng and Chakraborty 1989] W. T. Cheng, and T. J. Chakraborty, "Gentest: An Automatic Test-Generation System for Sequential Circuits," *Computer*, Vol. 22, No. 4, pp. 43-49, April 1989.

[Cheng et al. 1990] K. T. Cheng, V. D. Agrawal, and E. S. Kuh, "A Simulation-Based Method for Generating Tests for Sequential Circuits," *IEEE Transactions on Computers*, Vol. 39, No. 12, pp. 1456-1463, December 1990.

[Cormen et al. 1990] T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, New York, New York, 1990.

[Eichelberger 1965] E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM Journal of Research and Development*, pp. 90-99, March 1965.

[Fujiwara and Shimono 1983] H. Fujiwara, and T. Shimono, "On The Acceleration of Test Generation Algorithms," *IEEE Transactions on Computers*, Vol. C-32, No. 12, pp. 1137-1144, December 1983.

[Goel 1981] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, Vol. C-30, No. 3, pp. 215-222, March 1981.

[Hanna and Horth 1991] J. P. Hanna, and W. J. Horth, "A New Methodology For Test Program Set Generation and Re-hosting," *Conference Record of IEEE Automatic Testing Conference*, Anaheim, California, pp. 279-285, September 24-26, 1991.

[Hayes 1976] J. P. Hayes, "On the Properties of Irredundant Logic Networks," *IEEE Transactions on Computers*, Vol. C-25, No. 9, pp. 884-892, September 1976.

[Jacob and Biswas 1987] J. Jacob, and N. N. Biswas, "GTBD Faults and Lower Bounds on Multiple Fault Coverage of Single Fault Test Set," *Proceedings of IEEE International Test Conference*, Los Alamitos, California, pp. 849-855, September 1987.

[Jain and Agrawal 1985] S. K. Jain, and V. D. Agrawal, "Statistical Fault Analysis," *IEEE Design & Test of Computers*, pp. 38-44, February 1985.

[Mano 1984] M. M. Mano, *Digital Design*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[Muth 1976] P. Muth, "A Nine-Valued Circuit Model for Test Generation," *IEEE Transactions on Computers*, Vol. C-25, No. 6, pp. 630-636, June 1976.

[Putzolu and Roth 1971] G. R. Putzolu, and J. P. Roth, "A Heuristic Algorithm for the Testing of Asynchronous Circuits," *IEEE Transactions on Computers*, Vol. C-20, No. 6, pp. 639-647, June 1971.

[Roth 1966] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM Journal of Research and Development*, Vol. 10, No. 4, pp. 278-291, July 1966.

[Savir et al. 1984] J. Savir, G. S. Ditlow, and P. H. Bardell, "Random Pattern Testability," *IEEE Transactions on Computers*, Vol. C-33, No. 1, pp. 79-90, January 1984.

[Schulz et al. 1987] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *Proceedings of IEEE International Test Conference*, Los Alamitos, California, pp. 1016-1026, September 1987.

[Seth et al. 1985] C. Seth, L. Pan, and V. D. Agrawal, "PREDICT - Probabilistic Estimation of Digital Circuit Testability," *Proceedings of IEEE International Symposium on Fault-Tolerant Computing*, Ann Arbor, Michigan, pp. 220-225, June 1985.

[Soong 1981] T. T. Soong, *Probabilistic Modeling and Analysis in Science and Engineering*, John Wiley & Sons, Inc., New York, New York, 1981.

[Wagner et al. 1987] K. D. Wagner, C. K. Chin, and E. J. McCluskey, "Pseudorandom Testing," *IEEE Transactions on Computers*, Vol C-36, No. 3, pp. 332-343, March 1987.

APPENDIX

TEST PATTERN GENERATION EXAMPLES

(f1,f2,f3,f4,f5,p1,p2,p3,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13) =
(x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,x,0,x,x,x,x,0,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,x,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,0,0,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,1,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0),
(1,0,0,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0),
(x,0,0,1,1,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,1,1,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,1,1,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0), (1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0),
(x,0,0,1,1,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0), (0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,1,0,0,0,0,0), (0,0,0,1,1,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,0,1,0,0,0,0), (0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,1,0,0,0,0,0), (1,0,0,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),

Figure A.1. Test pattern generation without guidance

(1,0,0,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(x,0,0,1,1,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0),
(x,1,1,x,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,1,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (1,1,1,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (x,1,1,1,1,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,1,1,1,1,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0), (x,1,1,1,1,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0), (x,1,1,1,1,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0), (0,1,1,1,1,1,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0,0), (0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,1,1,1,1,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,1,0,0,0,0,0), (0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,0,1,0,0,0,0), (0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(1,1,1,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0), (x,1,1,1,1,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0),

Figure A.1. Test pattern generation without guidance (continued)

