National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

Syndrome Signature in Built-In Self-Testing of VLSI Circuits

by

Nita Goel

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

ISBN 0-612-11556-9

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor, Dr. Sunil R. Das, for his encouragement and guidance throughout the course of this research.

Appreciation is also expressed to the staff, professors, and graduate students of the Department of Electrical Engineering for their helps and efforts in providing the academic environment.

I also wish to thank my friends and all members of my family, especially my husband, Nishith, for their love, supports, and patience. Finally, I would like to dedicate this thesis to my children, Neha and Shilpa.

ABSTRACT

Testing has become very important in the context of modern VLSI and is a severe challenge for testing engineers. A number of basic analytic and heuristic methods exists for the solution of the fault detection and location problem in combinational circuits. Classical testing of combinational circuits requires a list of fault-free responses of the circuit to the test set. For most practical circuits implemented today, the large storage requirement makes the test procedure very expensive.

In this thesis, a syndrome signature is proposed that is particularly well suited for exhaustive testing of VLSI circuits and is based on the idea originally developed by Savir for syndrome testing. A syndrome is defined as the number of minterms realized by a function under exhaustive application of all possible input patterns. Given an *n*-input combinational circuit, a *syndrome signature* is defined by an *(n+1)*-element vector consisting of the primary syndrome of the circuit function *F* and *n* other *subsyndromes* corresponding to the *subfunctions* obtained by setting the *ith* variable in *F* equal to *0* or *1*. A *multiple-output syndrome signature* generation is also discussed, which preserves the desirable properties of the conventional single-output circuits response analyzers. The proposed technique was simulated on various combinational circuits and the results look very promising. The signature generators can be easily implemented using the current VLSI technology.

Table of Contents

Chapter 1

INTRODUCTION

1.1. Introduction

Digital systems, even when designed with high reliability can develop some faults. Faliures are caused by design errors, material defects, process defects, extremes in operational environment, deterioration due to length of operation, aging etc. Phenomena causing faliures can be physical or chemical in nature. However, when a system ultimately does develop a fault, it has to be detected and located so that its effect can be removed. *Fault detection* means the discovery of something wrong in a digital system or circuit. *Fault location* means the identification of the faults with components, functional modules, subsystems, depending on the requirements. *Fault diagnosis* includes both fault detection and fault location.

The rapid growth of integrated circuit technology has allowed LSI/VLSI devices to provide more powerful functions and to become more popular in commercial, educational, industrial, and military applications. These complex devices must be tested for correct functional operation to obtain certain assured reliability before being adopted in an individual application. Due to the increasing design complexity, however, testing problems have become more difficult than ever. The high complexity of VLSI systems renders gate-level testing in particular very difficult and expensive. The need for both manufacturers and users to find reliable, low cost, comprehensive testing techniques has already become a major problem in modern LSI/VLSI technology.

Fault detection in a logic circuit is carried out by applying a sequence of test inputs and observing the resulting outputs. Any fault in a nonredundant n-input combinational circuit can be completely tested by applying all 2^n input combinations to it; however, 2^n increases very rapidly as n increases. To analyze the faulty behavior and develop techniques to detect and locate faliures, abstract fault models are used. The most commonly modeled faults are line *stuck type* faults. In this model, each line of the circuit can have two possible faults: stuck-at-0 or stuck-at-1. Faults can occur singly or in multiples. A large portion of literature in this field deals with singly occurring faults. While this assumption simplifies the analysis, the fact is multiple faults do occur. An n-line circuit will have $2n$ single stuck faults. The assumption that only single *stuck-at* faults are present at any time has been used throughout this thesis.

In a combinational circuit, a specific *stuck-at* fault can be tested by a single input vector. Test generation for sequential circuits is much more complex than combinational circuits. In general, a sequence of vectors is required to test a fault in sequential circuits. Test generation in combinational circuits can be automatic, though the biggest problem is that the cost grows with the circuit size as stated in Chapter 2. A prerequisite for automatic test generation is an algorithm that can be programmed. Several methods of test generation have been discussed in Chapter 3.

Though there is no universally accepted criterion of fault coverage by verification tests, manufacturing tests are often required to cover 90-100% of all single stuck-at faults. The *reject ratio* is the real measure of the effectiveness of manufacturing tests and is defined as the ratio of faulty chips among the chips passed by the tests. Thus, the *fault coverage* should be high to keep the reject

ratio as low as possible. The coverage of a set of test vectors is measured through *fault simulation*. Several fault simulation algorithms have been developed for efficient programmability. *Parallel, deductive* and *concurrent* are well known algorithms for fault simulation. Fault simulation has been discussed in some details in Chapter 3.

While the increasing package density has caused dramatic reduction in per circuit cost, the percentage of those costs consumed by testing has stubbornly increased. This effect was forced primarily by the loss of ability to control and observe internal circuit nodes as the gate-to-pin ratio worsened. The need for designing circuits with testability in mind become more acute. *Design for testability*(DFT) commonly refers to those design styles that result in designs for which tests can be generated by known methods. The two most commonly used design for testability techniques are *scan* and *built-in self-test(BIST)*. Scan and BIST can nicely complement each other. While scan solves test problems arising from the sequential nature, the built-in self-test(BIST) avoids the problem of test generation for complex combinational circuits. BIST is widely used as an aternative testing method to conventional testing of digital circuits, where a circuit has the added capabilities to generate its own test vectors and check its output to produce a good or bad result. Built-in self-testing is steadily gaining popularity and has been very sucessfully used in industry. A more detailed discussion on BIST in the context of VLSI circuit testing.can be found in Chapter 2.

As it is obviously undesirable to build into the circuit a bit by bit comparison of its test responses to the expected responses because of the large storage requirement, it is usual to perform some form of data compression on responses(both the test response as well as reference response) before the comparison is being made. The compressed

response is referred to as the *signature* of the circuit under test. Any data compression method tends to lose information(unless the data being compressed is highly redundant) by producing the same compressed signature as the fault-free circuit. This loss is usually referred to as the *masking* and is measured by the probability that a faulty circuit will produce the same signature as the fault-free circuit. The relative masking probabilities can be used as one measure of efficiency of various data compression techniques. Chapter 4 describes many data compression techniques suggested in the literature. Of these, signature analysis technique is widely accepted in the industry.

A new data compression technique referred to as *syndrome signature* is developed by in this thesis. Chapters 5, 6, 7, and 8 are devoted to its development, and implementation, and conclusions drawn based on the implementation results.

1.2. Historic Development

When digital systems consisted of discrete components, testing used to be done basically in three different stages. First, each individual component used to be tested to a specification. Second, these components were assembled into digital elements and these elements were tested for their intended function. Third and final testing called *functional testing* was done at application level where all the interactions of timing, loading, noise etc. came into consideration.

The advancement in the technological development of the process of component miniaturization, in turn, fueled the trend to build smaller machines with higher component densities. In about 1970 the breakthroughs offered by LSI

permitted the placement of several thousands of logic gates on a pluggable unit.

With low circuit density, testing of an assembly was basically a task of probing circuit points, measuring voltages, and observing waveforms. The introduction of LSI increased the difficulty of the testing problem by some order of magnitude. There is no practical way of probing individual logic gates or memory circuits on an LSI chip, and chips of early design with about 300 logic gates had only about 45 access points to the logic circuitry. As the small geometry of integrated circuits permits only a few connections to probe pads or package terminals, the problem of that era was to test hundreds of circuits through tens of terminals. With further advancement in digital technology into 1980's, a ratio of 1000 circuits behind each pin became the nature of the test problem. In addition, the increasing logic gate count required more efficient means of testing to be devised. Today the number of circuits that are designed and fabricated on one silicon chip has grown to hundreds of thousands and the benifits of application of increased circuit density will force the levels of integration to continue to grow.

As early as 1959 the groundwork for automatic test pattern generation (TPG) was laid. There followed, over the next several years, some fundamental papers describing methods and algorithms for test pattern generation of combinational and sequential logic networks. A renewed dependence on automated methods of providing test patterns caused TPG to develop rapidly into a separate science. While technology moved at a very rapid pace, testing and design automation (DA) support underwent rapid growth.

Concurrent with the development of TPG methods went the development of fault simulation, which is needed as a tool for determining how well the tests perform. Fault simulation

did not require fundamentally new solutions, but good and economical applications of understood principles. The basic approaches are *parallel*, *deductive*, and *concurrent* fault simulation.

The test equipment has developed from simple testers to today's complex machines, which permit programming of the test data. The speed of test application has also increased dramatically. Current status of testing is designing logic for testability and built-in self-testing which can significantly improve testability of VLSI chips and save testing time.

Techniques for design for testability and built-in self-test (BIST) consider the testing problem during the design stage of digital device. As the digital circuit technology is moving to high densities of integration, built-in self-testing has become important in VLSI circuit design. The main idea behind this technique is to have the chip test itself. This technique generates test patterns and evaluates output responses inside the chip. The test patterns used in this technique can either be exhaustive or random.

1.3. Research Objective

In built-in self-testing, the test output responses are compressed by the output response analyzer into signatures. The various available output compaction techniques include parity bit checking[1,3,5,7,17,26,30], transition count [1,11,14,30,31], one's count [1,30,31], Walsh coefficients [15,36] and linear feedback shift register [12]. Based on these approaches, the compressed response data can be used to evaluate the correctness of the circuit under test (CUT).

Savir[32,33] suggested a novel approach of designing combinational circuits so that the storage requirement for the implementation of the test procedure is restricted to only one specific number, called the *syndrome* of the circuit, which is basically related to the number of minterms realized by the corresponding switching function. Since it is not necessarily true that a fault-free and a faulty circuit should have different syndromes, special design techniques are required in order to make circuits syndrome testable, if it is not so originally. This requires an increase in the number of pins on the chip. It is well known that one of the most severe restrictions on IC manufacturers is the number of pins per chip.

To overcome the disadvantage of syndrome-testable design we have proposed a new output compaction technique called *syndrome signature* particularly well suited for exhaustive testing of VLSI circuits. This technique is based on the idea originally developed by Savir. A possible extension of the syndrome signature concept from single-output to multiple-output circuits and some thoughts on its VLSI implementation has also been considered.

1.4. An Overview of the Thesis

The importance of testing in the context of modern VLSI is discussed in Chapter 2. Built-in self-test has been discussed as an efficient technique for testing of VLSI circuits. Circuit partitioning, while using exhaustive testing, is suggested as one way to solve the problems of testing for complex combinational circuits.

In Chapter 3 various deterministic and probabilistic methods for test generation in combinational circuits are

given. Some of them are discussed with examples while some others are discussed only in brief.

The various available output compaction techniques are discussed in Chapter 4. Simple examples have been used to show how the techniques exactly work. For most practical circuits implemented today, these techniques solve the storage problem associated with the test procedure.

A computationally efficient output compaction technique is given in Chapter 5. This technique uses the primary syndrome of the circuit under test and n other subsyndromes corresponding to the subfunctions to generate a signature which is more likely to detect a fault in the circuit. Extension of the concept to the testing of multiple-output circuits is also provided. Also, it has been shown that the method does not work in some cases of combinational logic circuits.

In Chapters 6 and 7, the implementation results on some simple combinational logic circuits having number of inputs not more than 6, and on some benchmark circuits[6] that are partitioned in such a way that each subcircuit does not have more than 18 inputs, are presented. Most of the circuits are tested for input line stuck faults, but results have been presented for internal line stuck faults also for some of the above circuits.

Chapter 8 concludes with the summary of the work and suggestions for future research.

Chapter 2

IMPORTANCE OF TESTING IN THE CONTEXT OF MODERN VLSI

Testing is a major portion of the effort in the design, production, and use of digital circuits. As the digital system technology moved from discrete components to increasigly high levels of integration, the demands on testing methods and their effectiveness have caused the test technology to move to a high degree of sophitication.

Very large scale integration(VLSI) is the fabrication of thousands of circuits at once by a common set of manufacturing steps. Its advantages are reduced system cost, better performance, and greater reliability. These advantages would be lost if VLSI devices are not tested economically. The object of integrated circuit testing is to determine if the circuit is functioning. In their case no repair is possible; so a failed test leads to a decision to discard a unit. The small geometry of integrated circuits permits only a few connections to probe pads or package terminals. The exact relationship varies, but in general testing thousands of circuits behind each pin is becoming the nature of test problem. As many applications for low cost integrated circuits find their way in day to day life, the demand that they be properly tested for correct functional operation becomes intense. The inconvenience or danger that can result from a faulty integrated circuit in an application is not something which can be ignored by manufacturers of such circuits. This pressure for quality demands for better test techniques which can be applied without appreciable increase

in the cost to integrated circuits designed for these low cost applications.

The role of testing in the VLSI device realization process is illustrated in Figure 2.1.[38]

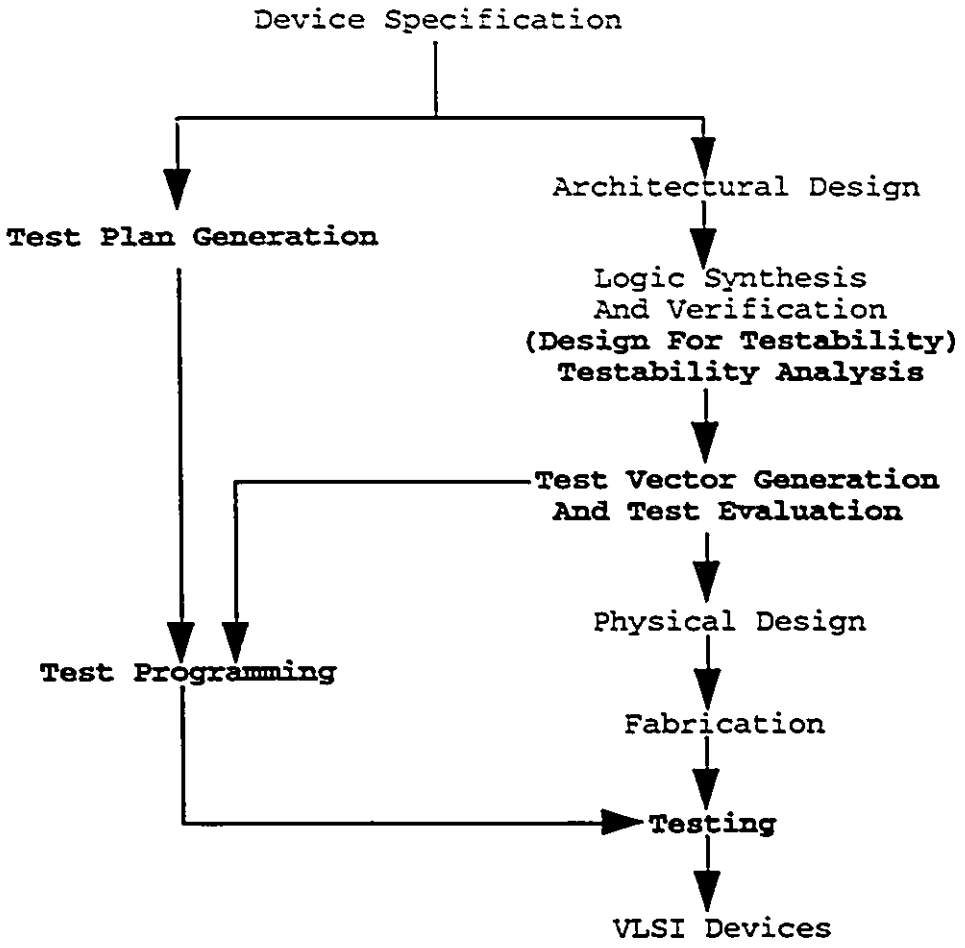


Fig. 2.1. Test functions in VLSI realization.

Testing appears into the life cycle of a VLSI device in several places. First, in the factory, *manufacturing tests* are done on device wafers and packages. Second, *acceptance test* is conducted by users of VLSI devices. Third, the devices undergo *systems test* performed on the systems where

the VLSI chips are used. Finally the devices undergo *maintenance tests* in the field.

Built-in self-test(BIST) as the solution to testing VLSI designs

The benifits of the application of increased circuit density will force the levels of integration to continue to grow. As the number of circuits that can be integrated into one silicon chip grows from hundreds to thousands, and on to hundreds of thousands, conventional externally applied tests will become less and less satisfactory, and the testing problems will become more intense. It is in easing the test portion of the task where built-in self-test has its benifits and is a clear alternative to the externally applied tests. With the increasing levels of integration some small portion of the circuits on silicon chip could be devoted to testing. This concept is called *built-in self-test*. Thus, it appears that the only economical method to reduce functional testing cost and to improve the quality of test is to include circuitry on chip to facilitate testing. When built-in self-test is incorporated in a digital network to meet a testability specification, at that point, test is a part of design. The design styles that result in testable design are collectively grouped under the name *design for testability(DFT)*. The problems of delivering test patterns to an unpackaged VLSI chip through impedence discontinuities at high speed are considerably eased by built-in self-test. Very little power and a clocking signal need be supplied to a VLSI structure with suitable built-in self-test in order for an accept or reject decision to be made. Systems designed with components having BIST have much better reliability, dignosability, and repairability.

Built-in self-testing can be either *structural* or *functional* in nature [39]. *Pseudorandom testing* is a type of

built-in self-test that performs a structural test of the network involved. Pseudorandom patterns can be generated by simple built-in generators. Other built-in self-test approaches perform either functional or structural testing work with either stored deterministic patterns or execute functional programs that are designed to test specific parts of the network. *Exhaustive testing* is another type of structural built-in self-test, which requires logical partitioning to make the tests practical. The method chosen for test generation may vary based on considerations of desired quality and cost of tests.

Testing Complex Combinational Circuits

For combinational circuits, the biggest problem in automation of test generation, evaluation, and application is that their costs grow faster than the linear rate with the increase in circuit size. The computation time of available methods for test generation and evaluation grows at least as the square of the number of gates in the circuit. The test volume, reflecting the cost of test application, also grow proportional to the square of the number of gates [38, 39]. At high circuit densities, a circuit overhead of 1-3% for built-in self-test is enough to implement some quite sophisticated built-in self-test schemes. The requirements on automatic test equipment is drastically eased by built-in self-test procedures. The pressure on test application time is much less if lower cost equipment can be used without compromising the quality.

In some special situations, high quality can be obtained at a very low cost. For example, for a circuit with small number of primary inputs, application of all possible input patterns will ensure complete fault coverage. Such patterns can be easily generated by software or hardware and may be applied quickly at electronic speed. Exhaustive testing can

be extended to more complex circuits. A concept of *divide and conquer* is a common element in most such cases. A complex circuit is designed as an interconnection of circuits which, in turn, can be furthur partitioned into smaller circuits or subcircuits, and so on. Logic partitioning by itself is not sufficient to reduce the complexity of the test problems. The divide and conquer is useful only if the subproblems resulting from the division are independently solvable and there is a simple way of relationg their solutions.

Unfortunately, the problem of logic partitioning is intractable, and hence cannot be fully automated. In other words, the total cost of exhaustive test generation does not reduce but simply moves toward partitioning the logic. Subcircuits can be tested exhaustively in built-in self-test environment where hardware test generators in the form of multiple-input linear feedback shift registers are included on the chip. Since in this environment it would be impractical to store the response(which may run into millions of bits) of the fault-free circuit to a very large number of test vectors, some form of data compression is usually performed on the responses before the comparison is done with the actual response of the circuit under test. The compressed response is referred to as the *signature* of the circuit under test, and comparison is made to the precomputed and stored reference signature. A basic test module for exhaustive testing is shown in Figure 2.2 [38].

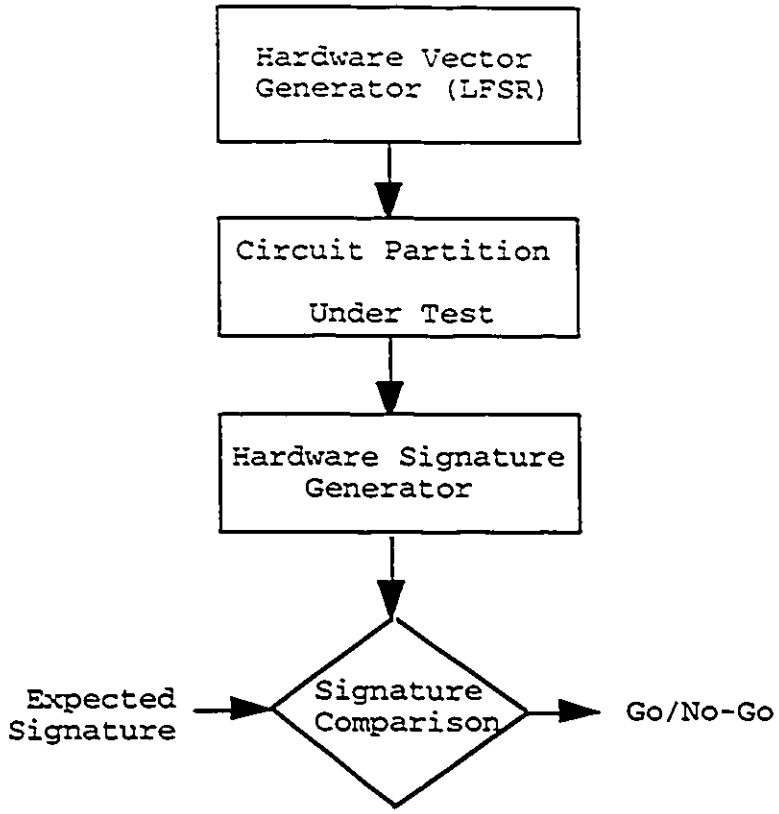


Fig. 2.2. Basic test module for exhaustive test.