(1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,x,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,0,1,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,1,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,0,1,0,1,0,1,1,0,1,0,1,0,0,0,0),
(1,0,1,0,1,0,1,0,1,0,1,1,1,0,0,0,0,0,0,0,0), (1,0,1,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0), (x,0,1,1,0,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,1,0,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,1,0,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,1,0,0,0,0,0), (x,0,1,1,0,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0),
(0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,0,0,0,0,0),
(0,0,1,1,0,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,0,1,0,0,0,0),
(0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0,0,0,0,0),
(1,0,1,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (x,0,1,1,0,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0), (x,1,0,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),

Figure A.1. Test pattern generation without guidance (continued)

(x,1,0,0,1,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,1,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(1,1,0,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(x,1,0,1,0,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,1,0,1,0,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0),
(x,1,0,1,0,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,1,0,0,0,0,0),
(x,1,0,1,0,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0),
(0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,0,1,0,0,0,0),
(0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,1,0,1,0,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0),
(0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,0),
(0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1,0,0,0,0,0), (1,1,0,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0),
(x,1,0,1,0,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,1,x,x,x,x,1,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,0,0,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,0,0,0,1,x,x,x,x,x,0,0,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,x,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0),
(1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,1,0,1,0,0,0)

Figure A.1. Test pattern generation without guidance (continued)

(f1,f2,f3,f4,f5,p1,p2,p3,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13) =

(x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,x,0,x,x,x,x,0,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,x,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,0,0,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,1,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0),
(1,0,0,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0),
(x,0,0,1,1,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,1,1,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,1,1,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0),
(x,0,0,1,1,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0), (0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0), (0,0,0,1,1,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,0,1,0,0,0,0), (0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(x,0,0,1,1,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0),
(x,1,1,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0),

Figure A.2. Test pattern generation with incorrect guidance using t5=0

(0,1,1,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,1,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (1,1,1,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (x,1,1,1,1,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,1,1,1,1,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0), (x,1,1,1,1,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0), (x,1,1,1,1,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0), (0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0), (0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,1,1,1,1,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,1,0,0,0,0,0), (0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,0,1,0,0,0,0), (0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(1,1,1,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0), (x,1,1,1,1,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,x,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,0,1,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,1,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),

Figure A.2. Test pattern generation with incorrect guidance using t5=0 (continued)

(1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0), (1,0,1,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0), (x,0,1,1,0,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,1,0,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,1,0,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,1,0,0,0,0,0), (x,0,1,1,0,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0),
(0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,1,0,0,0,0,0),
(0,0,1,1,0,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,0,1,0,0,0,0),
(0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0,0,0,0,0),
(1,0,1,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (x,0,1,1,0,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0), (x,1,0,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,1,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(1,1,0,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(x,1,0,1,0,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,1,0,1,0,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0),
(x,1,0,1,0,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),

Figure A.2. Test pattern generation with incorrect guidance using t5=0 (continued)

(0,1,0,1,0,1,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,1,0,0,0,0,0),
(x,1,0,1,0,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0),
(0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,0,1,0,0,0,0),
(0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,1,0,1,0,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0),
(0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,0),
(0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1,0,0,0,0,0), (1,1,0,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0),
(x,1,0,1,0,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,1,1,0), (x,x,x,x,x,x,x,x,x,x,x,x,0,x,x,x,x,1,1,1,0),
(x,x,x,x,x,x,x,x,x,x,x,x,0,0,1,x,x,1,1,1,0), (x,0,0,x,x,x,x,x,x,x,x,x,0,0,1,x,x,1,1,1,0),
(x,0,0,0,1,x,x,x,x,x,x,x,0,0,1,x,x,1,1,1,0), (x,0,0,0,1,x,x,x,x,x,x,1,1,0,0,1,x,x,1,1,1,0),
(x,0,0,0,1,x,x,x,0,0,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,0,1,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,0,1,x,x,1,0,0,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,0,1,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0),
(x,0,0,0,1,x,x,x,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,0,1,x,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,0,1,x,x,1,0,1,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,0,1,x,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0),
(x,0,0,0,1,x,x,x,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,0,1,x,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,0,1,0,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,0,1,1,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,0,1,x,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,0,1,0,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,0,1,1,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,0,1,x,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,0,1,0,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,0,1,0,1,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,0,1,0,1,0,1,0,1,1,0,0,1,0,0,1,1,1,0), (1,0,0,0,1,0,1,0,1,0,1,1,0,0,1,1,1,1,1,1,0),
(1,0,0,0,1,1,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,0,1,1,1,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,0,1,1,1,0,1,0,1,1,0,0,1,0,0,1,1,1,0), (1,0,0,0,1,1,1,0,1,0,1,1,0,0,1,1,1,1,1,1,0),
(x,0,0,1,0,x,x,x,x,x,x,0,0,1,x,x,1,1,1,0), (x,0,0,1,0,x,x,x,x,x,1,1,0,0,1,x,x,1,1,1,0),
(x,0,0,1,0,x,x,x,0,0,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,1,0,1,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,x,x,1,0,0,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,1,0,1,x,1,0,0,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,1,0,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,1,0,1,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,0,1,1,1,0), (1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,1,1,1,1,1,0),
(x,0,0,1,0,x,x,x,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,x,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,1,0,1,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,1,0,0,0,1,1,1,0,0,1,x,x,1,1,1,0),

Figure A.2. Test pattern generation with incorrect guidance using t5=0 (continued)

(0,0,0,1,0,1,0,0,0,1,1,1,0,0,1,0,0,1,1,1,0), (0,0,0,1,0,1,0,0,0,1,1,1,0,0,1,1,1,1,1,1,0),
(0,0,0,1,0,1,1,0,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,1,1,0,0,1,1,1,0,0,1,0,0,1,1,1,0),
(0,0,0,1,0,1,1,0,0,1,1,1,0,0,1,1,1,1,1,1,0), (0,0,0,1,0,x,x,1,0,1,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,1,0,1,x,1,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,1,0,1,0,1,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,1,0,1,0,1,0,1,1,1,0,0,1,0,0,1,1,1,0), (0,0,0,1,0,1,0,1,0,1,1,1,0,0,1,1,1,1,1,1,0),
(0,0,0,1,0,1,1,1,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,1,1,1,0,1,1,1,0,0,1,0,0,1,1,1,0),
(0,0,0,1,0,1,1,1,0,1,1,1,0,0,1,1,1,1,1,1,0), (1,0,0,1,0,x,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,1,0,1,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,1,0,1,0,1,0,0,0,1,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,1,0,1,0,0,0,1,1,1,0,0,1,0,0,1,1,1,0), (1,0,0,1,0,1,0,1,0,0,0,1,1,1,0,0,1,1,1,1,1,1,0),
(x,0,0,1,0,x,x,x,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,x,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,1,0,0,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,0,0,1,0,x,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,0,0,1,0,0,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,1,0,x,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,1,0,0,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,0,0,1,0,0,1,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,0,0,1,0,0,1,0,1,0,1,1,0,0,1,0,0,1,1,1,0), (1,0,0,1,0,0,1,0,1,0,1,1,0,0,1,1,1,1,1,1,0),
(x,1,1,x,x,x,x,x,x,x,x,0,0,1,x,x,1,1,1,0), (x,1,1,0,1,x,x,x,x,x,x,0,0,1,x,x,1,1,1,0),
(x,1,1,0,1,x,x,x,x,x,1,1,0,0,1,x,x,1,1,1,0), (x,1,1,0,1,x,x,x,0,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,0,1,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,0,1,x,x,1,0,0,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,0,1,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0), (x,1,1,0,1,x,x,x,0,1,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,0,1,x,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,0,1,x,x,1,0,1,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,0,1,x,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0), (x,1,1,0,1,x,x,x,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,0,1,x,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,0,1,0,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,0,1,1,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,0,1,x,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,0,1,0,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,0,1,1,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,0,1,x,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,0,1,0,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,0,1,0,1,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,0,1,0,1,0,1,0,1,1,0,0,1,0,0,1,1,1,0),
(1,1,1,0,1,0,1,0,1,0,1,1,0,0,1,1,1,1,1,1,0), (1,1,1,0,1,1,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,0,1,1,1,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,0,1,1,1,0,1,0,1,1,0,0,1,0,0,1,1,1,0),
(1,1,1,0,1,1,1,0,1,0,1,1,0,0,1,1,1,1,1,1,0), (x,1,1,1,0,x,x,x,x,x,x,0,0,1,x,x,1,1,1,0),
(x,1,1,1,0,x,x,x,x,x,1,1,0,0,1,x,x,1,1,1,0), (x,1,1,1,0,x,x,x,0,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,1,0,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,1,0,1,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,1,0,x,x,1,0,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,1,0,1,x,1,0,0,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,1,0,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,1,0,1,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,1,0,1,1,0,0,0,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,1,0,1,1,0,0,0,1,1,0,0,1,0,0,1,1,1,0),
(1,1,1,1,0,1,1,0,0,0,1,1,0,0,1,1,1,1,1,1,0), (x,1,1,1,0,x,x,x,0,1,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,1,0,x,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,1,0,1,x,0,0,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,1,0,1,0,0,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,1,0,1,0,0,0,1,1,1,0,0,1,0,0,1,1,1,0),
(0,1,1,1,0,1,0,0,0,1,1,1,0,0,1,1,1,1,1,1,0), (0,1,1,1,0,1,1,0,0,1,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,1,0,1,1,0,0,1,1,1,0,0,1,0,0,1,1,1,0), (0,1,1,1,0,1,1,0,0,1,1,1,0,0,1,1,1,1,1,1,0),
(0,1,1,1,0,x,x,1,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,1,0,1,x,1,0,1,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,1,0,1,0,1,0,1,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,1,0,1,0,1,0,1,1,1,0,0,1,0,0,1,1,1,0),
(0,1,1,1,0,1,0,1,0,1,1,1,0,0,1,1,1,1,1,1,0), (0,1,1,1,0,1,1,1,0,1,1,1,0,0,1,1,1,1,1,1,0),
(0,1,1,1,0,1,1,1,0,1,1,1,0,0,1,0,0,1,1,1,0), (0,1,1,1,0,1,1,1,0,1,1,1,0,0,1,1,1,1,1,1,0),