The *syndrome signature* an output compaction technique developed in this thesis and discussed in Chapter 5 combines built-in self-test technique in which test patterns are generated exhaustively.

Chapter 3

TESTING COMBINATIONAL CIRCUITS - AN OVERVIEW OF VARIOUS METHODS

3.1. Introduction

Despite designing digital systems with highly reliable components, faults do occur. The object of testing digital networks is to detect faults in order to ensure error-free operations. The goal of test generation is to obtain test vectors of high quality at an affordable cost.

A model of faults which does take into account most of the major defects that may cause circuit failures is the *stuck-at model*. While more complex models have been proposed, the stuck-at model is the accepted model used throughout the industry. The stuck-at model assumes that a logic gate input or output is fixed at either logic 0 or logic 1. The fault analysis becomes simplified if only single stuck faults are considered. Historically, a test with a high level of single stuck-at fault coverage has resulted in components with low defect levels being passed on to the next level of assembly. The total number of possible single stuck-at faults in a logic network is a function of the number of connecting lines; for p lines there could be $2p$ single stuck-at faults.

The term *fault coverage*, commonly considered as a measure of test quality, is the fraction of modeled faults detected by the test vectors. The test costs may be considered to be associated with the processes of test generation and test application respectively. While the cost

associated with test generation is the one time cost measured by the computational effort required to generate test vectors, the cost associated with the test application is a recurrent cost in the production and field environments, and refers to the time it takes to apply the test vectors to the circuit under test.

The problem of VLSI testing consists of the following two subproblems:

1) Test pattern generation(TPG).
2) Fault simulation.

Both these problems are discussed in the following sections.

3.2. Test Pattern Generation(TPG)

The main objective of test pattern generation for logic networks is to find tests for the faults modeled in the network. This should be done economically. Hence, automatic test pattern generation technique has gained high importance. Because of the increasing complexity of modern digital circuits, the ability to generate tests automatically is becoming more and more difficult. The number of multiple stuck faults increases exponentially with the number of gates for large LSI chips. Hence, test generation for multiple faults leads to computations which are impractical.

Test generation for both combinational and sequential logic circuits can be classified as either *deterministic* or *probabilistic*.

3.2.1. Deterministic approach

A number of basic analytic and heuristic methods, viz. the fault table method[4], Armstrong's path sensitization method[4,21], and the equivalent normal form(ENF) method[4], methods utilizing Karnaugh maps and tabular techniques, the ENF-Karnaugh map method, analytical techniques based on the concept of Boolean difference, conventional, modified, or partial[4,10,20,21], the algorithmic techniques such as the D-algorithm of Roth *et al.*[4,21], the PODEM[4,21] or the SPOOF method[42] all fall under the category of deterministic test generation methods of combinational logic circuits. Of these, the fault table method is the most classic approach to the problem. It is completely general, and always yields a minimum set of diagnostic tests. However, it suffers from the disadvantage that it requires a very large fault table to be constructed. To overcome this problem, the concept of path sensitization was introduced. The path sensitization method requires use of no fault table, but it suffers from another drawback that the method is suitable only for the class of tree type combinational circuits and some special non-tree type circuits.

A heuristic, systematic procedure derived from the concept of path sensitization known as the equivalent normal form (ENF) method, is then introduced. Although this method has eliminated the two imperfections that the fault table and path sensitization methods have, it instead introduces an unattractive new feature, the requirement of the cumbersome computation of a *score function* for every *literal* in the ENF and the complemented ENF of the circuit. Then comes an ENF-Karnaugh map method which eliminates the above three shortcomings. This method is a combination of the ENF method, and the Karnaugh map and tabular method. Both the ENF method and the ENF- Karnaugh map method do not guarantee minimal

experiments, and there is no guarantee that a set of sensitized paths can be found for every circuit.

A simple and direct method for constructing a minimal complete fault detection test set for any two-layer irredundant combinational logic circuit utilizes certain basic properties of the prime implicants of a minimized function when displayed on the Karnaugh map. By using these properties not only stuck-at-0 and stuck-at-1 tests for each gate can be read out from the map, but also the information about the effect of the variable change on the output is also provided. However, it is necessary to know what the output should be for a specific test so that the result of the applied test can be monitored at the output. This method may be considered to have graphical and tabular versions. The graphical version which uses the Karnaugh map is easy to apply to small circuits having input variables not more than six but preferably upto four. The tabular method allows the circuit to have any finite number of input variables and is therefore particularly attractive from computer implementation point of view.

The Boolean difference method is a conceptually simple and a straightforward way of deriving test patterns for combinational logic circuits. The Boolean difference is an EXOR sum of the two Boolean functions: one representing the fault-free(normal) circuit, while the other representing the faulty circuit. The Boolean difference methods have proved to be convenient and mathematically elegant for deriving tests for detection of any single and multiple faults in any part of the circuit, without using any fault tables or maps. Like the Boolean difference methods, the SPOOF(structure and parity observing output function) method is also a general and convenient method. It provides all the necessary information for the complete analysis of the effects of possible faults on the functional characteristics of a given

combinational logic circuit. The derivation of the SPOOF[42] of a given circuit is very similar to that of ENF, and is an easy-to-derive method for obtaining tests for the detection of single and multiple faults in combinational logic circuits.

The algorithmic techniques such as the D-algorithm is considered to be the most widely used test generation technique so far for digital systems. Two computer programs originally due to Roth, DALG-II(D-algorithm Version II) and TEST-DETECT, besides others are available. The former computes a test to detect a failure, and the latter ascertains all failures detected by a given test set. Both are based upon the utilization of a *calculus of D-cubes* that provides the means for effectively performing the necessary computations for large combinational circuits.

Some of these test generation methods are discussed here in brief. While discussing these methods the main emphasis is on combinational logic circuits and on the basic assumption that the circuit under test is a nonredundant and only single stuck-at fault is present at any time.

1) Fault Table Method

A test set or experiment derived by this method[4,21] is obtained by comparing the truth tables of the normal and faulty circuits. Two types of testing experiment are possible.

One is *fixed scheduled*, which means that the choice of test schedules is completely independent of the outcome of the individual tests in the sequence.

The other is *adaptive* or *sequential scheduled*, which means that the choice of the test schedules depends upon the outcome of the individual tests in the sequence.

It is proved that the length of a minimal adaptive fault detection test set is the same as that of a minimal fixed scheduled fault detection test set. Hence there is no advantage in using an adaptive fault detection experiment. But for fault location and fault location-to-within-modules, the possible reduction in test schedule lengths could be quite substantial when adaptive experiments are used.

The fault table method requires construction of the fault table, which in general appears impracticable. For example, if a circuit has 20 inputs and if there are 30 lines within the circuit, there will be over 3×10^7 entries in the associated fault table. To overcome this problem, the idea of path sensitization was developed which is discussed next.

2) Path Sensitization Method

The basic principle behind the path sensitization method[4,21] is that a fault detection test set may be readily found by simply examining the paths of transmission from the origin of failure to one of the primary outputs of the circuit. A path is said to be *sensitized* if the inputs to the gates along this path are assigned logical values so as to propagate any change in the faulty line along the chosen path to the output terminal the path leads to. In a logic circuit, a *primary input* is a line that is not fed by any other line in the circuit and a *primary output* is a line whose signal output is accessible to the exterior of the circuit. A *transmission path* or simply a *path* in a combinational circuit, is a connected, directed graph

containing no loops from a primary input or internal line to one of its primary outputs.

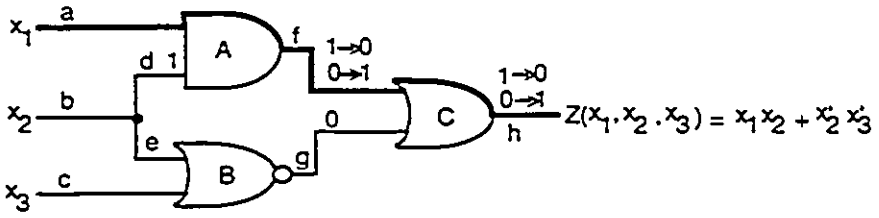


Fig. 3.1. Circuit to illustrate the concept of sensitized path.

Example: The method can be illustrated using the circuit in Figure 3.1 as shown above. Each line of the circuit is labeled by a letter, $^{a,b,c,\cdots}$. Suppose that we want to detect the fault a_0 (line *a* s-a-0). The connected, directed graph *afh* is a path containing the faulty line *a*. Now if we assign a value 0 to the input *g* of the *C* gate (an OR gate) and a value 1 to the input *d* of the *A* gate (and which happens to be an AND gate), we then see that:

Table 3.1

Value change on line *a*	Corresponding change in value on line *f*	Corresponding change in value on line *h*
0→1	0→1	0→1
1→0	1→0	1→0

The actual change in values along path *afh* upon occurrence of the fault in line *a* are indicated in Figure 3.1. Path *afh* is thus said to be *sensitized*. From value of line *g=0* it is required that *c* or *e* or both should be equal to 1. If we choose to make *e=1* and *c=0* or *1* (don't care),

then the values of d and e are 1 and thus are consistent; hence a test exists for detecting the fault a_0 using this method.

Furthermore, it is apparent from Table 3.2 that a test which detects fault a_0 also detects faults f_0 and h_0 and that a test which detects fault a_1 also detects faults f_1 and h_1. Thus the above two tests together would detect all the s-a-0 and s-a-1 faults on this path.

This method, however, does not necessarily yield a test for detecting a fault. But it always yields a test for a class of combinational circuits, known as tree like circuits. A *tree like* circuit is a circuit in which 1) each input is an independent input line to the circuit, and 2) the fan-out of every gate is 1. The fan-out of a logic gate is the number of gates that can be reliably driven by the gate.

Table 3.2

Path	Input variable values required to sensitize the path	Value of the input variable of the path	Response
afh	$x_2 = 1$, $x_3 = 0$ or 1 (don't care)	$x_1 = 1$	1, if the circuit is faultless 0, if the circuit has fault a_0
		$x_1 = 0$	0, if the circuit is faultless 1, if the circuit has fault a_1

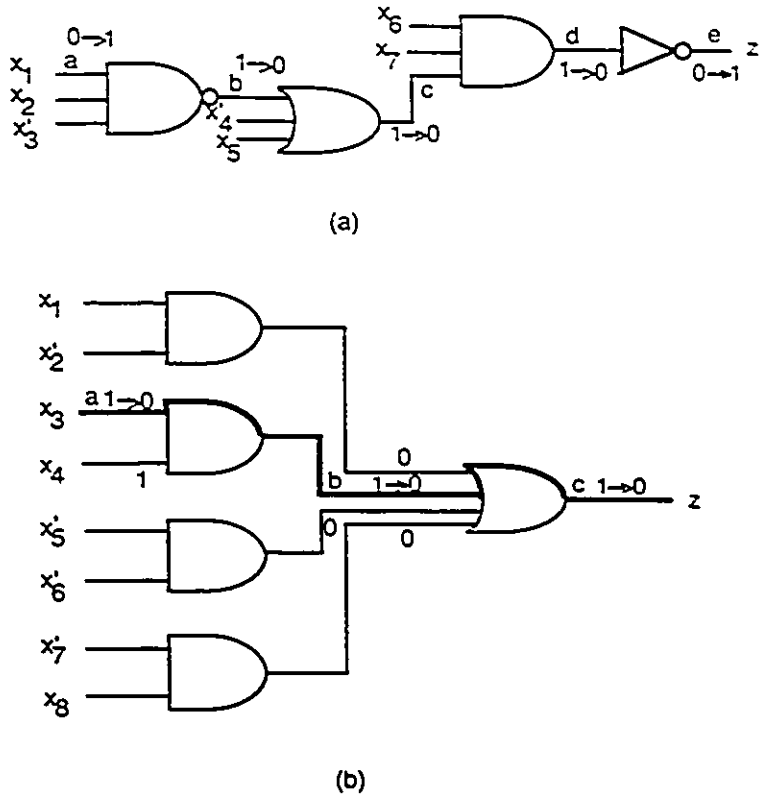


Fig. 3.2. Two tree like circuits.

The circuits of Figure 3.2(a) and (b) are examples of *tree like* circuits. The reason that the path sensitizing method always yields a test for detecting any single fault in a tree like circuit is because that a consistent input combination always exists for any sensitized path in the circuit. For example, the faults a_1, b_0, c_0, d_0, and e_1 of the circuit of Figure 3.2(a) can be detected by the test $x_1 = 0$, $x_2 = 1$, $x_3' = 1$, $x_4' = 0$, $x_5 = 0$, $x_6 = 1$, $x_7 = 1$, and the faults a_0, b_0, and c_0 of the circuit of Figure 3.2(b) can be detected by the test $x_1 = x_2' = x_5' = x_6' = x_7' = x_8 = 0$ and $x_3 = x_4 = 1$.

For general combinational circuits, a contradiction of line value may occur at the gates, which are points of

reconvergence for two or more fan-out paths from some preceding gate.

There can be cases where none of the individual paths of a circuit is sensitizable, but if we sensitize a group of them simultaneously, they become sensitizable. For example, none of the individual paths of the circuit of Figure 3.3 from x_1 to z for detecting the fault line a s-a-0 is sensitizable, but if we sensitize all the three paths from x_1 to z simultaneously, they become sensitizable.

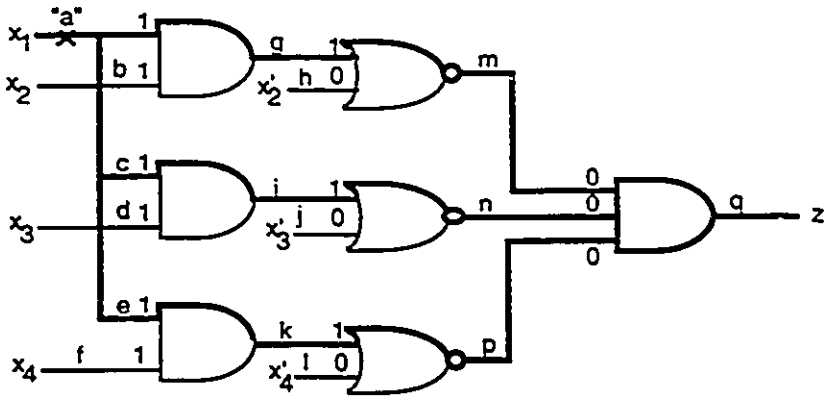


Fig. 3.3. Example of sensitizing several paths simultaneously.

This shows that in searching out the sensitizable path for a particular fault, all possible combinations of single paths sensitized simultaneously must be included.

3) Equivalent Normal Form Method (ENF Method)

The equivalent normal form (ENF)[41] of a circuit is obtained by expressing the output of each gate as a sum-of-

products expression of its inputs and preserving the identity of each gate by a suitable subscript. Each subscripted input variable in an ENF is called a *literal* and an appearance of a literal in a term is also called a *literal*.

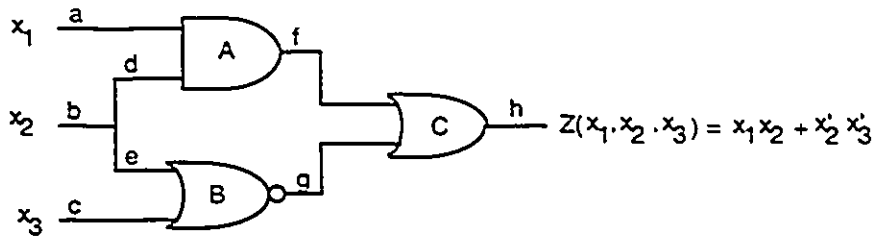


Fig. 3.4. Circuit example.

Example: The ENF of the circuit of Figure 3.4 is

$$h = (f+g)_h = (ad)_{fh} + (e'c')_{gh}$$
$$= (x_1)_{afh}\,(x_2)_{bdfh} + (x_2')_{begh}\,(x_3')_{cgh}$$

The ENF of a combinational circuit possesses the following characteristics:

1) The ENF is a sum-of-products expression; hence the name *normal form*.

2) Each literal consists of an input variable, possibly primed, subscripted by a sequence of gate numbers. The variable and its subscript sequence identifies a path from the corresponding circuit input to the output.

3) There must be at least one literal appearing in the ENF for every possible path from each input to the outputs, before any term of the ENF being discarded for the reason described in the next characteristic.

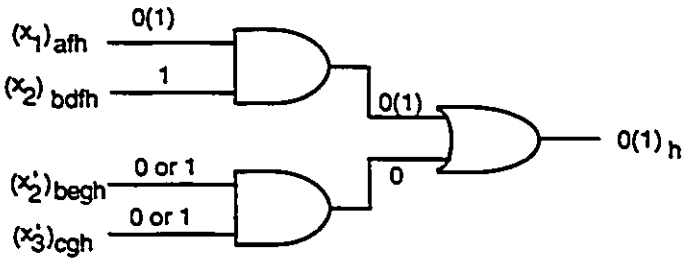
4) Any term in the ENF may be discarded if it contains a variable and its complement.

5) A literal may make several appearances in an ENF.

6) The ENF may contain redundant terms and literals, even though the original circuit is irredundant.

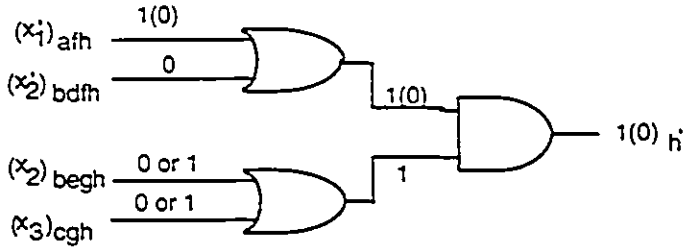
7) If the path associated with a literal in the ENF contains an odd number of inversions (an inversion is produced by each NAND, NOR, and NOT element in the path), the priming on the literal will be opposite to that on the corresponding input in the original circuit. If the number of inversions on the path is even, the priming on the literal will be the same as that on the corresponding input in the original circuit.

8) Every ENF corresponds to a hypothetical two level AND-OR equivalent circuit. For example, the ENF of the circuit of Figure 3.4 corresponds to the hypothetical two level equivalent circuits of Figure 3.5.



(a)

Hypothetical eqivalent two level AND-OR circuit of the circuit in Fig. 3.4 derived from the ENF.



(b)

Hypothetical eqivalent two level OR-AND circuit of the circuit in Fig. 3.4 derived from the complemented ENF.

Fig. 3.5. Hypothetical equivalent circuits of Example of Fig.3.4.

According to the rules of path sensitizing and the eighth characteristic described above, to test a particular literal for s-a-1 (Figure 3.5(a)), that literal must be assigned value 0, while the remaining literals in the term are assigned value 1. Also, at least one literal in each remaining term must be 0, in order that all remaining inputs to the OR gate be 0. By similar reasoning, a test that assigns value 1 to all literals of a particular term and assigns at least one 0 to literal in each remaining term tests all literals in that particular term for s-a-0. These rules are referred as *literal sensitizing rules*.

The four literals $(x_1)_{afh}$, $(x_2)_{bdfh}$, $(x_2')_{begh}$, and $(x_3')_{cgh}$ identifying the four paths in the circuit of Figure 3.4 can be tests for a s-a-0 and s-a-1, which are shown in Table 3.3. Since the paths associated with these four literals contain all the connections of the circuit and the set of tests of the rightmost column of Table 3.3 tests each of the four literals for s-a-0 and s-a-1, this set of tests detects every s-a-0 and s-a-1 fault in the circuit. From this table it is seen that tests 2 and 5 are essential, and these

two tests plus two other tests, test 0 or 4 and test 6 or 7, constitute a minimal complete test set for detecting every single s-a-0 and s-a-1 fault. The same set of tests can be obtained by using the fault table method, but the equivalent normal form method requires much less effort compared to the fault table method.