Figure A.2. Test pattern generation with incorrect guidance using t5=0 (continued)

(1,1,1,1,0,x,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,1,0,1,x,0,0,1,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,1,0,1,0,0,0,1,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,1,0,1,0,0,0,1,1,1,0,0,1,0,0,1,1,1,0),
(1,1,1,1,0,1,0,0,0,1,1,1,0,0,1,1,1,1,1,1,0), (x,1,1,1,0,x,x,x,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,1,0,x,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,1,0,0,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(0,1,1,1,0,x,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0), (0,1,1,1,0,0,x,1,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,1,0,x,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,1,0,0,x,0,1,0,1,1,0,0,1,x,x,1,1,1,0),
(1,1,1,1,0,0,1,0,1,0,1,1,0,0,1,x,x,1,1,1,0), (1,1,1,1,0,0,1,0,1,0,1,1,0,0,1,0,0,1,1,1,0),
(1,1,1,1,0,0,1,0,1,0,1,1,0,0,1,1,1,1,1,1,0), (x,x,x,x,x,x,x,x,x,x,x,0,1,0,x,x,1,1,1,0),
(x,0,1,x,x,x,x,x,x,x,x,0,1,0,x,x,1,1,1,0), (x,0,1,0,0,x,x,x,x,x,x,x,0,1,0,x,x,1,1,1,0),
(x,0,1,0,0,x,x,x,x,x,1,1,0,1,0,x,x,1,1,1,0), (x,0,1,0,0,x,x,x,0,0,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,0,0,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,0,0,x,x,1,0,0,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,0,0,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0), (x,0,1,0,0,x,x,x,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,0,0,x,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,0,0,x,x,1,0,1,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,0,0,x,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (x,0,1,0,0,x,x,x,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,0,0,x,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,0,0,0,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,0,0,1,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,0,0,x,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,0,0,0,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,0,0,1,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,0,0,x,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,0,0,0,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,0,0,0,1,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,0,0,0,1,0,1,0,1,1,0,1,0,0,0,1,1,1,0),
(1,0,1,0,0,0,1,0,1,0,1,1,0,1,0,1,1,1,1,1,0), (1,0,1,0,0,1,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,0,0,1,1,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,0,0,1,1,0,1,0,1,1,0,1,0,0,0,1,1,1,0),
(1,0,1,0,0,1,1,0,1,0,1,1,0,1,0,1,1,1,1,1,0), (x,0,1,1,1,x,x,x,x,x,x,0,1,0,x,x,1,1,1,0),
(x,0,1,1,1,x,x,x,x,x,1,1,0,1,0,x,x,1,1,1,0), (x,0,1,1,1,x,x,x,0,0,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,1,1,1,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,x,x,1,0,0,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,1,1,1,x,1,0,0,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,1,1,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,1,1,1,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,1,1,1,1,0,0,0,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,1,1,1,1,1,0,0,0,1,1,0,1,0,0,0,1,1,1,0),
(1,0,1,1,1,1,1,0,0,0,1,1,0,1,0,1,1,1,1,1,0), (x,0,1,1,1,x,x,x,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,x,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,1,1,1,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,1,1,0,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,1,1,1,1,0,0,0,1,1,1,0,1,0,0,0,1,1,1,0),
(0,0,1,1,1,1,1,0,0,0,1,1,1,0,1,1,1,1,1,1,0), (0,0,1,1,1,1,1,0,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,x,x,1,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,1,1,1,x,1,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,1,0,1,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,1,1,1,0,1,0,1,1,1,0,1,0,0,0,1,1,1,0),
(0,0,1,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,1,1,0), (0,0,1,1,1,1,1,1,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,1,1,1,0,1,1,1,0,1,0,0,0,1,1,1,0), (0,0,1,1,1,1,1,1,0,1,1,1,0,1,0,1,1,1,1,1,0),
(1,0,1,1,1,x,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,1,1,1,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,1,1,1,0,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,1,1,1,1,0,0,0,1,1,0,1,0,0,0,1,1,1,0),
(1,0,1,1,1,1,0,0,0,1,1,1,0,1,0,1,1,1,1,1,0), (x,0,1,1,1,x,x,x,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,x,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,1,1,0,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,0,1,1,1,x,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,0,1,1,1,0,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,0,1,1,1,x,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,1,1,0,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),

Figure A.2. Test pattern generation with incorrect guidance using t5=0 (continued)

(1,0,1,1,1,0,1,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,0,1,1,1,0,1,0,1,0,1,1,0,1,0,0,0,1,1,1,0),
(1,0,1,1,1,0,1,0,1,0,1,1,0,1,0,1,1,1,1,1,0), (x,1,0,x,x,x,x,x,x,x,x,x,0,1,0,x,x,1,1,1,0),
(x,1,0,0,0,x,x,x,x,x,x,x,0,1,0,x,x,1,1,1,0), (x,1,0,0,0,x,x,x,x,x,x,1,1,0,1,0,x,x,1,1,1,0),
(x,1,0,0,0,x,x,x,0,0,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,0,0,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,0,0,x,x,1,0,0,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,0,0,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0),
(x,1,0,0,0,x,x,x,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,0,0,x,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,0,0,x,x,1,0,1,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,0,0,x,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0),
(x,1,0,0,0,x,x,x,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,0,0,x,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,0,0,0,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,0,0,1,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,0,0,x,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,0,0,0,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,0,0,1,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,0,0,x,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,0,0,0,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,0,0,0,1,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,0,0,0,1,0,1,0,1,1,0,1,0,0,0,1,1,1,0), (1,1,0,0,0,0,1,0,1,0,1,1,0,1,0,1,1,1,1,1,0),
(1,1,0,0,0,1,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,0,0,1,1,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,0,0,1,1,0,1,0,1,1,0,1,0,0,0,1,1,1,0), (1,1,0,0,0,1,1,0,1,0,1,1,0,1,0,1,1,1,1,1,0),
(x,1,0,1,1,x,x,x,x,x,x,0,1,0,x,x,1,1,1,0), (x,1,0,1,1,x,x,x,x,x,1,1,0,1,0,x,x,1,1,1,0),
(x,1,0,1,1,x,x,x,0,0,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,1,1,1,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,x,x,1,0,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,1,1,1,x,1,0,0,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,1,1,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,1,1,1,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,1,1,1,1,0,0,0,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,1,1,1,1,0,0,0,1,1,0,1,0,0,0,1,1,1,0), (1,1,0,1,1,1,1,0,0,0,1,1,0,1,0,1,1,1,1,1,0),
(x,1,0,1,1,x,x,x,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,x,x,0,0,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,1,1,1,x,0,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,1,0,0,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,1,1,1,0,0,0,1,1,1,0,1,0,0,0,1,1,1,0), (0,1,0,1,1,1,0,0,0,1,1,1,0,1,0,1,1,1,1,1,0),
(0,1,0,1,1,1,1,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,1,1,0,0,1,1,1,0,1,0,0,0,1,1,1,0),
(0,1,0,1,1,1,x,1,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,0,0,1,1,1,0), (0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,1,1,0),
(0,1,0,1,1,1,1,1,0,1,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,1,1,1,0,1,1,1,0,1,0,0,0,1,1,1,0),
(0,1,0,1,1,1,1,1,0,1,1,1,0,1,0,1,1,1,1,1,0), (1,1,0,1,1,x,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,1,1,1,x,0,0,1,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,1,1,1,0,0,0,1,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,1,1,1,0,0,0,1,1,1,0,1,0,0,0,1,1,1,0), (1,1,0,1,1,1,0,0,0,1,1,1,0,1,0,1,1,1,1,1,0),
(x,1,0,1,1,x,x,x,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,x,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,1,1,0,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (0,1,0,1,1,x,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0),
(0,1,0,1,1,0,x,1,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,1,1,x,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,1,1,0,x,0,1,0,1,1,0,1,0,x,x,1,1,1,0), (1,1,0,1,1,0,1,0,1,0,1,1,0,1,0,x,x,1,1,1,0),
(1,1,0,1,1,0,1,0,1,0,1,1,0,1,0,0,0,1,1,1,0), (1,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,1,1,1,1,0)

Figure A.2. Test pattern generation with incorrect guidance using t5=0 (continued)

,f2,f3,f4,f5,p1,p2,p3,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13) =
(x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,x,1,x,x,x,x,1,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0), (x,0,0,x,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,0,0,0,1,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,x,x,0,0,1,0,1,x,x,1,0,0,0),
(x,0,0,0,1,x,x,x,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,x,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,0,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0), (1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,1,0,1,0,0,0)

Figure A.3. Test pattern generation with correct guidance using t5=1

(f1,f2,f3,f4,f5,p1,p2,p3,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13) =
(x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,x,0,x,x,x,x,0,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,0,0,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,1,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (1,0,0,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0),
(x,0,0,1,1,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,1,1,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,1,1,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0), (x,0,0,1,1,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,0,0,1,1,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(1,0,0,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(x,0,0,1,1,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (x,1,1,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),

Figure A.4. Guided test generation using t9=0

(0,1,1,0,0,1,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (1,1,1,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0),
(x,1,1,1,1,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,1,1,1,1,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,1,1,1,1,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0), (x,1,1,1,1,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,1,1,1,1,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,1,0,0,0,0,0),
(1,1,1,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0),
(x,1,1,1,1,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,1,0,0,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,0,1,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,1,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(1,0,1,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0), (x,0,1,1,0,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,1,0,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,1,0,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,1,1,0,0,0,1,1,0,1,1,1,0,0,0,0),
(x,0,1,1,0,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),