Table 3.3

Tests for detecting the four literals of the ENF for s-a-0 and s-a-1

ENF	$h = (x_1)_{afh}(x_2)_{bdfh} + (x_2')_{begh}(x_3')_{cgh}$				Tests
Test $(x_1)_{afh}$ for s-a-0	1	1	0	d	6 or 7
Test $(x_1)_{afh}$ for s-a-1	0	1	0	d	2 or 3
Test $(x_2)_{bdfh}$ for s-a-0	1	1	0	d	6 or 7
Test $(x_2)_{bdfh}$ for s-a-1	1	0	1	0	5
Test $(x_2')_{begh}$ for s-a-0	d	0	1	1	0 or 4
Test $(x_2')_{begh}$ for s-a-1	0	1	0	1	2
Test $(x_3')_{cgh}$ for s-a-0	d	0	1	1	0 or 4
Test $(x_3')_{cgh}$ for s-a-1	d	0	1	0	1 or 5

4) Boolean Difference Method

The basic principle of the Boolean difference[4,10,20,21] is to derive two Boolean expressions- one representing normal fault-free behavior of the circuit, and the other representing the logical behavior under an assumed single s-a-1 or s-a-0 fault condition. These two expressions are then EXORed; if the result is 1, a fault is indicated.

If one of the inputs(say x_i) to the logic function $F(X) = F(x_1, \cdots, x_n)$, is faulty, then the output would be $F(X) = F(x_1, \cdots, \overline{x_i}, \cdots, x_n)$. The Boolean difference of $F(X)$ with respect to x_i can then be expressed as

$$\frac{dF(x_1, \cdots, x_i, \cdots, x_n)}{dx_i} = \frac{dF(X)}{dx_i}$$

$$= F(x_1, \cdots, x_i, \cdots, x_n) \oplus F(x_1, \cdots, \overline{x_i}, \cdots, x_n).$$

The function $dF(X)/dx_i$ is called the *Boolean difference* of $F(X)$ with respect to x_i.

It can be seen that when $F(x_1, \cdots, x_i, \cdots, x_n) \neq F(x_1, \cdots, \overline{x_i}, \cdots, x_n)$, $dF(X)/dx_i = 1$ and that when $F(x_1, \cdots, x_i, \cdots, x_n) = F(x_1, \cdots, \overline{x_i}, \cdots, x_n)$, $dF(X)/dx_i = 0$. To detect a fault on x_i, it is therefore necessary to find input patterns(tests) so that whenever x_i changes to $\overline{x_i}$ (due to fault), $F(x_1, \cdots, x_i, \cdots, x_n)$ will be different from $F(x_1, \cdots, \overline{x_i}, \cdots, x_n)$. In other words, the aim is to find input patterns for each fault occurring on x_i such that $dF(X)/dx_i = 1$.

If $F(X)$ is logically invariant under the complementation of x_i, then the Boolean function of $F(X)$ is *independent* of x_i. i.e. if

$$F(x_1, \cdots, x_i, \cdots, x_n) = F(x_1, \cdots, \overline{x_i}, \cdots, x_n)$$

$$dF(X)/dx_i = 0.$$

The following example illustrates how a Boolean difference of a switching function can be obtained:

Example: Consider the logic circuit shown in Figure 3.6. The functional equation of this circuit can be given as

$$F(X) = x_1x_2 + x_3x_4$$

$$\frac{dF(X)}{dx_3} = \frac{d(x_1x_2 + x_3x_4)}{dx_3}$$

$$= \overline{x_1x_2}\frac{d(x_3x_4)}{dx_3} \oplus \overline{x_3x_4}\frac{d(x_1x_2)}{dx_3} \oplus \frac{d(x_1x_2)}{dx_3}\frac{d(x_3x_4)}{dx_3}$$

$$= \overline{x_1x_2}\frac{d(x_3x_4)}{dx_3}$$

$$= \overline{x_1x_2}\left(x_3\frac{dx_4}{dx_3} \oplus x_4\frac{dx_3}{dx_3} \oplus \frac{dx_3}{dx_3}\frac{dx_4}{dx_3}\right)$$

$$= \overline{x_1x_2}x_4$$

This shows that a fault on x_3 will cause the erroneous output only if $\overline{x_1x_2}x_4 = 1$. i.e. if x_1 or x_2 (or both) are equal to 0 and x_4 equal to 1. This can be verified by inspection of Figure 3.6.

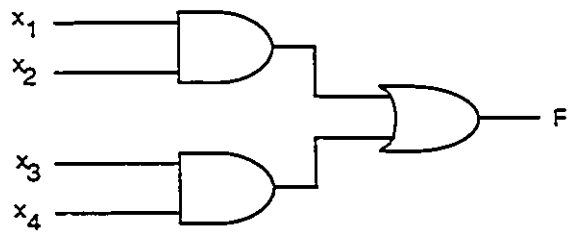


Fig. 3.6. Circuit example.

The Boolean difference method cannot be used only to derive tests for input lines but also for internal line faults.

If a combinational circuit realizing the function *F(X)* has *h* as an internal wire in the circuit then the tests for *h* can be found by expressing *F* as a function of *h*, $F(x_1, x_2, \cdots, x_n, h)$, and *h* as a function of inputs, $h(x_1, x_2, \cdots, x_n)$. Tests for *h* s-a-0 and s-a-1 are given by $h\frac{dF}{dh}$ and $\bar{h}\frac{dF}{dh}$ respectively.

The conventional Boolean difference method, in general, is not capable of deriving tests for all the internal nodes of a logic network. Therefore to develop a complete test set which will detect both input line and internal node faults, it is necessary to find tests which will exercise completely each and every path connecting a primary input to the primary output. This can be carried out by using the *partial Boolean difference* technique. A conventional Boolean difference expression may be formed by concatenating individual Boolean differences. For example, if *Z = f(y)* and *y = f(x)*, then

$$\frac{dZ}{dx} = \frac{dZ}{dy}\frac{dy}{dx}$$

where dZ/dx is termed the partial Boolean difference with respect to x.

The Boolean difference method offers a direct analytical method of obtaining all test patterns. It is a complete algorithm and does not require any trial and error. However, the method is costly in terms of computation time and memory requirements and is applicable to circuits of reasonable size only.

5) D-Algorithm

The D-algorithm[4,21] is an algorithmic method for generating tests for nonredundant combinational circuits. The problems with most of the methods discussed before were largely solved by the D-algorithm. This method generates the conditions required at the inputs to carry out the test. This is done through a *back trace*, reflecting the signal conditions back to input conditions. The D-algorithm guarantees a test if there exists such a test for detecting a fault in the circuit.

This algorithm uses the symbols D and $\overline{D}$. The symbol D may assume only one value 0 or 1 and $\overline{D}$ takes on the value complementary to D. The behavior of the gates can then be expressed in terms of D-cubes for propagation and for testing. The *propagation D-cubes* of a gate causes the output of the gate to depend only on one or more of its specified inputs. This in turn propagates the fault on these inputs to the output. The *primitive D-cube of a fault* specifies the existence of a given fault in the circuit. It is basically an input pattern which carries the influence of a fault to the gate output. To build sensitized paths in the circuit a term called *D-intersection* is used which transmits the fault from

an input to the primary output. A detailed discussion of the D-algorithm can be found in [21].

The main advantage of the D-algorithm over other existing test generation techniques is that it does not require the complete circuit under test to be represented and stored in terms of tables, formulas, or equations. This makes it possible to apply it to large circuits without facing storage problems and requiring large computation time.

6) PODEM

PODEM(path oriented decision making) is another algorithmic method of test generation for combinational logic circuits. It is more efficient than the D-algorithm. It uses five valued logic $(0,1,x,D,\overline{D})$ for test generation and is basically derived from the D-algorithm.

The PODEM does not require much trial and error before a test is found. This is because the variety of propagation paths and the attendant consistency operations that are required in the D-algorithm leading to a waste of effort if a given fault is undetectable are eliminated in PODEM. A detailed description of the PODEM algorithm can be found in [21].

3.2.2. Probabilistic approach

The above deterministic methods of test generation are often unable to cope with complexity of LSI and VLSI chips. These methods may become prohibitively expensive for large and complex logic circuits as the computation time required by these methods to generate a test set that detects all single and some multiple stuck-at faults increases with the complexity of the logic circuit. The *random test generation method* which comes under the category of probabilistic test generation can overcome some of the inherent disadvantages of the deterministic methods.

1) Random test generation in digital logic circuits

There are two types of random testing procedures. In the first, known as the *random test generation method*, a set of input patterns selected from randomly generated patterns is applied to the primary inputs of the circuit under test such that this test set will detect all of the single faults in the circuit. The random input patterns can easily be generated by using a computer which can produce *0's* or *1's* with any given probability. The second type of procedure consists of the requirement of the application of identical inputs to the circuit under test and a fault-free circuit(or its digital simulator), and then a comparison of the outputs of these circuits. In either case, the number of random input patterns that will detect most of the single faults in the circuit is needed. The random testing may be seriously considered as an alternative only if this number is less than 2^n, for an n input circuit. The number of random input patterns required can be readily found through logic analysis. But the effort involved in such a preset analysis

may be of the same order as that needed for generating test patterns by other methods if the circuit is very large. However, the method, if used properly, can provide good results at a reasonable cost.

A comprehensive analysis of random test generation methods for combinational logic circuits can be found in [1,2,21,28]. In conventional random test generation methods, the effect of a failure in a logic network is propagated to the network output by applying random patterns to the primary inputs of the network. The outputs of the fault-free and the faulty networks are than compared by using a *simulator* as shown in Figure 3.7. If the output values differ, the applied random input pattern is retained as a *test*. For complete fault detection, the process is continued until every fault in the network has been detected by *at least* one input pattern. The accuracy of the random testing method largely depends on the input bit stream *length*.

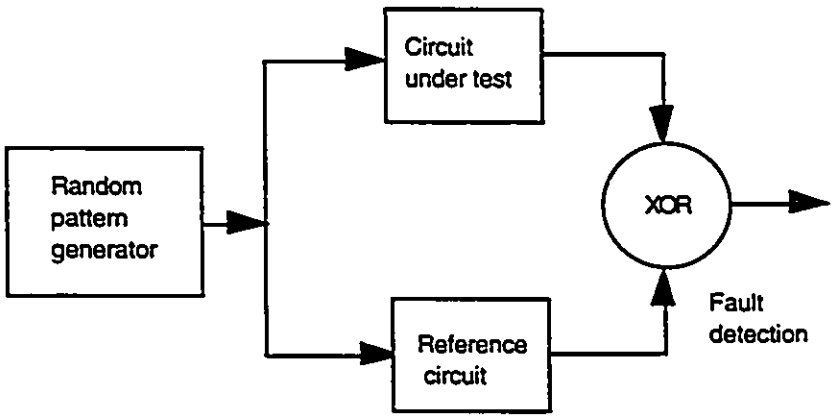


Fig. 3.7. Principle of random testing.

In order to solve the aforementioned problems concerning the random test generation in combinational logic circuits there exist approaches that take help of the parameters such

as *latency* and *detection surface*, or the models of circuit structure. Each one of these approaches has a distinct direction of its consideration, but they surely demonstrate the fact that the *detection probability* is highly related to the input bit stream *length* and the input bit stream *probability distribution*. In the following we first discuss the circuit model approach in brief and then the latency and detection surface approaches to project the basic concepts of random testing in digital logic circuits. In random testing it is important to know the number of random input patterns that will complete the overall circuit testing with a high *measure of confidence*.

For the purpose of probabilistic analysis of random test generation method in combinational logic circuits, Agrawal and Agrawal[2] used a simple tree structure comprised of identical *n*-input NAND gates as their network model, as shown in Figure 3.8. The lines from the primary inputs to the primary outputs are identified according to the levels at which they occur. All the primary inputs are at level zero, and every time as the signal path goes through a gate, the levels are incremented by one. The level of a gate is the same as the level of its output line. Using two-valued logic(0,1), the probabilities of a line at a particular level *k* are obtained as a function of *k* and the number of fanins *n* of the gate.

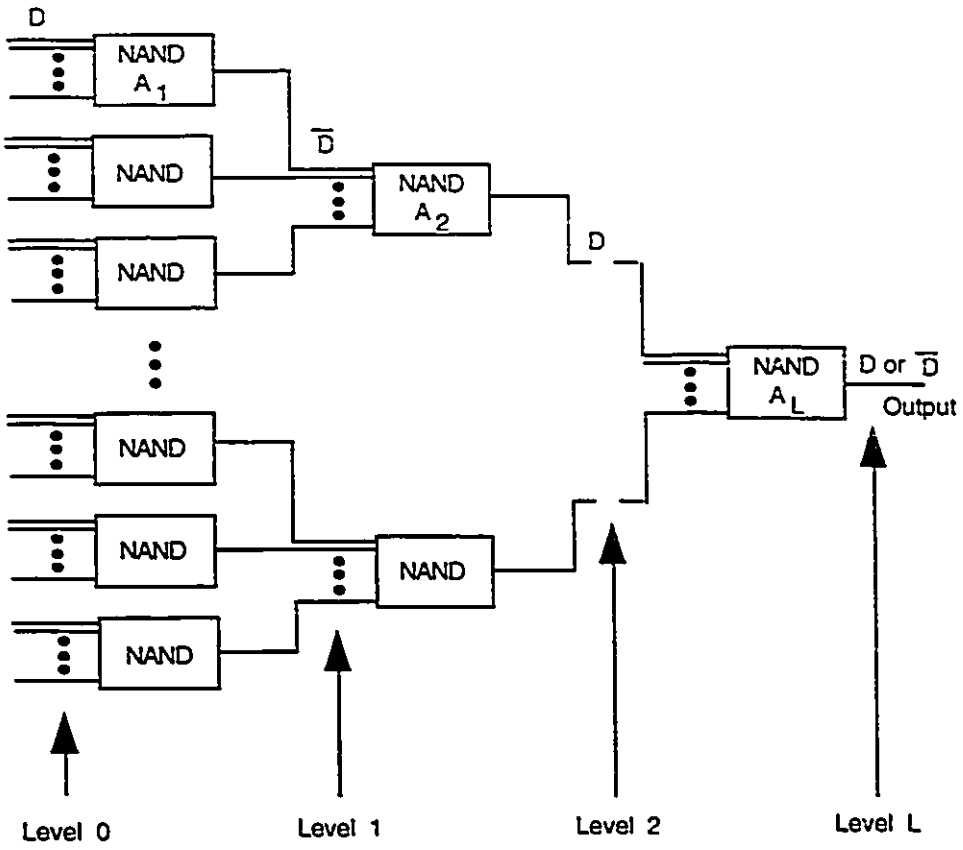


Fig. 3.8. NAND tree network used as a model of analysis.

Assuming that all the inputs of a gate belong to the same level, the probability of a logical 0 occurring at level k is given by:

$$P_k^0 = (P_{k-1}^1)^n = (1 - P_{k-1}^0)^n.$$

This is the probability of having n *1's* at the *(k-1)th* level. Clearly, the probability of a 1 occurring at the level k is obtained as:

$$P_k^1 = 1 - P_k^0 = 1 - (P_{k-1}^1)^n.$$

At any level k the number of fanins can be expressed as $n(k)$. In general, $n(k)$ may be regarded as a random variable.

For the fan-out-free NAND tree considered in Figure 3.8, the problem of finding the probability of sensitizing a path of length L reduces to one of finding $n-1$ 1's at the inputs of the each of the L gates along the path, since the effect of the fault is present in one input of each gate. In the notation of the D-algorithm of Roth *et al.*, this is similar to having a D or $\overline{D}$ at one input of each gate, D($\overline{D}$) being 1(0) for the fault-free(normal) network and 0(1) for the faulty one.

Considering only those faults involving the primary inputs, since every other fault in a fan-out-free network is indistinguishable from an input fault, the probability of propagating a D or $\overline{D}$ through a path of length L is given by

$$P(L) = P(0) \prod_{k=0}^{L-1} (P_k^1)^{n-1},$$

where $(P_k^1)^{n-1}$ is the probability of having $n-1$ 1's at level k, and $P(0)$ is the probability of D or $\overline{D}$ occurring at the fault site. L is the level of the primary output. $P(0)$ is assumed to be 1 for the primary input faults. $P(L)$, the probability of sensitizing a path of length L by random input, is called as the *detection probability*.

$P(L,M)$, the probability of sensitizing a path of length L by at least one out of M independent random input patterns applied to the network for the detection of a fault, is then expressed as

$P(L,M)$ = 1 - (probability of none of the M patterns sensitizing the path)

$$= 1 - \{1-P(L)\}^M,$$

which, on solving for M, gives:

$$M = \log \{1 - P(L,M)\} / \log \{1 - P(L)\}.$$

$P(L,M)$ is considered as a *measure of confidence*.

Agrawal and Agrawal[2] has presented experimental results for circuits with different maximum number of levels L.

Probabilistic models are used by many to characterize the behavior of digital circuits in the presence of faults. Some of the models detect intermittent faults, while some others detect permanent or nontransient faults by random patterns. An *intermittent fault* is a fault which, when exists in the system, may be *active* at one instant of time causing the system to malfunction, but may be *inactive* at another instant allowing the system to operate correctly. A *permanent* or a *nontransient fault* is a fault which remains permanently in the system. To characterize the behavior of digital systems in the presence of intermittent faults, Su *et al.*[35] proposed a *continuos parameter Markov model* with two states.

A fault in a digital circuit does not cause an immediate error in the output of the circuit. There is always a *delay* between an occurrence of a fault and an appearance of the first error in the circuit output. This delay is termed as the *error latency* of the fault. The error latency, an attribute of the fault, depends on the circuit, the fault, and the input pattern applied to the circuit. The random testing of digital circuits is analyzed by Shedletsky and McCluskey[34] using the *error latency model*. This model is

proved to be very useful for the analysis of random testing procedures in sequential circuits.

The problem of fault detection in combinational logic circuits by applying random input sequences to a circuit under test is discussed by Parker and McCluskey[28,29]. It is shown that the probability of detection of a fault depends upon the probability of inputs. While the best probability of inputs (being the same for all the inputs) may be found for some faults, it may be rather difficult to find the best probability of inputs for the whole set of possible faults. The application of error latency model to the random test generation and random testing leads to the conclusion that random testing may not be a good method for faults which are detectable only by one among all the possible 2^n input patterns. Most of the common networks do not reach this upper bound. Hence, if the testing time does not exceed a reasonable limit, random testing may still be worthwhile.

In a paper by David and Blanchet on random fault detection of combinational networks, *detection surface* is introduced as a characteristic parameter of combinational logic network. A distinction between *testing quality* and *detection quality* discussed there indicates that rather than considering the probability of testing for every fault, only the probability of testing for the most difficult to detect fault can be considered. The detection surface of a fault f_i denoted by σ_i, is the number of different input vectors which can detect the fault f_i. The detection surface of a network, denoted by σ, is the detection surface of the fault which is the most difficult to detect, i.e. $\sigma = \min_i \sigma_i$. The length of the input sequence applied to the network can be obtained from the two characteristic parameters n and σ of the network.

2) Fault detection in sequential digital logic circuits using continuos parameter Markov model

In a recent paper by Das *et al.*[9], the problem of detecting permanent faults in sequential logic circuits by random testing is analyzed. A *continuos parameter Markov model* with three states designated as: state 0, state 1, and state 2, as shown in Figure 3.9, describes the behavior of a sequential logic circuit in the presence of a fault and under the application of random input vectors. The state transition probabilities depend linearly on the infinitesimal time step Δt; λ_i s and μ_j s are the constants of proportionalities. These probabilities increase as the time step Δt increases. Once the random pattern is applied, the circuit may stay in any one of the above three states. The circuit is in state 0, if the fault exists, but causes no error output and error state transition; the circuit is in state 1, if the fault causes an erroneous output; and the circuit is in state 2, if the fault causes only error state transition, but no erroneous output in the output terminals. Different state transition probabilities have been defined to represent the interaction among the three states 0, 1, and 2 for the infinitesimal time step Δt in the circuit.