Figure A.4. Guided test generation using t9=0 (continued)

(0,0,1,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,0,1,1,0,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0,0,0,0,0), (1,0,1,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (x,0,1,1,0,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(x,1,0,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,1,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(1,1,0,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0), (x,1,0,1,0,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,1,0,1,0,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,1,0,1,0,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,1,0,0,0,0,0),
(x,1,0,1,0,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,1,0,1,0,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0), (0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1,0,0,0,0,0), (1,1,0,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0), (x,1,0,1,0,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),

Figure A.4. Guided test generation using t9=0 (continued)

(1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,1,0,0,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,1,x,x,x,x,1,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,0,0,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,0,0,0,1,x,x,x,x,x,0,0,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,x,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,1,0,1,0,0,0)

Figure A.4. Guided test generation using t9=0 (continued)

(f1,f2,f3,f4,f5,p1,p2,p3,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13) =
(x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,x,0,x,x,x,x,0,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,x,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,0,0,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,0,0,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,0,0,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,0,0,1,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (1,0,0,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0),
(x,0,0,1,1,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,0,0,1,1,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,0,0,1,1,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0), (x,0,0,1,1,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,0,0,1,1,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,0,1,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,1,1,1,0,1,1,1,0,0,0,0,1,0,0,0,0),
(1,0,0,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(x,0,0,1,1,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,0,0,1,1,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,0,0,1,1,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,0,0,1,1,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (x,1,1,x,x,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,1,1,0,0,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(x,1,1,0,0,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,0,0,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,0,0,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),

Figure A.5. Guided test generation using t9=1

(0,1,1,0,0,1,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (1,1,1,0,0,1,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,0,0,1,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0),
(x,1,1,1,1,x,x,x,x,x,x,0,0,0,x,x,0,0,0,0), (x,1,1,1,1,x,x,x,x,x,1,1,0,0,0,x,x,0,0,0,0),
(x,1,1,1,1,x,x,x,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,x,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,x,1,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,x,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,x,0,0,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0), (x,1,1,1,1,x,x,x,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,1,1,1,1,x,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,x,1,0,1,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,0,1,0,1,1,1,0,0,0,0,1,0,0,0,0),
(0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,1,1,1,0,1,1,1,0,0,0,0,1,0,0,0,0),
(1,1,1,1,1,x,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,x,0,0,1,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0),
(x,1,1,1,1,x,x,x,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (0,1,1,1,1,x,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0),
(0,1,1,1,1,0,x,1,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,x,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,0,x,0,1,0,1,1,0,0,0,x,x,0,0,0,0), (1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,x,x,0,0,0,0),
(1,1,1,1,1,0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,0,1,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,0,1,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,0,1,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,0,1,1,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(1,0,1,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0), (x,0,1,1,0,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,0,1,1,0,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,0,1,1,0,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,0,1,0,0,0,0),
(x,0,1,1,0,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),

Figure A.5. Guided test generation using t9=1 (continued)

(0,0,1,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,0,1,1,0,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,0,1,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,0), (1,0,1,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (x,0,1,1,0,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,0,1,1,0,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,0,1,1,0,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(x,1,0,x,x,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,1,0,0,1,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (x,1,0,0,1,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,0,1,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,0,1,1,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(1,1,0,0,1,1,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0), (x,1,0,1,0,x,x,x,x,x,x,0,1,1,x,x,0,0,0,0),
(x,1,0,1,0,x,x,x,x,x,1,1,0,1,1,x,x,0,0,0,0), (x,1,0,1,0,x,x,x,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,x,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,x,1,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,x,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,x,0,0,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,1,0,0,0,1,1,0,1,1,0,1,0,0,0,0),
(x,1,0,1,0,x,x,x,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,1,0,1,0,x,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,x,1,0,1,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,0,1,0,0,0,0), (0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,0), (1,1,0,1,0,x,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,x,0,0,1,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,0), (x,1,0,1,0,x,x,x,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),
(0,1,0,1,0,x,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0), (0,1,0,1,0,0,x,1,1,0,1,1,0,1,1,x,x,0,0,0,0),
(1,1,0,1,0,x,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,0,x,0,1,0,1,1,0,1,1,x,x,0,0,0,0),

Figure A.5. Guided test generation using t9=1 (continued)

(1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,x,x,0,0,0,0), (1,1,0,1,0,0,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,x,1,x,x,x,x,1,0,0,0), (x,x,x,x,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,0,0,x,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,0,0,0,1,x,x,x,x,x,0,0,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,x,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0),
(1,0,0,0,1,1,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,1,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,1,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0), (x,0,0,0,1,x,x,x,x,x,0,1,1,0,1,x,x,1,0,0,0),
(x,0,0,0,1,x,x,x,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,0,1,x,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,0,1,0,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,0,1,0,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,0,1,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0), (0,0,0,0,1,0,1,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,0,1,0,1,0,1,1,0,1,1,0,1,0,1,1,0,0,0), (0,0,0,0,1,1,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,0,1,1,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,0,1,1,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,0,0,0,1,1,1,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,0,1,1,1,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,0,0,0,1,x,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,0,1,0,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,0,1,0,0,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,0,1,0,0,1,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,0,0,0,1,0,1,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,0,1,0,1,1,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,0,0,0,1,1,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,0,1,1,0,1,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,0,1,1,0,1,1,1,0,1,1,0,1,0,1,1,0,0,0), (0,0,0,0,1,1,1,1,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,0,1,1,1,1,1,1,0,1,1,0,1,0,1,1,0,0,0), (1,0,0,0,1,x,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,0,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0), (1,0,0,0,1,1,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,1,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,1,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(x,0,0,0,1,x,x,x,x,x,1,0,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,0,0,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,x,x,1,0,0,1,0,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,0,1,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,x,x,1,0,1,1,0,1,0,1,x,x,1,0,0,0), (x,0,0,0,1,x,x,x,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,x,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,0,1,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,0,1,1,1,0,1,0,1,0,1,0,1,1,0,0,0),
(1,0,0,0,1,1,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,0,1,1,1,1,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,0,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,0,0,0), (x,0,0,1,0,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,0,0,1,0,x,x,x,x,x,0,0,1,0,1,x,x,1,0,0,0), (x,0,0,1,0,x,x,x,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,x,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,0,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,0,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(x,0,0,1,0,x,x,x,x,x,0,1,1,0,1,x,x,1,0,0,0), (x,0,0,1,0,x,x,x,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,1,0,x,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,1,0,0,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,1,0,0,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,1,0,0,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,0,0,1,0,0,1,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,1,0,0,1,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,0,0,1,0,x,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,1,0,0,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,0,0,1,0,0,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,1,0,0,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,0,0,1,0,0,1,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,0,0,1,0,0,1,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(1,0,0,1,0,x,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,0,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,0,0,0,0,1,1,0,1,1,0,1,x,x,1,0,0), (1,0,0,1,0,0,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0),

Figure A.5. Guided test generation using t9=1 (continued)

(x,0,0,1,0,x,x,x,x,x,1,0,1,0,1,x,x,1,0,0,0), (x,0,0,1,0,x,x,x,0,0,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,x,x,1,0,0,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,1,x,1,0,0,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,1,1,1,1,0,0,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,1,1,1,1,0,0,1,0,1,0,1,0,1,1,1,0,0,0),
(x,0,0,1,0,x,x,x,0,1,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,x,x,1,0,1,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,1,x,1,0,1,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,1,0,1,0,1,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,0,0), (x,0,0,1,0,x,x,x,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,x,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,0,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,0,0,1,0,0,1,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,0,0,1,0,0,1,1,1,0,1,0,1,0,1,0,1,1,0,0,0),
(x,1,1,x,x,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0), (x,1,1,0,1,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,1,1,0,1,x,x,x,x,x,0,0,1,0,1,x,x,1,0,0,0), (x,1,1,0,1,x,x,x,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,1,1,0,1,x,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,1,1,0,1,0,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,1,1,0,1,0,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,1,1,0,1,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0),
(1,1,1,0,1,1,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,1,1,0,1,1,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,1,1,0,1,1,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0), (x,1,1,0,1,x,x,x,x,x,0,1,1,0,1,x,x,1,0,0,0),
(x,1,1,0,1,x,x,x,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,0,1,x,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,1,1,0,1,0,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,0,1,0,1,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,1,1,0,1,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0), (0,1,1,0,1,0,1,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,1,1,0,1,0,1,0,1,1,0,1,1,0,1,0,1,1,0,0,0), (0,1,1,0,1,1,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,1,1,0,1,1,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,0,1,1,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,1,1,0,1,1,1,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,0,1,1,1,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,1,1,0,1,x,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,0,1,0,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,1,1,0,1,0,0,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,0,1,0,0,1,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,1,1,0,1,0,1,1,1,1,0,1,1,0,1,0,1,1,0,0,0), (0,1,1,0,1,0,1,1,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,1,1,0,1,1,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,0,1,1,0,1,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,1,1,0,1,1,0,1,1,1,0,1,1,0,1,0,1,1,0,0,0), (0,1,1,0,1,1,1,1,1,1,0,1,1,0,1,0,1,1,0,0,0),
(x,1,1,0,1,x,x,x,x,x,1,0,1,0,1,x,x,1,0,0,0), (x,1,1,0,1,x,x,x,0,0,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,0,1,x,x,1,0,0,1,0,1,0,1,x,x,1,0,0,0), (x,1,1,0,1,x,x,x,0,1,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,0,1,x,x,1,0,1,1,0,1,0,1,x,x,1,0,0,0), (x,1,1,0,1,x,x,x,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,0,1,x,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,0,1,0,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,0,1,0,1,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,0,1,0,1,1,1,0,1,0,1,0,1,0,1,1,0,0,0),
(1,1,1,0,1,1,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,0,1,1,1,1,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,0,1,1,1,1,1,0,1,0,1,0,1,0,1,1,0,0,0), (x,1,1,1,0,x,x,x,x,x,x,1,0,1,x,x,1,0,0,0),
(x,1,1,1,0,x,x,x,x,x,0,0,1,0,1,x,x,1,0,0,0), (x,1,1,1,0,x,x,x,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,x,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,0,x,1,1,1,0,0,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,0,0,1,1,1,0,0,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0),
(x,1,1,1,0,x,x,x,x,x,0,1,1,0,1,x,x,1,0,0,0), (x,1,1,1,0,x,x,x,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,1,1,1,0,x,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,1,0,0,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,1,1,1,0,0,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,1,0,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0),