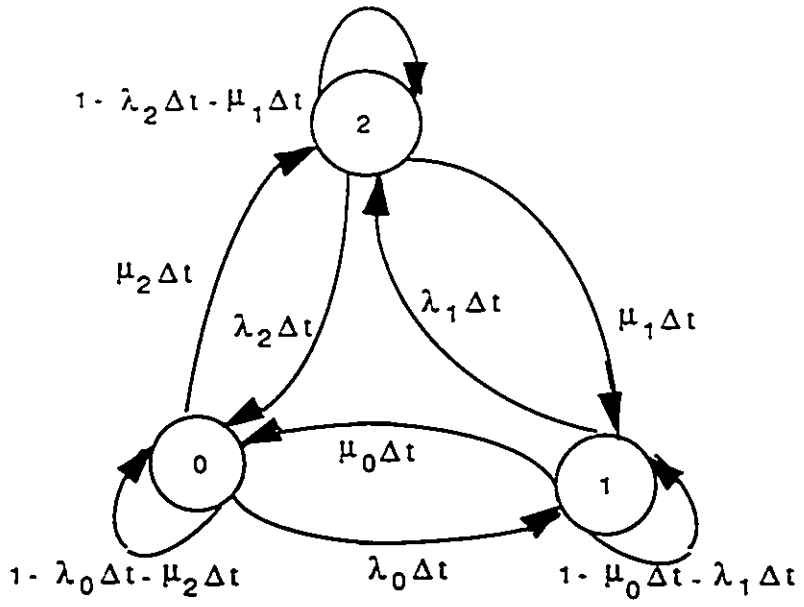


Fig. 3.9. Continous parameter Markov model.

The parameters of the model depend on the circuit, and faults under consideration. Once the fault-free state table and faulty state table(corresponding to the faults specified) of the circuit are constructed, all the parameters of the model can be obtained by computer simulation under constant input signal probabilities. When the parameters of the model are obtained, the system differential equations involving the state probabilities can be written on the assumption that at time $T = 0$, the faults cause no error output and error state transition in the circuit. These equations can then be solved for the necessary condition for detecting a fault quickly with a given probability. This approach does not require the construction of a product state table corresponding to the fault-free state table and its faulty version, which is particularly difficult while dealing with large circuits.

3.3. Fault Simulation

The second phase of the VLSI testing problem is the difficulty of fault simulation. *Fault simulation* is the process by which the fault coverage is determined for a specific set of input test patterns. In particular, at the conclusion of the fault simulation, every fault that is detected by a given set of test patterns is listed. For a given logic network with 1,000 two input logic gates, the maximum number of single stuck-at faults which can be assumed is 6,000. Some reduction in the number of single stuck-at faults can be achieved by fault equivalence. However, the number of single stuck-at faults needed to be assumed is about 3,000. Fault simulation is the process of applying every given test pattern to a fault-free network and to each of the 3,000 copies of the fault-free network containing one, and only one, of the single stuck-at faults. Thus fault simulation, with respect to run time, is similar to doing 3,001 fault-free network simulations.

In all commercial simulators an effort is made to reduce the computation time by simulating more than one fault in one pass for a given input vector. *Parallel, deductive,* and *concurrent* simulations are the known methods of fault simulation. In *parallel fault simulation*[38], *W-1* different faults are simulated in parallel in each pass; hence, the name of the method. A total of *F* faults would be simulated in *F/(W-1)* passes, where W is the word size of the computer.

Techniques are available to reduce the complexity of fault simulation. The solution is the *deductive fault simulation* [38]in terms of reducing the number of passes. It just needs one pass for simulation independent of the number of faults simulated. The idea behind this technique is to associate with each line a list of only those faults

sensitized to that line by the simulated input vector. Simulation of a gate requires deducing the fault list at the gate output from the input fault lists.

Some undesirable characteristics of deductive method can be overcome by *concurrent fault simulation*. In concurrent fault simulation a good circuit is simulated in its entirety, but a faulty one is simulated only for gates whose *states* differ from their good circuit states.

3.4. Conclusions

The rapid growth of LSI/VLSI devices has resulted in the development of highly complex computing systems. This, combined with the fact that many applications now demand error-free computation, has considerably increased the level of desired system reliability. Test generation has also become one of the most complicated, costly, and time-consuming problems in VLSI system design. A more pragmatic approach to the solution of the problem will surely be one to conceive of methods that are less formal and exhaustive, and from that point of view random testing appears preferable to deterministic testing, specially for LSI/VLSI circuits. The random test generation techniques can reduce computation costs and time, and can be effectively used for faliure detection in relatively complex digital systems.

Chapter 4

AN OVERVIEW OF VARIOUS OUTPUT COMPACTION TECHNIQUES

4.1. Introduction

Classical testing of combinational circuits requires a bit by bit response of the fault-free circuit to the test set. For most practical circuits implemented today, the large storage requirement for the response makes such a test procedure very expensive. Techniques for reducing the volume of output data are called *output compaction techniques* and their use is usually called *compact testing*. In compact testing, the output response pattern is passed through a circuit called a *compactor* that has fewer output bits than input bits. The output of the compactor is called the *signature* of the test response. A fault is detected if the signature of the faulty circuit differs from the signature of the fault-free version of the circuit. The aim is to reduce the number of bits that must be examined to determine whether the circuit under test is faulty.

The signature and the associated compaction technique should meet the undermentioned four qualitative guidelines:

1. The technique must be simple enough so that it can be implemented as part of the built-in test circuitry.

2. The implementation should be fast enough so that test time is not a limiting factor.

3. To minimize the reference signature storage volume, the compression technique should provide a logarithmic compression of the output response data.

4. The compression method must not lose information.

In general, a fault will go undetected if none of the input test patterns produces an incorrect circuit output in the presence of the fault. With output response compaction, it is also possible for a fault to go undetected even though the output response differs from the fault-free response. This will happen whenever the output response from a faulty circuit produces a signature that is identical to the signature of a fault-free circuit. This phenomenon is called *aliasing* or *masking*[21, 39].

An output response signature of a faulty circuit that is identical to the fault-free signature is called an *alias*.

When a common estimating framework is used, the relative masking probabilities of various output compression techniques can be used as one measure of their efficiency.

The most common system designs for compact testing have three parts, and are shown in Figure 4.1.

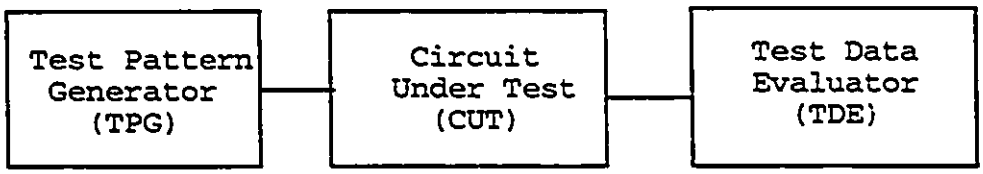


Fig. 4.1. Self-testing architecture for compact testing.

The test pattern generator generates the test patterns from the testing method being used, and then applies them to the inputs of the circuit under test(CUT). The test data evaluator receives the output from the CUT, evaluates it, computes the test results, and then transmits these results for external observation.

Because masking is difficult to measure in arbitrary circuitry with arbitrary test sets, some output compaction techniques restrict the circuit design or require special test sets or both.

Many output compaction techniques, viz. the signature analysis, transition count, parity bit signature(single- and multiple-output), syndrome and space compression are available for compact testing. This chapter describes these techniques that apply to built-in self-testing in each of the three categories:

1. Techniques that neither require special test sets nor circuit modification.

2. Techniques that do use special test sets, but do not require circuit modification.

3. Techniques that require special test sets as well as circuit modification.

4.2. Signature Analysis

Signature analysis technique[12,13,21,26] for compact testing is heavily reliant on planning done in the design stage. It is a technique that detects errors in data streams caused by hardware faults. It uses a unique data compression technique which reduces long data streams into signatures. By

applying known input sequences to a digital system, unique signatures can be obtained at various nodes in the system. These correct signatures can be recorded and later compared with the signatures obtained at the same nodes of a faulty system. Any difference in signature at a node indicates that the node is not operating as expected. The cause of an incorrect signature may be discovered by tracing back through the circuit and observing the signatures at appropriate nodes; the fault giving rise to the error can finally be located through the process of elimination. Thus the signature analysis technique is a digital equivalent of the voltage and waveform checking of analog circuits, and is used in the same way. The degree of fault coverage provided by signature analysis technique does not depend upon the order of test patterns(for combinational circuits).

The integral part of the signature analysis approach is a linear feedback shift register. Signatures can be generated from data streams by feeding the data into an *n*-bit linear feedback shift register. The feedback mechanism consists of EXORing selected taps of the shift register with the input serial data as shown in Figure 4.2. After the data stream has been clocked through, a residue of the serial data is left in the shift register. This residue is unique to the data stream and represents its *signature*. Though another data stream may differ by only one bit from the previous data stream, but its signature may be different from the previous one. To form the signature of a data stream, the shift register is first initialized to a known state (normally, the all-0 state) and then shifted using the data stream.

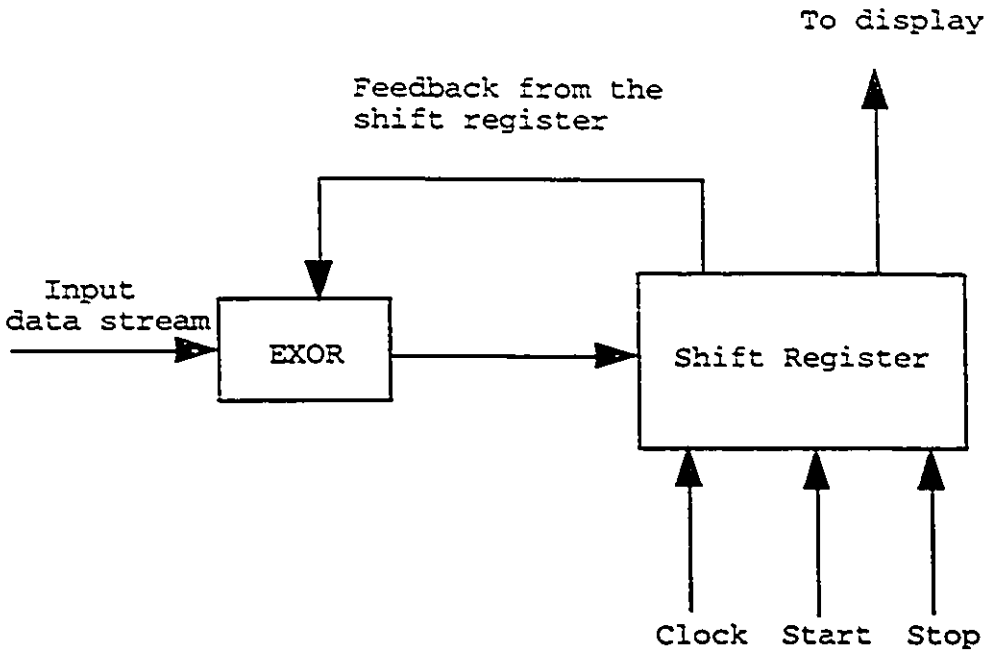


Fig. 4.2. Signature analyzer circuit.

A signature analyzer circuit as shown in Figure 4.2[21] has three control signals, viz. *Start*, *Clock* and *Stop*. These are either positive or negative edge-triggered signals. The *Start* signal resets all the bits in the shift register to logic 0 and initiates the measurement window. During this time window the incoming data to the signature analyzer circuit are entered into the shift register using the *Clock* signal. Finally, the *Stop* signal terminates the measurement window.

An n-bit signature generator can generate 2^n signatures. However, many input sequences can map into one signature. This mapping gives rise to fault masking errors, because a signature generated from an input sequence carrying fault information may map into the same signature as the good sequence. The signature analyzer has a probability of $1-2^{-r}$ of detecting all errors, where the error detection probability is strictly a function of r, the length of the shift

register. The probability of fault masking errors will be low if a signature generator has many stages, and thus capable of generating a large number of signatures.

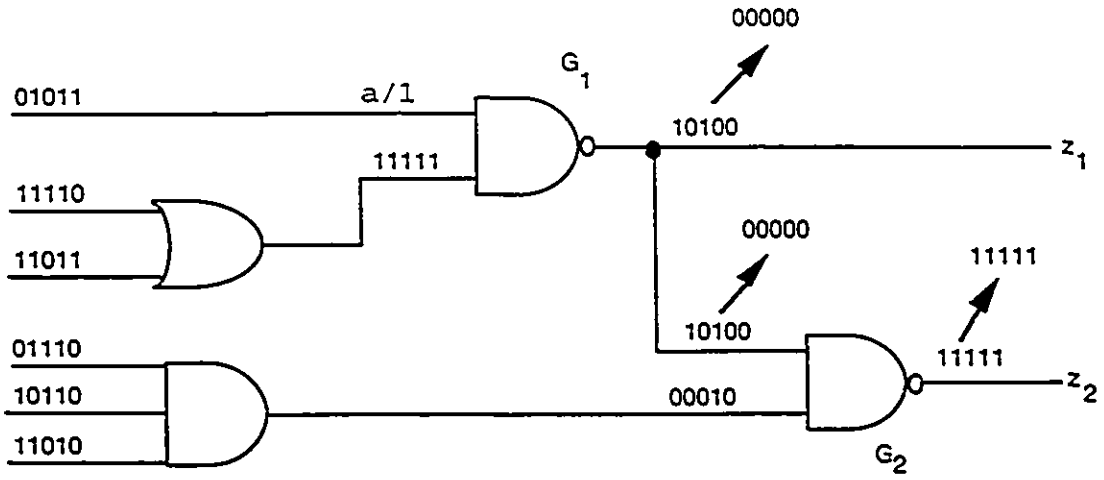
The signature analysis technique identifies faults by tracing the signal flow in a circuit from input to output. If the signature at a circuit node does not match the empirically determined correct signature, one of the components connected to the node is faulty. Tracing from input to output generally proves useful for field testing.

Signature analysis has been accepted as a valuable technique in locating hardware faults in digital systems. It offers unique alternatives to other test methods and can provide high test confidence at reasonable cost for both production and field testing.

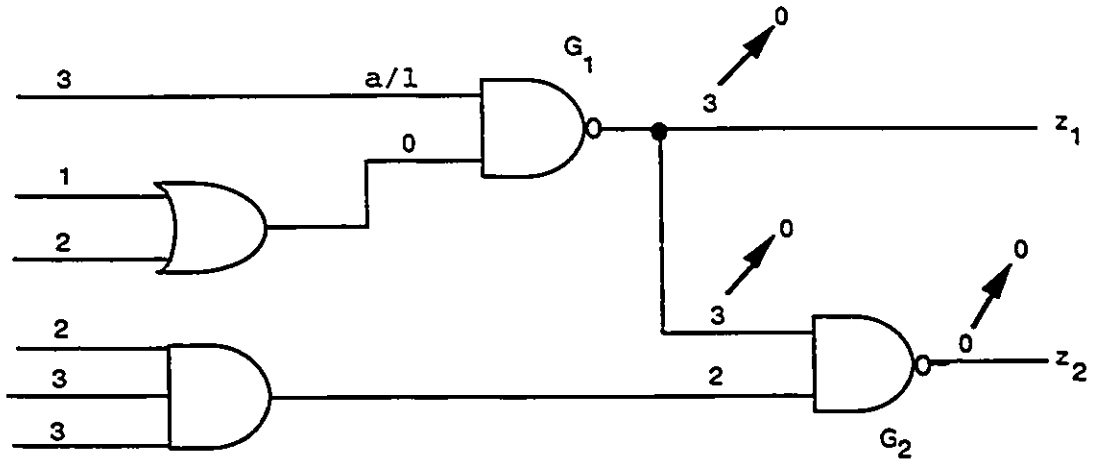
4.3. Transition Count Testing

The *transition count*[1,14,21,31] is defined as the number of times the signals forming the output *Z* change value from *0* to *1* and *1* to *0* for a given input sequence. For example, if a response sequence is *Z=101011*, then the transition count is *c(Z)=4*. In transition count testing, as in conventional testing, a predetermined sequence of test patterns is applied to the circuit under test(CUT). The response sequence *Z* of the circuit appearing at some selected test point is then monitored. This is done in a way which is different from that in conventional testing. Instead of recording the entire response sequence *Z*, only the transition count *c(Z)* of *Z* is recorded. This transition count is then compared with the one expected, that should appear at the test point if the circuit under test is fault-free. If the two transition counts differ, the circuit is declared faulty; otherwise, it is fault-free. A repeated use of this test

procedure for other test points in the circuit under test gives a high degree of fault detection and isolation. This method of compact testing is referred to as *transition count testing* or TC testing.



(a)



(b)

Fig. 4.3. (a) Response to test sequence of length 5.
(b) Changes in the corresponding trasition counts.

Consider the circuit shown in Figure 4.3. In Figure 4.3(a), the response sequences at various nodes of the circuit resulting from the application of a test sequence *S*

of length 5 are shown. Figure 4.3(b) shows the corresponding TCs. A fault f = line a stuck-at-1 in the circuit changes the transition counts at certain nodes in the circuit (shown by arrow). As can be seen in the diagram, the response sequence on Z_1 changes from 10100 to 00000. This change is reflected in TC from 3 to 0 in Figure 4.3(b), resulting in the detection of the fault f, if S is used as the test sequence and Z_1 as the test point. The fault f does not changes the response sequence at Z_2. Therefore Z_2 is not a suitable test point for detecting f for the given test sequence.

The basic test circuitry needed in a transition count generator is a counter and a transition detector. The main advantage of transition count testing is that, only the TCs and not the entire response sequences(both the fault-free and faulty) at any test point are required to be stored. This results in the reduction of data storage requirements. However, this compaction technique may give rise to fault-masking errors, because most transition counts corresponds to more than one sequence. Hence, there is a possibility that a fault may go undetected. However, this problem becomes less predominant as the sequence length increases.

The main disadvantage of transition count testing is that the fault coverage achieved by a test pattern depends on the order in which they are applied. This may not be a major problem in testing combinational circuits but in the case of sequential circuits it matters. Besides, with multiple-output circuits a particular test pattern may give different fault coverage at different outputs; the optimal test patterns can only be determined by using simulation techniques.

4.4. Parity Bit Signature

Built-in self-testing[26,30] considers the testing problem during the design stage of the digital device. The test patterns used in built-in self-testing can be either random or exhaustive. Exhaustive test patterns are relatively easy to generate and excites all faults. Since all inputs are involved, one can concentrate on the overall functional behavior rather than just that corresponding to certain selected inputs. The major problem in testing lies in the determination of the relationship between the faults occurring in the circuit and the observed error indications. A parity bit checking[7] coupled with exhaustive testing improves this relationship. The design of parity testable combinational circuits is discussed in [5].

A parity bit signature was proposed by Akers[3] for single-output circuits. A possible extension of the idea to multiple-output circuits was considered by Jone and Das[7]. Both, the single-output and multiple-output parity bit signatures are briefly discussed in the following sections.

4.4.1. Single-output parity bit signature

In this section we will first mention some of the basic and important properties of the parity bit, and then we will discuss the parity bit signature and the properties it possesses.

Properties of the parity bit

The parity of a function is the EXOR sum of the entire output sequence obtained, when the exhaustive test patterns are applied to the circuit under test.

If $F(x_1, x_2, \cdots, x_n)$ is *independent* of a variable x_i, then its parity (relative to $(x_1, x_2, \cdots, x_n)$) will always be 0.

or conversely,

If the parity of the function *F* is equal to 1, then *F* must be a *dependent* function of all the variables.

These are the two basic and important properties of the parity bit. The other properties of the parity bit involves specific functions, namely the product, inclusive sum and the exclusive sum, parity relationship between two functions *F* and *G* of the same set of *n*-variables and the behavior of the parity bit under various transformations. Some properties involve functions having different sets of variables.

A parity bit signature

A parity bit signature[3,5] consists of the parity of the function itself, and *n* additional parity bits(for an *n*-input combinational logic circuit) corresponding to the functions obtained, for each of the input variables, when that variable is set to either 0 or 1.

Thus, in general, a parity bit signature of an *n*-variable function *F* is an *(n+1)*-element vector and can be defined as:

$$s(F) = \langle p_0, p_1, \ldots, p_n \rangle$$

where $p_0 = p(F)$ and $p_i(F) = p_i(F_b^i)$, $i = 1, 2, \ldots, n$ and $b = 0$ or 1.

The generation of a parity bit signature can be illustrated by a simple example given below in Table 4.1.

Table 4.1

Illustration of a Single-Output Parity Bit Signature

x_1	x_2	x_3	x_4	F	$F_0^{x_1}$	$F_0^{x_2}$	$F_0^{x_3}$	$F_0^{x_4}$
0	0	0	0	1	1	1	1	1
1	0	0	0	0		0	0	0
0	1	0	0	1	1		1	1
1	1	0	0	1			1	1
0	0	1	0	0	0	0		0
1	0	1	0	0		0		0
0	1	1	0	0	0			0
1	1	1	0	0				0
0	0	0	1	1	1	1	1	
1	0	0	1	1		1	1	
0	1	0	1	1	1		1	
1	1	0	1	1			1	
0	0	1	1	1	1	1		
1	0	1	1	0		0		
0	1	1	1	1	1			
1	1	1	1	0				
				s(F)= <1	0	0	1	1>

Figure 4.4[3] shows an implementation of a parity bit signature in which the *n+1* bits are directed into *n+1* flip-flops. The output *F* as well as the *n* inputs are involved.

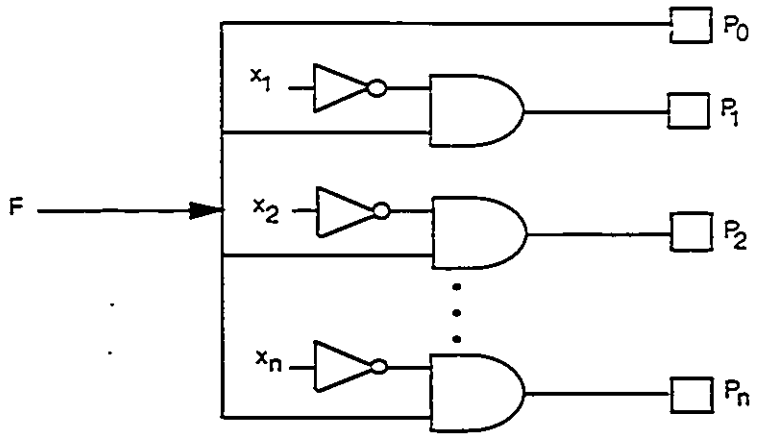


Fig. 4.4. Signature implementation.

Here, P_0 represents the parity of the function itself, and $P_1,\cdots,P_n$ are each the parity of the reduced function.

Properties of a parity bit signature

A parity bit signature for exhaustive testing possess the following properties[3,17].

1) It is a *functional signature*, that is, it is independent of the particular implementation involved.

2) It is *uniform*, that is, given an *n*-input combinational circuit, all 2^{n+1} possible *(n+1)*-bit signatures are equally likely to occur.

3) It is *test-order independent*, that is, the order in which 2^n tests are applied has no effect on the signature.

4) It is easily *implementable* and *effective* for input fault detection.

4.4.2. Multiple-output parity bit signature

A multiple-output parity bit signature[3,5,17] can be obtained by simply extending the concept of single-output parity bit signature, that is, generating a signature for each and every output separately. But excessive amount of hardware overheads make this approach basically impractical. For multiple-output circuits, two methods have been proposed for signature analysis. One is the serial signature analyzer which uses a multiplexer to direct each one of the circuit outputs in turn to the signature analyzer. The other is the parallel signature analyzer in which all the m network outputs are compacted in parallel.

A parity bit signature for an *n*-input and *m*-output combinational logic circuit can be generated by first EXORing all the outputs to generate a new output, which is then fed to a single-output parity bit signature generator.

As shown by Jone and Das, the multiple-output parity bit signature preserves all the desirable properties of the single-output parity bit signature.

An example of a multiple-output parity bit signature is given in Table 4.2.

Table 4.2

A Multiple-Output Parity Bit Signature

x_1	x_2	x_3	x_4	F_1	F_2	F	$F_0^{x_1}$	$F_0^{x_2}$	$F_0^{x_3}$	$F_0^{x_4}$
0	0	0	0	1	0	1	1	1	1	1
1	0	0	0	0	0	0		0	0	0
0	1	0	0	0	1	1	1		1	1
1	1	0	0	0	1	1			1	1
0	0	1	0	0	0	0	0	0		0
1	0	1	0	0	0	0		0		0
0	1	1	0	0	0	0	0			0
1	1	1	0	0	1	1				1
0	0	0	1	1	0	1	1	1	1	
1	0	0	1	1	1	0		0	0	
0	1	0	1	1	0	1	1		1	
1	1	0	1	0	0	0			0	
0	0	1	1	1	0	1	1	1		
1	0	1	1	0	1	1		1		
0	1	1	1	1	1	0	0			
1	1	1	1	0	0	0				
				s(F)	=	<0	1	0	1	0>

4.5. Syndrome and Syndrome Properties

The *syndrome* of a switching function is defined as $s = K/2^n$ where K is the number of minterms realized by the function and n the number of binary input lines[33]. Obviously, the syndrome is a functional property and is independent of the particular realization involved. Clearly,

$$0 \leq s \leq 1,$$

where the bounds are attained by the two constant functions.

The syndrome of various n-input gates, following Savir, are given as :

AND gate : $s = 2^{-n}$
OR gate : $s = 1 - 2^{-n}$
EXOR gate : $s = 1/2$

It is necessary to find input-output syndrome relations between various interconnected components of a logic circuit. These are discussed in detail by Savir and are useful in determining the syndrome of any fan-out-free combinational circuit. Incidentally, the input-output syndrome relations of networks terminating in an INVERTER, OR gate, AND gate or EXOR gate are similar to the input-output signal probability relations reported in Parker and McCluskey [28,29]. Similar results are discussed in [33] for interconnected blocks with shared or conjoint inputs.

The test procedure as shown in Figure 4.5 below basically consists of applying exhaustively all input combinations to the CUT and recording the syndrome, which can be implemented by a counter. If the actual syndrome equals the expected syndrome, the circuit is considered fault-free; otherwise, a fault is detected and the procedure stops. The approach has the advantage of requiring only one reference and of avoiding the costly test generation process. Since it is not necessarily true that a fault-free and a faulty circuit should have different syndromes, special design techniques are required in order to make a circuit syndrome-testable, if it is not so originally (please see subsequent work of Savir in this context in [32]).

A circuit is syndrome-testable if the syndrome of any faulty version of the circuit, induced by a single stuck-at fault, does not equal the syndrome of the fault-free circuit.

Savir made a detailed study on the problem of syndrome-testability and syndrome-testable designs for combinational circuits.

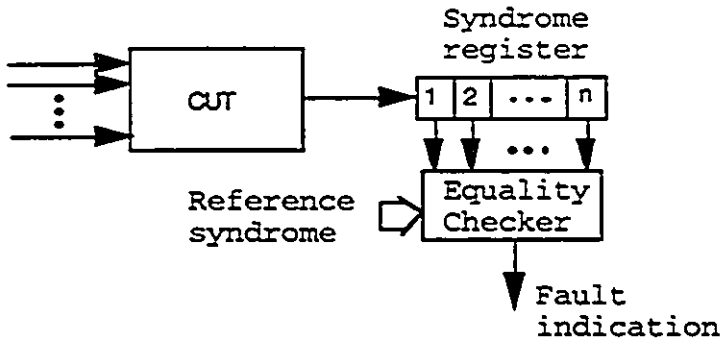


Fig. 4.5. Test procedure following Savir[7].

Let us consider a simple example from Savir.

Example: Consider a function $F = xz + y\bar{z}$. The fault-free syndrome is 1/2. The faulty syndrome induced by the fault *z/0* (line *z* stuck-at-0) is also 1/2. Hence *F* is <u>not</u> syndrome-testable.

However, the function *F* may be made syndrome-testable by inserting one extra input ω to realize the new function

$$F_1 = \omega xz + y\bar{z}$$

During normal operation the input ω is fed with a constant logic 1, while for testing purposes it is used as a valid input. The circuit is now syndrome-testable since

$$s(F_1) \neq s(F).$$

Note that a two-level irredundant circuit which realizes a unate function in all its variables is always syndrome

testable. Also, every two-level irredundant combinational network can be made syndrome-testable by attaching control (extra) inputs to the AND gates.

4.6 Space Compression

Compression techniques can be divided into two classes: *time compression* and *space compression*. The output data compression techniques discussed above all relate to time compression.

The basic idea behind space compression is to produce a single signature or a limited number of signatures for multiple-output circuits by using a space compressor as shown in Figure 4.6, rather than producing an individual signature for each output. The sequences coming out of the space compressor are stored in a time compressor as signature. The main advantage of using a space compressor is the reduction of the chip area occupied by the time compressor, but suffers from the disadvantage that it may lose some useful information. However, if the information loss is tolerable (in many cases, the information loss is zero), it is still worthwhile because of the savings in hardware overhead.

Li and Robinson proposed a space compression technique called *hybrid space compression*[22]. Instead of using only EXOR gates as an output compression tool, HSC used AND, OR and EXOR gates as an output compression tree to compress the multiple outputs of the CUT into a single line. The authors reported significant reduction in the number of outputs and reference signatures with information loss within 0-20% as compared with separate signatures.

Subsequently, Jone and Das[16] proposed a modified scheme called *dynamic space compression* or DSC that

dynamically estimates the error probabilities of the outputs. Experimental evidence with DSC demonstrates that it works rather efficiently. The authors also developed a theory to predict the performance of general space compression methods whose validity was also shown by extensive simulation results. The reader might wish to read the original source for a more detailed discussion concerning DSC.

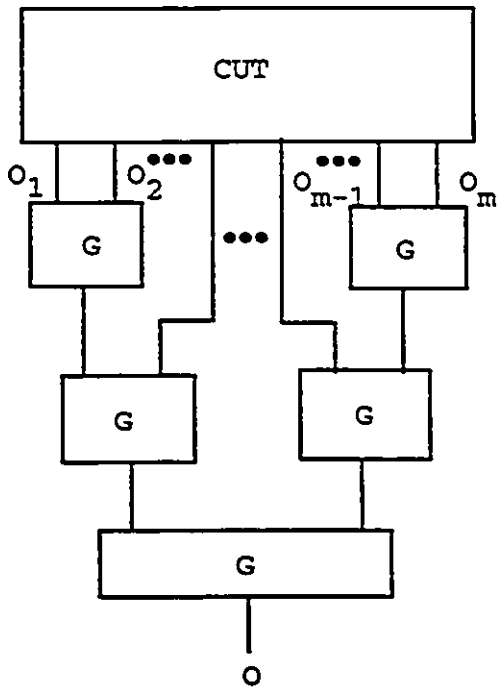


Fig. 4.6. Space compression using logic reduction network.

4.7. Conclusions

The classical testing of combinational circuits makes the testing procedure very expensive in terms of storage requirement, time and computational cost which increases with the circuit size. The use of various output compaction techniques discussed in this chapter reduces this problem to an acceptable alternative.

Though the abovementioned data compression techniques may give rise to fault masking errors or require some pin-penalty they still result in the reduction of the output data storage requirements. Most of these techniques are test-order independent(except for the transition count testing) and provides an excellent fault coverage in testing combinational logic circuits where test-order dependency may not be a major problem. The first four techniques relate to time compression, whereas the last technique concerns space compression and is well suited for multiple-output circuits where it provides a reduction of the chip area occupied by the time compressor.

Chapter 5

SYNDROME SIGNATURE

5.1. Introduction

Techniques for design for testability and built-in self-test consider the testing problem during the design stage of digital devices and have been found to be extremely effective, as noted earlier. The central idea behind built-in self-testing or BIST is to have the chip test itself. This technique generates test patterns and evaluates output responses inside the chip. It is already well recognized that BIST can significantly improve the testability of VLSI chips and save testing time as well.

The test patterns used in BIST can be either random or exhaustive. Exhaustive test patterns have the following advantages over random test patterns:

1) test patterns can be generated with relative ease;
2) no specific fault models are required; and
3) a high fault coverage can be achieved.

However, the main drawback of using exhaustive test patterns is the exponential growth of test length with increase in the number of inputs. This shortcoming in many cases can be overcome by using good circuit partitioning methods[18,27].

The test output responses in BIST are compressed by output response analyzer into a signature. Briefly speaking, the circuit under test (CUT) receives inputs, exhaustive or random, from a test pattern generator, and the output streams

from the CUT are fed into a data compressor, which converts the output streams into a signature of the CUT. The signature is compared with a known correct value, and faults are detected if a match does not occur. The various available output compaction techniques as have been proposed in the literature include parity-bit checking[3,5,7], transition count[14], one's count[33], Walsh coefficients[15,36], and linear feedback shift register or LFSR[13], and were discussed in Chapter 4. Based on these approaches, the compressed response data can be used to evaluate the correctness of the circuit under test (CUT).

In this chapter we have proposed an output data compression technique called syndrome signature particularly well suited for exhaustive testing of VLSI circuits. The syndrome signature is based on and an extension of the novel idea of syndrome of a circuit originally conceived by Savir[33] and is related to the number of minterms realized by the corresponding circuit switching function. In the following we will first define a syndrome signature and demonstrate its usefulness in exhaustive testing of combinational circuits. Some of the useful properties of syndrome signature will also be discussed. It will be shown that the use of the syndrome signature rather than just the primary syndrome though significantly increases the success of test coverage of existing circuits without going for a testable design with consequent increase in the number of pins in the circuit, cannot guarantee a complete fault coverage. A possible extension of the syndrome signature concept from single-output circuits to multiple-output circuits are then considered. Chapter 6 discusses implementation results involving both single-output and multiple-output circuits of reasonable sizes, while in Chapter 7 we give the results of implementation on ISCAS 85 benchmark circuits.

5.2. Single-Output Syndrome Signature

A single-output syndrome signature is defined as follows :

Definition: For an n-input single-output combinational switching function F with input variables $x_1, x_2, \ldots, x_n$, the syndrome signature $s(F)$ of F is given by an $(n+1)$-element vector:

$$s(F) = \langle s_0, s_1, \ldots, s_n \rangle$$

where $s_0 = s(F)$ and $s_i(F) = s_i(F_b^i)$, $i = 1,2,\ldots,n$, s_0 denoting the primary syndrome of the function F, while s_i representing the syndrome (or subsyndrome) of the subfunction (F_b^i) obtained by setting the ith variable in F equal to b (0 or 1).

As illustration, consider the following example.

Example: Assume a 4-input combinational function $F(x_1, x_2, x_3, x_4)$ defined by the truth table of Table 5.1. The syndrome signature $s(F)$ of F is given by the five-element vector:

$$s(F) = \langle 3/8,\ 5/8,\ 1/2,\ 1/2,\ 1/8 \rangle$$

where $s_0 = s_0(F) = 3/8$, and $s_i = s(F_0^i)$, $i = 1,\ldots,4$, s_0 denoting the primary syndrome of the function F, with s_i representing the syndrome of the subfunction (F_b^i) obtained by setting the ith variable in F equal to b (0) as shown in Table 5.1.

Figure 5.1 shows a straightforward implementation of this $(n+1)$-element signature in which the $(n+1)$ bit streams of interest are directed towards syndrome registers with

equality checkers as an extension of the scheme as used for primary syndrome by Savir [33].

Some desirable properties of syndrome signature are considered through the undernoted theorems:

Theorem 5.1: A syndrome signature $s(F) = \langle s_0, s_1, \ldots, s_n \rangle$ for an n-variable switching function is a *functional signature*.

Proof: The proof of the theorem follows obviously since a syndrome signature does not depend on the particular implementation involved in its realization.

Theorem 5.2: A syndrome signature $s(F) = \langle s_0, s_1, \ldots, s_n \rangle$ for an n-variable switching function F is *test-order independent*.

Proof: It is evident that the order in which the exhaustive test patterns are generated have no effect on the number of minterms realized by the function F as well as by the n subfunctions $s_i(F_0^i)$, $i = 1,2,\ldots,n$ and hence in the signature.

Theorem 5.3: A syndrome signature $s(F) = \langle s_0, s_1, \ldots, s_n \rangle$ of an n-variable switching function F is *nonuniform*.

Proof: For an n-input arbitrary combinational function F, the parity bit signature $s(F)$ of F is *uniform* since all of the 2^{n+1} possible $(n+1)$-bit signatures are *equally likely* to result. With most counting procedures, this is, however, not the case. Some signatures are more likely to occur than others, thus making the signature nonuniform.

To prove the nonuniformity of the syndrome signature, consider the first element s_0 of the syndrome signature $s(F)$.

Now s_0 is the syndrome of F and can have any value(nonnormalized) between 0 and 2^n.

Now there can be 2^{2^n} possible ways of assigning 2^n input combinations to F, and of these exactly $\binom{2^n}{K}$ result in an F with the same number of minterms or syndrome K. Hence the probability that the function F have the syndrome K,

where $0 \le K \le 2^n$,

is $\binom{2^n}{K} / 2^{2^n}$, which obviously is a function of K, unlike the probability of its parity $p(F)$ being 0 or 1, which is $1/2$.

We can similarly prove this for the other elements of the signature $s(F)$. Consider, for instance, the $(i+1)th$ signature element s_i, $0 \le i \le n$, in $s(F)$. Now

$$\begin{aligned} s_i &= s(F^{x_i=0}) \\ &= s(\bar{x}_i . F) \\ &= s[\bar{x}_i . F(x_1, x_2, \cdots, x_n)] \\ &= s[F(x_1, x_2, \cdots, x_i = 0, \cdots, x_n)] \\ &= s(F') \end{aligned}$$

where $F' = F(x_1, x_2, \cdots, x_i = 0, \cdots, x_n)$ is a *subfunction* of $(n-1)$ variables $x_1, x_2, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n$. Following exactly identical reasoning as before, we can again show that this $(i+1)th$ element of $s(F)$ has a probability dependent on its syndromwe value.

Thus we conclude that all the different $(n+1)$ elements of the signature can each have any value (normalized) in the domain {0,1} and therefore the signature is *nonuniform*.

Lemma 5.1: A syndrome signature $s(F) = \langle s_0, s_1, \ldots, s_n \rangle$ of an n-variable switching function F is easily implementable and especially effective for input fault detection.

Table 5.1

Syndrome Signature for a Four-Variable Function

x_1	x_2	x_3	x_4	F	$F_0^{x_1}$	$F_0^{x_2}$	$F_0^{x_3}$	$F_0^{x_4}$
0	0	0	0	1	1	1	1	1
1	0	0	0	0		0	0	0
0	1	0	0	0	0		0	0
1	1	0	0	0			0	0
0	0	1	0	0	0	0		0
1	0	1	0	0		0		0
0	1	1	0	0	0			0
1	1	1	0	0				0
0	0	0	1	1	1	1	1	
1	0	0	1	1		1	1	
0	1	0	1	1	1		1	
1	1	0	1	0			0	
0	0	1	1	1	1	1		
1	0	1	1	0		0		
0	1	1	1	1	1			
1	1	1	1	0				
				s(F)= ⟨	3/8	5/8	1/2	1/2	1/8 ⟩

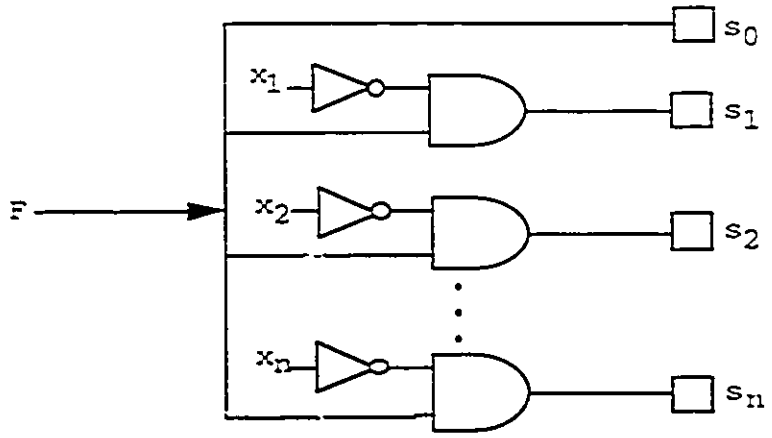


Fig. 5.1. An implementation of single-output syndrome signature.

The main objective as mentioned in using a syndrome signature rather than just the primary syndrome is to achieve fault coverage of existing circuits without taking resort to a testable design which is impossible for off-the-shelf circuits. To illustrate this, we consider the following example from Savir [33].

Example: Consider the function $F(x,y,z) = xz + y\bar{z}$. This function is not syndrome-testable for fault *z/0* (line *z* stuck-at-0). However, the circuit is syndrome signature-testable for the same fault. The fault-free and faulty signatures of the function are given in Tables 5.2 and 5.3, respectively.

Table 5.2

Fault-Free Signature of Function $F = xz + y\bar{z}$

x	y	z	F	F_0^x	F_0^y	F_0^z
0	0	0	0	0	0	0
0	0	1	0	0	0	
0	1	0	1	1		1
0	1	1	0	0		
1	0	0	0		0	0
1	0	1	1		1	
1	1	0	1			1
1	1	1	1			
			s(F)= ⟨	1/2	1/4	1/4	1/2 ⟩

Table 5.3

Faulty Signature of Function $F_{z/0} = F_1$

x	y	z	F_1	F_{10}^x	F_{10}^y	F_{10}^z
0	0	0	0	0	0	0
0	0	1	0	0	0	
0	1	0	1	1		1
0	1	1	1	1		
1	0	0	0		0	0
1	0	1	0		0	
1	1	0	1			1
1	1	1	1			
			$s(F_{z/0})$ = ⟨	1/2	1/2	0	1/2 ⟩

The above shows that a fault which may not be syndrome-testable could be syndrome signature-testable. Now the important question that arises is: Is the syndrome signature going to work for every case of single fault in a given circuit? The answer is obviously no, though it will work in most of the cases as our intensive experimentation with benchmark and other circuits conclusively shows.