Figure A.5. Guided test generation using t9=1 (continued)

(0,1,1,1,0,0,1,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,1,0,0,1,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,1,1,1,0,x,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,1,0,0,x,1,1,1,0,1,1,0,1,x,x,1,0,0,0),
(0,1,1,1,0,0,0,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,1,0,0,0,1,1,1,0,1,1,0,1,0,1,1,0,0,0),
(0,1,1,1,0,0,1,1,1,1,0,1,1,0,1,x,x,1,0,0,0), (0,1,1,1,0,0,1,1,1,1,0,1,1,0,1,0,1,1,0,0,0),
(1,1,1,1,0,x,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,0,x,0,1,1,0,1,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,0,0,0,1,1,0,1,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,0,0,0,1,1,0,1,1,0,1,0,1,1,0,0,0),
(x,1,1,1,0,x,x,x,x,x,1,0,1,0,1,x,x,1,0,0,0), (x,1,1,1,0,x,x,x,0,0,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,x,x,1,0,0,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,1,x,1,0,0,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,1,1,1,0,0,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,1,1,1,0,0,1,0,1,0,1,0,1,1,0,0,0),
(x,1,1,1,0,x,x,x,0,1,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,x,x,1,0,1,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,1,x,1,0,1,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,1,0,1,0,1,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,0,0), (x,1,1,1,0,x,x,x,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,x,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,0,x,1,1,0,1,0,1,0,1,x,x,1,0,0,0),
(1,1,1,1,0,0,1,1,1,0,1,0,1,0,1,x,x,1,0,0,0), (1,1,1,1,0,0,1,1,1,0,1,0,1,0,1,0,1,1,0,0,0),
(x,x,x,x,x,x,x,x,x,x,x,x,1,1,0,x,x,1,0,0,0), (x,0,1,x,x,x,x,x,x,x,x,x,1,1,0,x,x,1,0,0,0),
(x,0,1,0,0,x,x,x,x,x,x,x,1,1,0,x,x,1,0,0,0), (x,0,1,0,0,x,x,x,x,x,0,0,1,1,0,x,x,1,0,0,0),
(x,0,1,0,0,x,x,x,1,1,0,0,1,1,0,x,x,1,0,0,0), (1,0,1,0,0,x,x,1,1,1,0,0,1,1,0,x,x,1,0,0,0),
(1,0,1,0,0,0,x,1,1,1,0,0,1,1,0,x,x,1,0,0,0), (1,0,1,0,0,0,0,0,1,1,1,0,0,1,1,0,x,x,1,0,0,0),
(1,0,1,0,0,0,0,0,1,1,1,0,0,1,1,0,0,1,1,0,0,0)

Figure A.5. Guided test generation using t9=1 (continued)

VITA

Benny P. Phillips

Candidate for the Degree of

Doctor of Philosophy

Thesis: NEW PROBABILISTIC MEASURES FOR ACCELERATING THE AUTOMATIC TEST PATTERN GENERATION ALGORITHM

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in My Tho, Viet Nam, November 11, 1958, the son of Khe Pham and Hieu Thi Mai. Married to Yu-Hsin Huang on February 4, 1988.

Education: Graduated from Chieu Dinh Nguyen High School, My Tho, Viet Nam, in May 1977; received the Associate of Technology degree in Computer-Controlled Machine from Oklahoma State Technical School at Okmulgee in May, 1983; received the Bachelor of Science degree in Electrical Engineering from Oklahoma State University in July, 1987; received the Master of Science degree from Oklahoma State University in December, 1989; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in July, 1993.

Professional Experience: Teaching Assistant, Department of Electrical Engineering, Oklahoma State Univeristy, August, 1987 to May, 1988; Electronics Engineer, Oklahoma City Air Logistics Center, Tinker Air Force Base, May, 1988 to present. Adjunct Professor, Oklahoma City Community College, January, 1993 to present.