Example: A circuit which cannot be tested using syndrome signature is shown below:

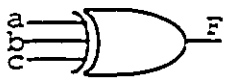


Fig. 5.2. A three input EXOR gate.

Table 5.4
Fault-Free Signature of Function $F = a \oplus b \oplus c$

a	b	c	F	F_i^a	F_i^b	F_i^c
0	0	0	0			
0	0	1	1			1
0	1	0	1		1	
0	1	1	0		0	0
1	0	0	1	1		
1	0	1	0	0		0
1	1	0	0	0	0	
1	1	1	1	1	1	1
			s(F)= ⟨	1/2	1/2	1/2	1/2 ⟩

Table 5.5
Faulty Signature of Function $F_{c/1} = F_1$

a	b	c	F_1	F_{1i}^a	F_{1i}^b	F_{1i}^c
0	0	0	1			
0	0	1	1			1
0	1	0	0		0	
0	1	1	0		0	0
1	0	0	0	0		
1	0	1	0	0		0
1	1	0	1	1	1	
1	1	1	1	1	1	1
			$s(F_{c/1})$ = ⟨	1/2	1/2	1/2	1/2 ⟩

By symmetry the test will not find *a/1* or *b/1*. Further, symmetry shows *a,b* or *c stuck-at 0* cannot be tested. Thus the test fails for any single input stuck-at fault.

However, we can also prove the result analytically based on our knowledge of switching functions and the definition of syndrome signature. The following results are pertinent.

Theorem 5.4: Two *n*-variable switching functions F_1 and F_2 comprised of the same number of minterms *m* and thus having the same syndrome $K = m/2^n$ must have at least one subsyndrome corresponding to the subfunction (F_b^i) obtained by setting the *ith* variable in both F_1 and F_2 equal to *b* (0) different, i.e.

$$s_i(F^i_{10}) \neq s_i(F^i_{20})$$

in order that F_1 and F_2 are syndrome signature-testable.

Proof: The theorem follows from the very notion of syndrome signature testability.

Corollary: If functions F_1 and F_2 are not syndrome signature-testable because

$$s_i(F^i_{10}) \neq s_i(F^i_{20})$$

then the implication is simply

$$N(\overline{x_i}F_1) \neq N(\overline{x_i}F_2)$$

where $N(\overline{x_i}F_1)$ and $N(\overline{x_i}F_2)$ denote respectively the number of minterms in the reduced functions $\overline{x_i}F_1$ and $\overline{x_i}F_2$.

Theorem 5.5: Two n-variable switching functions F_1 and F_2 comprised of the same number of minterms m and thus having the same syndrome $K = m/2^n$ and $F_1 = F_2$, may not have their syndrome signatures $s(F_1)$ and $s(F_2)$ different.

Proof: We will prove the theorem by considering the following individual situations:

F_1 and F_2 having the same number of minterms m and are different:

Case I: $m-1$ minterms identical and only minterm different.

Obviously, since the functions consist of the same number of minterms, their primary syndromes are identical, i.e. $s_0(F_1) = s_0(F_2)$. Assume that the minterm m_1 of F_1 is different from the minterm m_2 of F_2. If we now examine the n-tuples corresponding to minterms m_1 and m_2 in the table of combinations, obviously m_1 and m_2 must differ in <u>at least</u> one variable position, say x_i. Thus x_i is primed in one and unprimed in the other and hence

$$N(\overline{x_i}F_1) \neq N(\overline{x_i}F_2)$$

N denoting the number of minterms in the respective reduced functions, according to the corollary of Theorem 5.4. Thus

$$s_i(F^i_{10}) \neq s_i(F^i_{20})$$

and the syndrome signatures are different.

Case II: $m-2$ terms identical and two minterms different.

Assume m_1, m_2 of F_1 are different from m_1', m_2' of F_2. Then m_1, m_2 and m_1', m_2' must differ in <u>at least</u> two variable positions, say x_i and x_j. A possible assignment in the table of combinations for m_1, m_2 and m_1', m_2' is:

Table 5.6

F_1				F_2			
m_1		m_2		m_1'		m_2'	
x_i	x_j	x_i	x_j	x_i	x_j	x_i	x_j
0	0	1	1	1	0	0	1

In the above case, $N(\overline{x_i}F_1) = N(\overline{x_i}F_2)$ as well as $N(\overline{x_j}F_1) = N(\overline{x_j}F_2)$ and hence

$$s_i(F_{10}^j) = s_i(F_{20}^j) \text{ and } s_j(F_{10}^j) = s_j(F_{20}^j).$$

Thus two switching functions comprised of the same number of minterms and having the same primary syndrome might have their subsyndromes identical as well. This implies that their syndrome signatures are also identical.

Case III: Similarly, we may consider the case where F_1 and F_2 are identical in m-r minterms and differ in r minterms, and can conclude that their syndrome signatures for the case could possibly be identical.

Thus the theorem follows in general.

The following results are related and important in the context of syndrome signature testability.

Theorem 5.6: Consider two n-variable switching functions F_1 and F_2 and let F_1 and F_2 differ in M minterms.

Then the minimum number of variables in whose assignments in the truth table these must differ equals

$$\lceil \log_2 M \rceil$$

where $M = N_1 + N_2$, N_1 and N_2 being the terms in F_1 and F_2 respectively which are different $(N_1 = N_2 = M/2)$.

Corollary: Let F_1 and F_2 be two n-variable switching functions and let F_1 and F_2 have M of their minterms different. Let F_1 and F_2 need $\lceil \log_2 M \rceil$ variables with all their possible combinations in such a realization. If any variable x_i has equal number of *0's* and *1's* in $M/2$ minterms in F_1, any input line fault in x_i cannot change F_1 to F_2 having the same distribution of *0's* and *1's*. That is, the distribution of *0's* and *1's* must be different and as such

$$N(\overline{x_i}F_1) \neq N(\overline{x_i}F_2) \text{ and}$$

$$s_i(F^i_{10}) \neq s_i(F^i_{20})$$

and hence the subsyndromes must be different for at least one such x_i.

5.3 Multiple-Output Syndrome Signature

The concept of single-output syndrome signature can be readily extended to the multiple-output case. Evidently, the simplest strategy to extend the single-output syndrome signature to the multiple-output case is to generate a separate signature for each output. However, an excessive amount of hardware overhead makes this approach impractical.

Given an n-input m-output combinational circuit, a multiple-output syndrome signature is generated by EXORing

all the outputs to produce a new output which is then fed to a single-output syndrome signature generator. The signature can be generated by the vector

$$s_{mult}(F) = \langle s_0, s_1, \ldots, s_n \rangle .$$

Figure 5.3 shows the proposed implementation of a multiple-output syndrome signature.

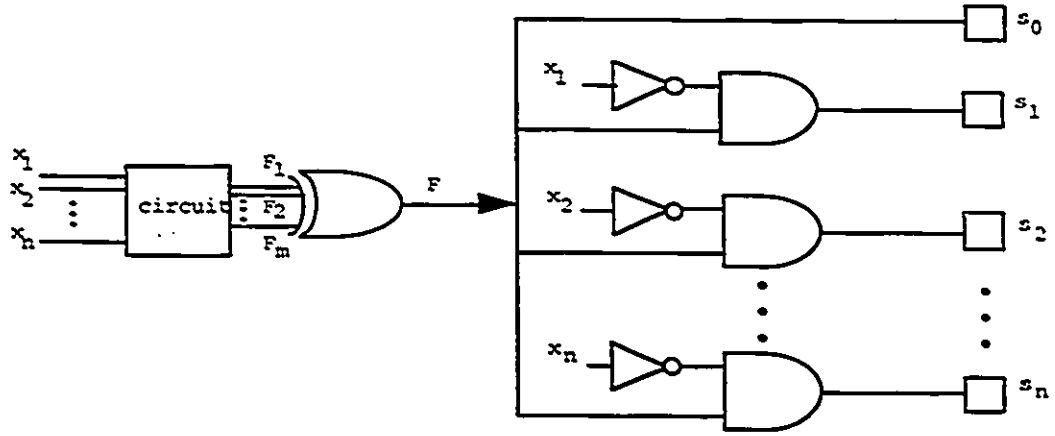


Fig. 5.3. Proposed implementation of a multiple-output syndrome signature.

Consider as illustration of multiple-output syndrome signature the following example.

Example: A 4-input 2-output function together with its multiple-output syndrome signature is given in Table 5.7.

The following properties of multiple-output syndrome signature are obvious.

Theorem 5.7: The proposed method of multiple-output syndrome signature generation makes the resultant signature *test-order independent*.

Theorem 5.8: The multiple-output syndrome signature like the single output case is a *nonuniform signature* since all possible signatures are not equally likely to occur.

Table 5.7

Multiple-Output Syndrome Signature

x_1	x_2	x_3	x_4	F_1	F_2	F	$F_0^{x_1}$	$F_0^{x_2}$	$F_0^{x_3}$	$F_0^{x_4}$
0	0	0	0	1	0	1	1	1	1	1
1	0	0	0	0	0	0		0	0	0
0	1	0	0	1	1	0	0		0	0
1	1	0	0	0	1	1			1	1
0	0	1	0	0	0	0	0	0		0
1	0	1	0	1	0	1		1		1
0	1	1	0	0	0	0	0			0
1	1	1	0	0	0	0				0
0	0	0	1	1	0	1	1	1	1	
1	0	0	1	1	1	0		0	0	
0	1	0	1	1	0	1	1		1	
1	1	0	1	0	0	0			0	
0	0	1	1	1	0	1	1	1		
1	0	1	1	0	0	0		0		
0	1	1	1	1	1	0	0			
1	1	1	1	0	1	1				
						$s_{mult}(F)=$	(7/16	1/2	1/2	1/2 3/8)

5.4. Some Thoughts on VLSI Implementation of the Proposed Signature Generator

The primary hardware in the implementation of the proposed signature generator is the syndrome register which can be readily implemented by using a counter. However, since the proposed signature generator, besides recording the primary syndrome needs to record, for an n variable function,

n other subsyndromes, $s_i(F_0^i)$, each corresponding to one of the variables x_i, in a parallel implementation of the signature generator we need to use a total of $n+1$ counters, the first one counting from 0 to 2^n-1, while the rest n counters counting from 0 to $2^{n-1}-1$ each. This apparently leads to an excessive overhead in incorporating the signature generator in a self-test environment. This is obviously a drawback if the implementation has to be strictly a parallel implementation.

One simple way to overcome this obvious shortcoming of the proposed generator is to use a serial implementation. In the serial implementation a single counter can be used to record both the primary syndrome and the other subsyndromes. The amount of storage is the same although RAM is smaller than flip-flop. The test strategy can be appropriately modified to take into account the nature of implementation [32].

Finally, the recent advances in VLSI technology may not prove to be a deterrent even in the parallel implementation of the signature generator. The inclusion of only binary counters consistent with the number of primary inputs may not be big hardware overhead afterall, while compared with the complexity of the original circuit that need to be tested using BIST.

5.5. Conclusions

In this chapter we have developed techniques for single-output and multiple-output signature generation for built-in self-testing using exhaustive test patterns by extending the syndrome concept originally developed by Savir. Though nonuniformity of the signatures is an inherent limitation, the property of test-order independence holding true makes

the application of exhaustive test patterns easier. We have also proved by an example in Figure 5.2 and as well as analytically that the syndrome signature does not work in all cases, though many "syndrome untestable" circuits could be "syndrome signature-testable".

However, in this context, it is relevant to refer to some subsequent works of Savir [32] relating to syndrome testing of "syndrome-untestable" circuits. Savir's idea in this case was tc perform multiple constrained syndrome tests on various portions of the circuit such that an overall full syndrome-test coverage could be achieved. A *constrained syndrome-test* is a syndrome-test performed on a subset of the input space, with other inputs being held at a constant value. These constraints are selected in a way that the output of the constrained function becomes unate with respect to the tested line and therefore syndrome-testable (since all unate functions are so testable). By taking resort to a prime implicant type covering procedure, Savir could minimize the number of syndrome test runs required to achieve complete coverage of all syndrome-untestable lines of the circuit. In our method utilizing an $(n+1)$-element syndrome signature for an n-input circuit, there is no scope of minimizing the number of subsyndromes to be recorded to test for different single faults of the circuit.

Chapter 6

IMPLEMENTATION RESULTS

To verify the proposed syndrome signature method, simulations were performrd on combinational circuits. The simulator consists of an input pattern generator, logic simulator, output compressor, and fault generator(and is executed on PC 486). All possible single stuck-at faults were introduced on the primary inputs lines(and on internal lines also for benchmark circuits discussed in Chapter 7) in the circuit, and the outputs were compressed by syndrome signature technique. Fault-free signatures were compared with the faulty signatures using syndrome counters.

In general, the dominating factors in the simulation were the number of input patterns and the number of faults that were injected.

A procedure describing an algorithm for testing a circuit for syndrome signature testability is presented here and the results of simulation for both single-output and multiple-output circuits are given in tabular form.

Tables 6.1 to 6.6 gives the results of simulation in detail for each circuit. Table 6.7 shows the number of input single stuck-at faults for each circuit; the number of faults missed by conventional syndrome; and the number of faults missed by syndrome signature. The percentage of missed faults in each circuit is also shown in Table 6.7. On the average, syndrome missed *45.8*% of faults, whereas syndrome signature missed none. Syndrome signature yields better fault coverage than conventional syndrome.

Procedure:

Step 1:

Given the number of inputs n

Calculate

R (the number of rows)= 2^n

C (the number of columns) = n

Step 2:

Given the faulty input and the fault in it, start the process of initialization:

X(n) = 0	Fault-free input array
XX(n) = 0	Faulty input array
Asum = 0	sum of fault-free outputs, whenever the output is 1. Final value of this will give the fault-free(actual) syndrome.
Fsum = 0	sum of faulty outputs, whenever the output is 1. Final value of this will give the faulty syndrome.

and also,

for m = 1, n

Asubs(m) = 0	sum of the fault-free outputs for each of the reduced functions, whenever the output is 1. Final value of this will give the actual subsyndromes.

Fsubs(m) = 0 sum of the faulty outputs for each of the reduced faulty functions, whenever the output is 1. Final value of this will give the faulty subsyndromes.

Step 3:

Main calculation loop:

Generate next fault-free and faulty input pattern(to start with, an all-0 combination will be the first fault-free input pattern and an all-0 combination with a given fault inserted will be the corresponding faulty input pattern).

Calculate actual and faulty functions, aF and fF respectively and perform the following assignments[1]:

If aF = 1, increase Asum by 1.
If fF = 1, increase Fsum by 1.

Now calculate fault-free and faulty subsyndromes as:

For m = 1, n
 If X(m) = 0 and aF = 1, increase Asubs(m) by 1.
 If X(m) = 0 and fF = 1, increase Fsubs(m) by 1.

---

[1]Division by 2^n on final Asum and Fsum and by 2^{n-1} on final Asubs(m) and Fsubs(m) are not performed here(according to the definitions of syndrome and subsyndromes), as we are doing only the comparison. Also, this avoids the round of errors while computing these values as the circuit size increases.

Repeat Step 3 until all the combinations(2^n) are done.

Step 4:

The decision to be made here is:
Is Asum is equal to Fsum?

If no, then the circuit is syndrome testable and
Goto Step 2;
else, try syndrome-signature testing and
Goto Step 2 until all faults of interest
are analyzed.

Example 6.1:

Consider the circuit realizing function $F(x_1, x_2, x_3) = x_1x_3 + x_2\overline{x_3}$.

This circuit has 3 inputs and therefore can have 6 input single stuck-at faults. The same circuit was discussed in Section 4.5 and was shown to be syndrome untestable for faults *x3/0* and *x3/1*. The circuit has been tested for syndrome-signature testability and was found to be syndrome-signature testable for these faults. The results of computer simulation for all the faults are presented below.

Table 6.1

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	4/2	yes		
$x_1/1$	4/6	yes		
$x_2/0$	4/2	yes		
$x_2/1$	4/6	yes		
$x_3/0$	4/4	no	4 1 1 2/4 2 0 2	yes
$x_3/1$	4/4	no	4 1 1 2/4 0 2 2	yes

Example 6.2:

Consider the circuit realizing function $F = x_1\overline{x_2} + \overline{x_1}x_3 + x_2\overline{x_3} + x_4x_5 + \overline{x_4}\,\overline{x_5}$.

This circuit has 5 inputs and therefore, can have 10 input single stuck-at faults. In Savir's paper[33] it is shown that this circuit is not syndrome testable and requires at least 1 control input (i.e., 1 extra pin) to make it syndrome testable. The same circuit when tested here for syndrome-signature testability, shows that it is syndrome-signature testable for all the faults mentioned above.

Table 6.2

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	28/28	no	28 14 14 14 14 14/ 28 14 12 12 14 14	yes
$x_1/1$	28/28	no	28 14 14 14 14 14/ 28 14 16 16 14 14	yes
$x_2/0$	28/28	no	28 14 14 14 14 14/ 28 12 14 12 14 14	yes
$x_2/1$	28/28	no	28 14 14 14 14 14/ 28 16 14 16 14 14	yes
$x_3/0$	28/28	no	28 14 14 14 14 14/ 28 12 12 14 14 14	yes
$x_3/1$	28/28	no	28 14 14 14 14 14/ 28 16 16 14 14 14	yes

- Continued...

Table 6.2 - Continued

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_4/0$	28/28	no	28 14 14 14 14 14/ 28 14 14 14 14 16	yes
$x_4/1$	28/28	no	28 14 14 14 14 14/ 28 14 14 14 14 12	yes
$x_5/0$	28/28	no	28 14 14 14 14 14/ 28 14 14 14 16 14	yes
$x_5/1$	28/28	no	28 14 14 14 14 14/ 28 14 14 14 12 14	yes

Example 6.3:

Consider the circuit realizing function $F = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + \overline{x_1}\,x_3\,x_4 + x_1\,\overline{x_2}\,\overline{x_3}\,x_4 + \overline{x_1}\,x_2\,\overline{x_3}\,x_4$.

This circuit has 4 inputs and therefore, can have 8 input single stuck-at faults. The results of computer simulation are listed below.

Table 6.3

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	6/10	yes		
$x_1/1$	6/2	yes		
$x_2/0$	6/8	yes		
$x_2/1$	6/4	yes		
$x_3/0$	6/8	yes		
$x_3/1$	6/4	yes		
$x_4/0$	6/2	yes		
$x_4/1$	6/10	yes		

Example 6.4:

Consider Data Selectors/MUX SN74150. It has 4 data select inputs and it selects one of sixteen data sources. The logic diagram and its truth table are given in Appendix B.
The data inputs are given by:

$E(m)$, where $m = 1, \cdots, 16$.

Let us consider one set of data inputs as:

1001000010001000

The circuit has been tested for these data inputs.

Table 6.4

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	12/12	no	12 6 5 5 5/12 6 4 6 6	yes
$x_1/1$	12/12	no	12 6 5 5 5/12 6 6 4 4	yes
$x_2/0$	12/10	yes		
$x_2/1$	12/14	yes		
$x_3/0$	12/10	yes		
$x_3/1$	12/14	yes		
$x_4/0$	12/10	yes		
$x_4/1$	12/14	yes		

Example 6.5:

Consider an 8-input Multiplexer SN74151. It has 3 data select inputs and it selects one bit of data from upto eight sources. The logic diagram and its truth table are given in Appendix B.

The data inputs are given by:

$D(m)$, where $m = 1, \cdots, 8$.

Let us consider one set of data inputs as:

10011010

The circuit has been tested for these data inputs.

Table 6.5

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	4/4	no	4 2 2 3/4 2 2 2	yes
$x_1/1$	4/4	no	4 2 2 3/4 2 2 4	yes
$x_2/0$	4/4	no	4 2 2 3/4 2 2 4	yes
$x_2/1$	4/4	no	4 2 2 3/4 2 2 2	yes
$x_3/0$	4/6	yes		
$x_3/1$	4/2	yes		

Example 6.6:

Consider a dual-output circuit described by the two logic functions given below as:

$$F_1 = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + \overline{x_1}\,x_2\,x_4 + x_1\,\overline{x_2}\,\overline{x_3}\,x_4 + \overline{x_1}\,\overline{x_2}\,x_3\,x_4$$

$$F_2 = x_2\,\overline{x_3}\,\overline{x_4} + x_1\,\overline{x_2}\,x_4 + x_1\,x_2\,x_3\,\overline{x_4} + \overline{x_1}\,x_2\,x_3\,x_4$$

$$F = F_1 \oplus F_2$$

This circuit has 4 inputs and can have 8 possible input single stuck-at faults.

Table 6.6

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	8/10	yes		
$x_1/1$	8/6	yes		
$x_2/0$	8/8	no	8 5 4 5 4/8 6 4 4 2	yes
$x_2/1$	8/8	no	8 5 4 5 4/8 4 4 6 6	yes
$x_3/0$	8/10	yes		
$x_3/1$	8/6	yes		
$x_4/0$	8/8	no	8 5 4 5 4/8 4 2 6 4	yes
$x_4/1$	8/8	no	8 5 4 5 4/8 6 6 4 4	yes

Summary of simulation results presented in Tables 6.1 through 6.6 is given next. CUT #s 6.1 through 6.5 are single-output circuits and CUT # 6.6 is a multiple-output circuit.

Table 6.7

CUT	Number of faults injected	Number of faults missed by syndrome	% of missed faults by syndrome	Number of faults missed by syndrome signature	% of missed faults by syndrome signature
6.1	6	2	33.3	none	0.0
6.2	10	10	100.0	none	0.0
6.3	8	none	0.0	none	0.0
6.4	8	2	25.0	none	0.0
6.5	6	4	66.6	none	0.0
6.6	8	4	50.0	none	0.0

Chapter 7

IMPLEMENTATION ON ISCAS 85 BENCHMARK CIRCUITS

In this chapter we give some experimental results on ISCAS 85 benchmark circuits[6]. Some of the benchmark circuits are not very large and can be handled quite easily whereas some others in the list are really large with respect to the number of primary inputs, internal lines and primary outputs. Since our method of testing using syndrome signature (in case the primary syndrome fails) is exhaustive, as far as input test patterns are concerned, storage and time become a problem if the circuit has more than *20* inputs. One obvious way out in such a situation is to partition the circuit into certain subcircuits depending on the dependence of a particular output on a proper subset of the inputs to make the problem tractable. We used this technique in the case of large benchmark circuits. Before we present our results of simulation experience on these circuits, we give some brief characterization of these combinational networks, as available from their netlist.

Since most of these benchmark circuits are large, considering even single fault, the possibilities are too many. As we have used microcomputers in our simulation, handling too many faults and bigger circuits was a primary concern.

In Tables 7.2 through 7.11 we present our results of experimentation on benchmark circuits. We have simulated only four out of ten benchmark circuits and the results obtained can be considered to be sufficient to show the implementation

of the proposed technique on large combinational logic circuits. We have considered both the primary input as well as the internal line faults. Since our major objective was only to see how the benchmark circuits relate to our concept of syndrome signature for detection of single faults, no attempt has been made to keep track of the CPU time.

It can be seen from the results, as given in the tables, that benchmark circuits "C17", "CKT" and a subcircuit(considering only that output which depends on less than 20 inputs) of "C432", are syndrome (a fault is syndrome signature testable also, if it is syndrome testable)testable for both input and internal line faults. The fourth circuit, "C880" can be partitioned into several subcircuits. Only two subcircuits of "C880" were considered for experimentation, one with *13* inputs and *9* outputs and another one with *10* inputs and *1* output. For some input and internal line faults, these subcircuits are syndrome testable while for some others they are syndrome-signature testable. There are some faults for which they are neither syndrome nor syndrome-signature testable(refer to Theorem 5.5 concerning nonvalidity of syndrome signature in some cases).

Tables 7.12 and 7.13 shows the number of faults missed by syndrome and their percentage as well as the number of faults missed by syndrome signature and their percentage, with respect to the total number of faults injected in each case of the single-output and multiple-output circuits. In the case of single-output circuits, syndrome signature has provided the same fault coverage as that of syndrome, but for multiple-output circuits syndrome signature has given a better fault coverage than syndrome.

Since in most of the multiple output benchmark circuits we used circuit partitioning, we never had to consider number

of inputs exceeding 18 which generates 2^{18} test patterns and could be handled without appreciable difficulty.

With today's computing power normally 2^{20} patterns could be generated in less than 1ms and as such testing a particular fault of interest should not be much of a problem, even if the syndrome signature(which uses exhaustive test patterns) needs to be used. Obviously, the signature would be working in most of the cases where the syndrome fails, though there may be cases where this signature might fail also(see counter example provided by the benchmark circuit).

Table 7.1

Characterization of Some of the Benchmark Circuits

Circuit No.	Lines from primary input gates	Lines from primary output gates	Lines from interior gate outputs
C17	5	2	4
CKT	6	1	6
C432	36	7	153
C880	60	26	357

Testing C17, CKT, C432 and C880 for input stuck-at faults:

Table 7.2

Results for "C17"

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	10/6	yes		
$x_1/1$	10/14	yes		
$x_2/0$	10/16	yes		
$x_2/1$	10/4	yes		
$x_3/0$	10/8	yes		
$x_3/1$	10/12	yes		
$x_4/0$	10/8	yes		
$x_4/1$	10/12	yes		
$x_5/0$	10/6	yes		
$x_5/1$	10/14	yes		

Table 7.3

Results for "CKT"

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	9/6	yes		
$x_1/1$	9/12	yes		
$x_2/0$	9/12	yes		
$x_2/1$	9/6	yes		
$x_3/0$	9/12	yes		
$x_3/1$	9/6	yes		
$x_4/0$	9/18	yes		
$x_4/1$	9/0	yes		
$x_5/0$	9/0	yes		
$x_5/1$	9/18	yes		
$x_6/0$	9/12	yes		
$x_6/1$	9/6	yes		

Table 7.4

Results for "C432"

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	301436/ 310184	yes		
$x_1/1$	301436/ 292688	yes		
$x_2/0$	301436/ 292688	yes		
$x_2/1$	301436/ 310184	yes		
$x_3/0$	301436/ 310184	yes		
$x_3/1$	301436/ 292688	yes		
$x_4/0$	301436/ 292688	yes		
$x_4/1$	301436/ 310184	yes		
$x_5/0$	301436/ 310184	yes		
$x_5/1$	301436/ 292688	yes		
$x_6/0$	301436/ 292688	yes		
$x_6/1$	301436/ 310184	yes		

- Continued...

Table 7.4 - Continued

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_7/0$	301436/ 310184	yes		
$x_7/1$	301436/ 292688	yes		
$x_8/0$	301436/ 292688	yes		
$x_8/1$	301436/ 310184	yes		
$x_9/0$	301436/ 310184	yes		
$x_9/1$	301436/ 292688	yes		
$x_{10}/0$	301436/ 292688	yes		
$x_{10}/1$	301436/ 310184	yes		
$x_{11}/0$	301436/ 310184	yes		
$x_{11}/1$	301436/ 292688	yes		
$x_{12}/0$	301436/ 292688	yes		
$x_{12}/1$	301436/ 310184	yes		

- Continued...

Table 7.4 - Continued

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_{13}/0$	301436/ 310184	yes		
$x_{13}/1$	301436/ 292688	yes		
$x_{14}/0$	301436/ 292688	yes		
$x_{14}/1$	301436/ 310184	yes		
$x_{15}/0$	301436/ 310184	yes		
$x_{15}/1$	301436/ 292688	yes		
$x_{16}/0$	301436/ 292688	yes		
$x_{16}/1$	301436/ 310184	yes		
$x_{17}/0$	301436/ 307997	yes		
$x_{17}/1$	301436/ 294875	yes		
$x_{18}/0$	301436/ 294875	yes		
$x_{18}/1$	301436/ 314558	yes		

Table 7.5

Results for "C880"

(subcircuit with 13 inputs and 9 outputs)

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	3200/3072	yes		
$x_1/1$	3200/3328	yes		
$x_2/0$	3200/3192	yes		
$x_2/1$	3200/3208	yes		
$x_3/0$	3200/3072	yes		
$x_3/1$	3200/3328	yes		
$x_4/0$	3200/3072	yes		
$x_4/1$	3200/3328	yes		
$x_5/0$	3200/3200	no	3200 1536 1596 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152/3200 1536 1536 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152	yes

- Continued...

Table 7.5 - Continued

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_5/1$	3200/3200	no	3200 1536 1596 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152/3200 1536 1656 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152	yes
$x_6/0$	3200/2976	yes		
$x_6/1$	3200/3424	yes		
$x_7/0$	3200/2976	yes		
$x_7/1$	3200/3424	yes		
$x_8/0$	3200/2976	yes		
$x_8/1$	3200/3424	yes		
$x_9/0$	3200/2976	yes		
$x_9/1$	3200/3424	yes		
$x_{10}/0$	3200/2752	yes		
$x_{10}/1$	3200/3648	yes		
$x_{11}/0$	3200/2976	yes		
$x_{11}/1$	3200/3424	yes		
$x_{12}/0$	3200/2304	yes		
$x_{12}/1$	3200/4096	yes		
$x_{13}/0$	3200/2304	yes		
$x_{13}/1$	3200/4096	yes		

Table 7.6

Results for "C880"

(subcircuit with 10 inputs and 1 output)

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$x_1/0$	512/512	no	512 256 256 256 256 256 256 256 256 256 256/512 256 256 256 256 256 256 256 256 256 256	no
$x_1/1$	512/512	no	512 256 256 256 256 256 256 256 256 256 256/512 256 256 256 256 256 256 256 256 256 256	no

Inputs x_2 to x_{10} are also neither syndrome nor syndrome-signature testable for both stuck-at-0 and stuck-at-1 faults.

Results on testing circuits C17, CKT, C432 and C880 for internal line stuck-at faults are given below.

Table 7.7

Results for "C17"(internal faults)

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$G_1/0$	10/14	yes		
$G_1/1$	10/6	yes		
$G_2/0$	10/8	yes		
$G_2/1$	10/8	yes		
$G_3/0$	10/0	yes		
$G_3/1$	10/16	yes		
$G_4/0$	10/14	yes		
$G_4/1$	10/6	yes		

Table 7.8

Results for "CKT"(internal faults)

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$G_1/0$	9/0	yes		
$G_1/1$	9/12	yes		
$G_2/0$	9/12	yes		
$G_2/1$	9/6	yes		
$G_3/0$	9/0	yes		
$G_3/1$	9/18	yes		
$G_4/0$	9/26	yes		
$G_4/1$	9/0	yes		
$G_5/0$	9/0	yes		
$G_5/1$	9/12	yes		
$G_6/0$	9/24	yes		
$G_6/1$	9/0	yes		

Table 7.9

Results for "C432"(internal faults)

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$G_1/0$	301436/ 292688	yes		
$G_1/1$	301436/ 310184	yes		
$G_2/0$	301436/ 292688	yes		
$G_2/1$	301436/ 310184	yes		
$G_3/0$	301436/ 292688	yes		
$G_3/1$	301436/ 310184	yes		
$G_4/0$	301436/ 292688	yes		
$G_4/1$	301436/ 310184	yes		
$G_5/0$	301436/ 292688	yes		
$G_5/1$	301436/ 310184	yes		
$G_6/0$	301436/ 292688	yes		
$G_6/1$	301436/ 310184	yes		

- Continued...

Table 7.9 - Continued

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$G_7/0$	301436/ 292688	yes		
$G_7/1$	301436/ 310184	yes		
$G_8/0$	301436/ 292688	yes		
$G_8/1$	301436/ 310184	yes		
$G_9/0$	301436/ 294875	yes		
$G_9/1$	301436/ 307997	yes		
$G_{10}/0$	301436/ 327680	yes		
$G_{10}/1$	301436/ 292688	yes		
$G_{11}/0$	301436/ 327680	yes		
$G_{11}/1$	301436/ 292688	yes		
$G_{12}/0$	301436/ 327680	yes		
$G_{12}/1$	301436/ 292688	yes		

- Continued...

Table 7.9 - Continued

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$G_{13}/0$	301436/ 327680	yes		
$G_{13}/1$	301436/ 292688	yes		
$G_{14}/0$	301436/ 327680	yes		
$G_{14}/1$	301436/ 292688	yes		
$G_{15}/0$	301436/ 327680	yes		
$G_{15}/1$	301436/ 292688	yes		
$G_{16}/0$	301436/ 327680	yes		
$G_{16}/1$	301436/ 292688	yes		
$G_{17}/0$	301436/ 327680	yes		
$G_{17}/1$	301436/ 292688	yes		
$G_{18}/0$	301436/ 327680	yes		
$G_{18}/1$	301436/ 294875	yes		

Table 7.10

Results for "C880"(internal faults)

(subcircuit with 13 inputs and 9 outputs)

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$G_1/0$	3200/5000	yes		
$G_1/1$	3200/3192	yes		
$G_2/0$	3200/4880	yes		
$G_2/1$	3200/3200	no	3200 1536 1596 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152/3200 1536 1536 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152	yes
$G_3/0$	3200/3200	no	3200 1536 1596 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152/3200 1536 1600 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152	yes

- Continued...

Table 7.10 - Continued

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$G_3/1$	3200/3200	no	3200 1536 1596 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152/3200 1536 1536 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152	yes
$G_4/0$	3200/3088	yes		
$G_4/1$	3200/5104	yes		
$G_5/0$	3200/3312	yes		
$G_5/1$	3200/4880	yes		
$G_6/0$	3200/3200	no	3200 1536 1596 1536 1536 1600 1488 1488 1488 1488 1376 1488 1152 1152/3200 1536 1604 1536 1536 1600 1376 1488 1488 1488 1488 1376 1152 1152	yes
$G_6/1$	3200/4992	yes		

Table 7.11

Results for "C880"(internal faults)

(subcircuit with 10 inputs and 1 output)

Single stuck-at faults	Fault-free syndrome/ Faulty syndrome	Syndrome testable yes/no	Syndrome signature Fault-free/Faulty	Syndrome-signature testable yes/no
$G_1/0$	512/512	no	512 256 256 256 256 256 256 256 256 256 256/512 256 256 256 256 256 256 256 256 256 256	no
$G_1/1$	512/512	no	512 256 256 256 256 256 256 256 256 256 256/512 256 256 256 256 256 256 256 256 256 256	no
$G_{28}/0$	512/256	yes		
$G_{28}/1$	512/768	yes		
$G_{29}/0$	512/768	yes		
$G_{29}/1$	512/256	yes		
$G_{30}/0$	512/768	yes		
$G_{30}/1$	512/256	yes		
$G_{31}/0$	512/768	yes		
$G_{31}/1$	512/0	yes		
$G_{32}/0$	512/768	yes		
$G_{32}/1$	512/0	yes		

Internal nodes G_2 to G_{27} are neither syndrome nor syndrome-signature testable for both stuck-at-0 and stuck-at-1 faults.

Summary of simulation results presented in Tables 7.2 through 7.11 are given below.

Table 7.12(a)

Single-Output Circuits

CUT	Total number of faults injected (input faults only)	Number of faults missed by syndrome	% of missed faults by syndrome	Number of faults missed by syndrome signature	% of missed faults by syndrome signature
CKT	12	none	0.0	none	0.0
C432	36	none	0.0	none	0.0
C880(10 inputs)	20	20	100.0	20	100.0

It can be seen next in Table 7.12(b) that with an increase in the total number of faults injected the percentage of missed faults by syndrome signature decreases.

Table 7.12(b)

Single-Output Circuits

CUT	Total number of faults injected (input and internal)	Number of faults missed by syndrome	% of missed faults by syndrome	Number of faults missed by syndrome signature	% of missed faults by syndrome signature
CKT	24	none	0.0	none	0.0
C432	72	none	0.0	none	0.0
C880(10 inputs)	84	54	64.3	54	64.3

Table 7.13

Multiple-Output Circuits

CUT	Total number of faults injected (input and internal)	Number of faults missed by syndrome	% of missed faults by syndrome	Number of faults missed by syndrome signature	% of missed faults by syndrome signature
C17	18	none	0.0	none	0.0
C880(13 inputs)	38	6	15.8	none	0.0

Chapter 8

CONCLUSIONS

We have developed techniques for single-output and multiple-output signature generation for built-in self-testing of VLSI circuits using exhaustive test patterns by extending the syndrome concept originally developed by Savir. The signature derived is an *(n+1)-element* vector consisting of the primary syndrome of the function as originally defined by Savir, and *n* other subsyndromes corresponding to the subfunctions obtained by setting each of the *n* input variables to *0* or *1*. Because the signature itself is simply the set of syndromes of *n+1* individual functions, we have examined a number of properties of syndrome which can be used in the analysis and manipulation of these signatures. The signature we have developed is itself a functional signature in the sense that it is independent of the particular implementation. Though nonuniformity of the signatures is an inherent limitation, the property of test-order independence holding true makes the application of exhaustive test patterns easier. The implementation of the proposed syndrome signature is shown by taking few examples(circuits with primary inputs less than 7 and ISCAS 85 benchmark circuits with primary inputs upto *18*) of which the results of computation are presented in Chapters 6 and 7. The signature developed is effective for both input and internal line fault detection as shown by the simulation results on ISCAS 85 benchmark circuits and hence is easily implementable. Some circuits(discussed in Chapter 6) which according to Savir needs extra inputs in order to make them syndrome testable, are shown to be syndrome signature testable. Also, the signature generation discussed for multiple-output circuits and shown in Tables 5.7, 6.6, 7.2 and 7.5 seems to work well

and preserves all the desirable properties of the single-output response analyzer. The masking effect of the proposed signature generator does not seem to be very high as only one case was reported(C880 with 10 inputs) out of *11* sample circuits, although an extensive simulation is needed for this statement to be valid, which is an important measure of a good compression technique.

Though this thesis is not primarily about the VLSI implementation of the proposed signature generator, some thought has been given in this direction too. The size of the compactor seems high as compared to the commonly used methods. The counters will require $O(n^2)$ flip-flops and $O(n^2)$ XORs and ANDs. For an *n-input* combinational circuit, this seems an excessive overhead in incorporating the signature generator in the BIST environment. This can be considered as a drawback of the proposed signature generator, if the implementation is strictly a parallel implementation. On the other hand, the serial implementation as suggested in Section 5.4 has the advantage of requiring only n flip-flops, but a disadvantage of requiring $n*2^n$ tests(thus increasing the test time). The number of memory cells required is the same as that in parallel implementation.

As the main drawback of exhaustive test patterns is their exponential growth with the number of inputs, such exhaustive testing is acceptable in cases(most of the benchmark circuits) where combinational circuits can be partitioned into subcircuits of about *20* inputs or less to keep the testing time under control.

Further research in this area needs to focus on relative effectiveness of the syndrome signature technique as a useful test compaction tool for fault detection through implementation of many more real world circuits where conventional syndromes fail. Also, implementation of such a

signature in VLSI could be a real worthwhile sequel to the whole exercise.

REFERENCES

[1] Agrawal, V.D., "An information theoretic approach to digital fault testing", *IEEE Trans. Computers*, Vol. C-30, August 1981, pp. 582-587.

[2] Agrawal, P. and Agrawal, V.D., "Probabilistic analysis of random test generation method for irredundant combinational logic networks", *IEEE Trans. Computers*, Vol. C-24, June 1975, pp. 691-695.

[3] Akers, S.B., "A parity bit signature for exhaustive testing", *IEEE Trans. Computer-Aided Design of VLSI Circuits and Systems*, Vol. 7, March 1988, pp. 333-338.

[4] Akers, S.B., "Test generation techniques", *IEEE Computer*, Vol. 13, March 1980, pp. 9-15.

[5] Bhattacharya, B.B. and Seth, S.C., "Design of parity testable combinational circuits", *IEEE Trans. Computers*, Vol. C-38, November 1989, pp. 1580-1584.

[6] Brglez, F. and Fujiwara, H., "A neutral netlist of 10 combinational circuits and a target translator in Fortran", *International Symposium on Circuits and Systems*, 1985.

[7] Carter, W.C., "Signature testing with guaranteed bounds for fault coverage", *Digest of Papers of Int. Test Conf.*, November 1982, pp. 75-82.

[8] Das, S.R., Jone, W.B., Nayak, A.R. and Choi, I., "On testing of sequential machines using circuit

decomposition and stochastic modeling", *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 25, March 1995, pp. 489-504.

[9] Das, S.R., Jone, W.B. and Wong, K.L., "Probabilistic modeling and fault analysis in sequential logic using computer simulation", *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 20, March/April 1990, pp. 490-498.

[10] Das, S.R., Srimani, P.K. and Datta, C.R., "On multiple fault analysis in combinational logic circuits by means of Boolean difference", *Proc. IEEE*, Vol. 64, September 1976, pp. 1447-1449.

[11] David, R. and Thevenod-Fosse, P., "Minimal detecting transition sequences : Application to random testing", *IEEE Trans. Computers*, Vol. C-29, June 1980, pp. 514-518.

[12] David, R., "Testing by feedback shift register", *IEEE Trans. Computers*, Vol. C-29, July 1980, pp. 668-673.

[13] Frowerk, R.A., "Signature analysis - A new digital field service method", *Hewlett-Packard Journal*, Vol. 28, May 1977, pp. 2-8.

[14] Hayes, J.P., "Transition count testing of combinational logic circuits", *IEEE Trans. Computers*, Vol. C-25, June 1976, pp. 613-620.

[15] Hsiao, T.-C. and Seth, S.C., "An analysis of the use of Rademacher-Walsh spectrum in compact testing", *IEEE Trans. Computers*, Vol. C-33, October 1984, pp. 934-937.

[16] Jone, W.B. and Das, S.R., "Space compression method for built-in self-testing of VLSI circuits", *International*

Journal of Computer Aided VLSI Design, Vol. 3, September 1991, pp. 309-322.

[17] Jone, W.B. and Das, S.R., "Multiple-output parity bit signature for exhaustive testing", *Journal of Electronic Testing : Theory and Applications*, Vol. 1, June 1990, pp. 175-178.

[18] Jone, W.B., Das, S.R. and Papachristou, C.A., "Pseudo-exhaustive testing of VLSI circuits through minimum vertex cut derivation of directed acyclic graphs", *Proc. of International Computer Symposium*, Taipei, Taiwan, ROC, December 1988, pp. 1091-1098 (Vol. II).

[19] Khabra, N.S. and Das, S.R., "Multiform partial symmetry and linearity", *IEEE Trans. Computers*, Vol. C-22, August 1973, p.804.

[20] Ku, C.T. and Masson, G.M., "The Boolean difference and multiple fault analysis", *IEEE Trans. Computers*, Vol. C-24, January 1975, pp. 62-71.

[21] Lala, P.K., Fault Tolerant and Fault Testable Hardware Design. London, U.K., Prentice-Hall, 1985.

[22] Li, Y.K. and Robinson, J.P., "Space compression method with output data modification", *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, Vol. 6, March 1987, pp. 290-294.

[23] Ma, H.T., Devadas, S., Newton, A.R. and Sangiovanni-Vincentelli, A., "Test generation for sequential circuits", *IEEE Trans. Computer Aided Design of VLSI Circuits and Systems*, Vol. 7, October 1988, pp. 1081-1093.

[24] Markowsky, G., "Syndrome-testability can be achieved by circuit modification", *IEEE Trans. Computers*, Vol. C-30, August 1981, pp. 604-606.

[25] McCluskey, E.J., Logic Design Principles with Emphasis on Testable Semicustom Circuits. Englewood Cliffs, NJ, Prentice-Hall, 1986.

[26] McCluskey, E.J., "Built-in self-test techniques", *IEEE Design and Test of Computers*, Vol. 2, April 1985, pp. 21-28.

[27] McCluskey, E.J. and Bozorgui-Nesbat, S., "Design for autonomous test", *IEEE Trans. Computers*, Vol. C-30, November 1981, pp. 866-875.

[28] Parker, K.P. and McCluskey, E.J., "Probabilistic treatment of general combinational networks", *IEEE Trans. Computers*, Vol. C-27, June 1978, pp. 668-670.

[29] Parker, K.P. and McCluskey, E.J., "Analysis of logic circuits with faults using input signal probabilities", *IEEE Trans. Computers*, Vol. C-27, May 1978, pp. 573-578.

[30] Savir, J. and Bardell, P.H., "Built-in self-test : milestones and challenges", *VLSI Design*, Vol. 1, March 1993, pp. 23-44.

[31] Savir, J. and McAnney, W.H., "On the masking probability with one's count and transition count", *Proc. of International Conference on Computer-Aided Design*, 1985, pp. 111-113.

[32] Savir, J., "Syndrome-testing of 'syndrome-untestable' combinational circuits", *IEEE Trans. Computers*, Vol. C-30, August 1981, pp. 606-608.

[33] Savir, J., "Syndrome-testable design of combinational circuits", *IEEE Trans. Computers*, Vol. C-29, June 1980, pp. 442-451; also *IEEE Trans. Computers*, Vol. C-29, November 1980, pp. 1012-1013 (correction).

[34] Shedletsky, J.J. and McCluskey, E.J., "The error latency of a fault in a sequential digital circuit", *IEEE Trans. Computers*, Vol. C-25, June 1976, pp. 655-659.

[35] Su, S.Y.H., Koren, I. and Malaiya, Y.K., "A continuous - parameter Markov model and detection procedure for intermittent faults", *IEEE Trans. Computers*, Vol. C-27, June 1978, pp. 567-570.

[36] Susskind, A.K., "Testing by verifying Walsh coefficients", *IEEE Trans. Computers*, Vol. C-32, February 1983, pp. 198-201.

[37] Yau, S.S. and Tang, Y.S., "An efficient algorithm for generating complete test sets for combinational logic nets", *IEEE Trans. Computers*, Vol. C-20, November 1971, pp. 1245-1251.

[38] Agrawal, V.D. and Seth, S.C., Test Generation for VLSI Chips. Washington, D.C., Computer Society Press, 1988.

[39] Bardell, P.H., McAnney, W.H. and Savir, J., Built-In Test for VLSI: Pseudorandom Techniques. New York, N.Y., John Wiley & Sons, 1987.

[40] Das, S.R., "Built-in self-testing of VLSI circuits", *IEEE Potentials*, Vol.10, October 1991, pp. 23-26.

[41] Armstrong, D.B., "On finding a nearly minimal set of fault detection tests for combinational logic nets",

IEEE Trans. Electron. Computers, Vol. EC-15, February 1966, pp. 66-73.

[42] Clegg, F.W., "Use of SPOOF's in the analysis of faulty logic networks", *IEEE Trans. Computers*, Vol. C-22, March 1973, pp. 229-234.

APPENDIX A

PROGRAM LISTINGS

A.1. Program Listing for Example 6.1

```
C*****************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 1
C
C  A CIRCUIT WITH 3 INPUTS AND 1 OUTPUT
C
C*****************************************************************

C     THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C   PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C   FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C   THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C   CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C   PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C   THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C   FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C   DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C   HAVE TO TRY SOME OTHER TESTING METHOD.

C   THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C   N              NUMBER OF INPUTS
C   R              TOTAL NUMBER OF INPUT COMBINATIONS
C   C              NUMBER OF COLUMNS IN INPUT ARRAY
C   X(N)         . ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C   XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C   I,J,M,K        COUNTERS
C   XN             FAULTY INPUT LINE
C   XFAULT         FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C   XX(N)          FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                  FAULT 'XFAULT' IN INPUT 'XN')
C   F              OUTPUT FUNCTION
C   PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C   CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C   ASUM           ACTUAL(FAULT-FREE) SYNDROME
C   FSUM           FAULTY SYNDROME
C   DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C   ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C   FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES.
C   DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C   AF             ACTUAL(FAULT-FREE) OUTPUT
C   FF             FAULTY OUTPUT
```

```
      PROGRAM TP3N

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(3),FSUBS(3),DSUBS(3),I,J,M,
     1        N,L
      LOGICAL X(3),XNEW(3),XX(3),XFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP3N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP3N.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N
      WRITE(6,200) N, R, C
200   FORMAT(' ', 'N=', I2, '  R=', I10, '  C=', I2)

      DO 5 L = 1,2*N
      READ (7,100) XN,XFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
```

```
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
              ENDIF
30          XX(J) = XNEW(J)
90          X(N) = XNEW(N)
            XX(XN) = XFAULT

          CALL CALF(N,X,AF)
          CALL CALF(N,XX,FF)
C
C         CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
          IF (AF) ASUM = ASUM+1
          IF (FF) FSUM = FSUM+1
C
C         CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C         CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C         (0 OR 1). HERE IT IS DONE BY SETTING IT 0.
C
          DO 40 M = 1,N
            IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40          IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000    CONTINUE
C
C
C       DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C       DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C       COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C       BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C       USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C       HAVE TO BE DIVIDED BY 2**(N-1).
C

        DSUM = ABS(ASUM - FSUM)

        IF(ASUM .NE. FSUM) THEN
            WRITE(6,260)ASUM,FSUM,DSUM
260         FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
            WRITE(6,265)
265         FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
        ELSE
          WRITE(6,270)ASUM,FSUM,DSUM
270       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

          WRITE(6,280)
280       FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
       1             ' SYNDROME SIGNATURE TESTING')
          DO 50 M = 1,N

            DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50          WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290         FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
        ENDIF
5       CONTINUE

        STOP
```

```
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N
      LOGICAL Y(N),F

      F = (Y(1) .AND. Y(3)) .OR. (Y(2) .AND. (.NOT. Y(3)))

      RETURN
      END
```

A.2. Program Listing for Example 6.2

```
C**********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 2
C
C  A CIRCUIT WITH 5 INPUTS AND 1 OUTPUT
C
C**********************************************************************

C     THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K        COUNTERS
C  XN             FAULTY INPUT LINE
C  XFAULT         FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C  XX(N)          FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                 FAULT 'XFAULT' IN INPUT 'XN')
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
```

```
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES.
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


       PROGRAM TP4N

       INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(5),FSUBS(5),DSUBS(5),I,J,M,
      1        N
       LOGICAL X(5),XNEW(5),XX(5),XFAULT,AF,FF

       OPEN(UNIT = 5)
       OPEN(UNIT = 7, FILE = 'TP4N.DAT', STATUS = 'OLD')
       OPEN(UNIT = 6, FILE = 'TP4N.OUT', STATUS = 'OLD')

       READ(7,100) N
       C = N
       R = 2**N
       WRITE(6,200) N, R, C
200    FORMAT(' ', 'N=', I2, '  R=', I10, '  C=', I2)

       DO 5 L = 1,2*N
       READ (7,100) XN,XFAULT
100    FORMAT (I2,L1)
C
C      STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C      IR = R/100
C
       WRITE(6,210)XN,XFAULT
210    FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
       DO 10 I = 1, N
         X(I) = .FALSE.
         XNEW(I) = .FALSE.
10        XX(I) = .FALSE.

       ASUM = 0
       FSUM = 0

       DO 20 M = 1,N
         ASUBS(M) = 0
         FSUBS(M) = 0
20     CONTINUE

C
C  MAIN CALCULATION LOOP
C
       DO 1000 I = 1,R

C       IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220      FORMAT(' ',I5,' % PROGRAM COMPLETE')

C        GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
```

```
          IF (I.EQ.1) GO TO 90
            DO 30 J = 1, N
              IF (J.EQ.1) THEN
                XNEW(J) = .NOT.X(J)
              ELSE
                IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                  X(J-1) = XNEW(J-1)
              ENDIF
30          XX(J) = XNEW(J)
90          X(N) = XNEW(N)
            XX(XN) = XFAULT

          CALL CALF(N,X,AF)
          CALL CALF(N,XX,FF)
C
C         CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
          IF (AF) ASUM = ASUM+1
          IF (FF) FSUM = FSUM+1
C
C         CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C         CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C         (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
          DO 40 M = 1,N
            IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40          IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000    CONTINUE
C
C
C       DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C       DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C       COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C       BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C       USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C       HAVE TO BE DIVIDED BY 2**(N-1).
C

        DSUM = ABS(ASUM - FSUM)

        IF(ASUM .NE. FSUM) THEN
            WRITE(6,260)ASUM,FSUM,DSUM
260         FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
            WRITE(6,265)
265         FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
        ELSE
          WRITE(6,270)ASUM,FSUM,DSUM
270       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

          WRITE(6,280)
280       FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
       1             ' SYNDROME SIGNATURE TESTING')
          DO 50 M = 1,N

            DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50          WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
```

```
290       FORMAT(' ','ACT. FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N
      LOGICAL Y(N),F

        F = (Y(1) .AND. (.NOT. Y(2))) .OR.
     1      ((.NOT. Y(1)) .AND. Y(3)) .OR.
     1      (Y(2) .AND. (.NOT. Y(3))) .OR.
     1      (Y(4) .AND. Y(5)) .OR.
     1      (.NOT. Y(4)) .AND. (.NOT. Y(5))

      RETURN
      END
```

A.3. Program Listing for Example 6.3

```
C*********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 3
C
C  A CIRCUIT WITH 4 INPUTS AND 1 OUTPUT
C
C*********************************************************************

C     THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K        COUNTERS
C  XN             FAULTY INPUT LINE
C  XFAULT         FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
```

```
C  XX(N)          FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                 FAULT 'XFAULT' IN INPUT 'XN')
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


      PROGRAM TP5N

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(4),FSUBS(4),DSUBS(4),I,J,L,
     1        M,N
      LOGICAL X(4),XNEW(4),XX(4),XFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP5N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP5N.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N

      WRITE(6,200) N, R, C
200   FORMAT(' ', 'N=', I2, '  R=', I10, '  C=', I2)

      DO 5 L =  1,2*N
      READ (7,100) XN,XFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE
```

```
C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        XX(J) = XNEW(J)
90        X(N) = XNEW(N)
          XX(XN) = XFAULT

        CALL CALF(N,X,AF)
        CALL CALF(N,XX,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1). HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
```

```
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES     : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1           ' ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N
      LOGICAL Y(N),F

        F = ((.NOT.Y(1)) .AND. (.NOT.Y(2)) .AND. (.NOT.Y(3))) .OR.
     1      ((.NOT.Y(1)) .AND. Y(3) .AND. Y(4)) .OR.
     1      (Y(1) .AND. (.NOT.Y(2)) .AND. (.NOT.Y(3)) .AND.
     1      Y(4)) .OR. ((.NOT.Y(1)) .AND. Y(2) .AND. (.NOT.Y(3))
     1      .AND. Y(4))

      RETURN
      END
```

A.4. Program Listing for Example 6.4

```
C*********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 4
C
C  DATA SELECTOR/MUX SN74150 (# OF PINS = 24, SELECTS ONE OF SIXTEEN
C    DATA SOURCES. IT HAS 4 PRIMARY INPUTS AND 1 OUTPUT. CIRCUIT
C    DESCRIPTION IS GIVEN IN APPENDIX B.
C
C*********************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
```

```
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N               NUMBER OF INPUTS
C  R               TOTAL NUMBER OF INPUT COMBINATIONS
C  C               NUMBER OF COLUMNS IN INPUT ARRAY
C  X(N)            ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)         NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L       COUNTERS
C  E(M)            DATA INPUT SET
C  XN              FAULTY INPUT LINE
C  XFAULT          FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C  XX(N)           FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                  FAULT 'XFAULT' IN INPUT 'XN')
C  F               OUTPUT FUNCTION
C  PF(K)           PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF            SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM            ACTUAL(FAULT-FREE) SYNDROME
C  FSUM            FAULTY SYNDROME
C  DSUM            DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)        ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)        ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)        ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF              ACTUAL(FAULT-FREE) OUTPUT
C  FF              FAULTY OUTPUT


      PROGRAM TP6N

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(4),FSUBS(4),DSUBS(4),I,J,M,
     1        L,N
      LOGICAL X(4),XNEW(4),XX(4),XFAULT,AF,FF,E(16)

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP6N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP6N.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N
      READ(7,110) (E(M), M = 1,16)
110   FORMAT(16L2)

      WRITE(6,200) N,R,C
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2)
      WRITE(6,205)
205   FORMAT(' ', 'THE DATA INPUTS ARE:')
      WRITE(6,206) (E(M), M = 1,16)
206   FORMAT(' ',(16L2))

      DO 5 L = 1,2*N
      READ(7,100) XN,XFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
```

```
      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        XX(J) = XNEW(J)
90        X(N) = XNEW(N)
          XX(XN) = XFAULT

        CALL CALF(N,X,E,AF)
        CALL CALF(N,XX,E,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1). HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
```

```
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1             ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,E,F)

      INTEGER N
      LOGICAL Y(N),F,E(16),A,B

        A = (E(1).AND.(.NOT.Y(1)).AND.(.NOT.Y(2)).AND.
     1       (.NOT.Y(3)).AND.(.NOT.Y(4)).OR.E(2).AND.(.NOT.
     1       Y(1)).AND.(.NOT.Y(2)).AND.(.NOT.Y(3)).AND.Y(4)
     1       .OR.E(3).AND.(.NOT.Y(1)).AND.(.NOT.Y(2)).AND.
     1       Y(3).AND.(.NOT.Y(4)).OR.E(4).AND.(.NOT.Y(1)).AND.
     1       (.NOT.Y(2)).AND.Y(3).AND.Y(4).OR. E(5).AND.(.NOT.
     1       Y(1)).AND.Y(2).AND.(.NOT.Y(3)).AND.(.NOT.Y(4))
     1       .OR.E(6).AND.(.NOT.Y(1)).AND.Y(2).AND.(.NOT.Y(3))
     1       .AND.Y(4).OR.E(7).AND.(.NOT.Y(1)).AND.Y(2).AND.
     1       Y(3).AND.(.NOT.Y(4)).OR.E(8).AND.(.NOT.Y(1))
     1       .AND.Y(2).AND.Y(3).AND.Y(4))
        B = (E(9).AND.Y(1)
     1       .AND.(.NOT.Y(2)).AND.(.NOT.Y(3)).AND.(.NOT.Y(4))
     1       .OR.E(10).AND.Y(1).AND.(.NOT.Y(2)).AND.(.NOT.
     1       Y(3)).AND.Y(4).OR.E(11).AND.Y(1).AND.(.NOT.
     1       Y(2)).AND. Y(3).AND.(.NOT.Y(4)).OR.E(12).AND.
```

```
     1       Y(1).AND.(.NOT.Y(2)).AND. Y(3).AND.Y(4).OR.
     1       E(13).AND.Y(1).AND.Y(2).AND.(.NOT.Y(3)).AND.
     1       (.NOT.Y(4)).OR. E(14).AND.Y(1).AND.Y(2).AND.
     1       (.NOT.Y(3)).AND.Y(4).OR.E(15).AND.Y(1).AND.Y(2)
     1       .AND.Y(3).AND.(.NOT.Y(4)).OR.E(16).AND.Y(1).AND.
     1       Y(2).AND.Y(3).AND.Y(4))

        F = .NOT.(A.OR.B)

      RETURN
      END
```

A.5. Program Listing for Example 6.5

```
C*********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 5
C
C  8-INPUT MULTIPLEXER SN74151 (# OF PINS = 16, SELECTS ONE BIT OF
C    DATA FROM UPTO EIGHT SOURCES. IT HAS 3 PRIMARY INPUTS AND 1 OUTPUT.
C    CIRCUIT DESCRIPTION IS GIVEN IN APPENDIX B.
C
C*********************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L      COUNTERS
C  E(M)           DATA INPUT SET
C  XN             FAULTY INPUT LINE
C  XFAULT         FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C  XX(N)          FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                 FAULT 'XFAULT' IN INPUT 'XN')
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
```

```
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


      PROGRAM TP7N

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(3),FSUBS(3),DSUBS(3),I,J,M,
     1        L,N
      LOGICAL X(3),XNEW(3),XX(3),XFAULT,AF,FF,E(8)

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP7N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP7N.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N
      READ(7,110) (E(M), M = 1,8)
110   FORMAT(16L2)

      WRITE(6,200) N,R,C
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2)
      WRITE(6,205)
205   FORMAT(' ', 'THE DATA INPUTS ARE:')
      WRITE(6,206) (E(M), M = 1,8)
206   FORMAT(' ',(16L2))

      DO 5 L = 1,2*N
      READ(7,100) XN,XFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
```

```
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220    FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        XX(J) = XNEW(J)
90        X(N) = XNEW(N)
          XX(XN) = XFAULT

        CALL CALF(N,X,E,AF)
        CALL CALF(N,XX,E,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)
```

```
        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1             ' SYNDROME SIGNATURE TESTING');
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,E,F)

      INTEGER N
      LOGICAL Y(N),F,E(8)

        F = E(1).AND.(.NOT.Y(1)).AND.(.NOT.Y(2)).AND.(.NOT.Y(3)).OR.
     1      E(2).AND.(.NOT.Y(1)).AND.(.NOT.Y(2)).AND.Y(3).OR.
     1      E(3).AND.(.NOT.Y(1)).AND.Y(2).AND.(.NOT.Y(3)).OR.
     1      E(4).AND.(.NOT.Y(1)).AND.Y(2).AND.Y(3).OR.
     1      E(5).AND.Y(1).AND.(.NOT.Y(2)).AND.(.NOT.Y(3)).OR.
     1      E(6).AND.Y(1).AND.(.NOT.Y(2)).AND.Y(3).OR.
     1      E(7).AND.Y(1).AND.Y(2).AND.(.NOT.Y(3)).OR.
     1      E(8).AND.Y(1).AND.Y(2).AND.Y(3)

      RETURN
      END
```

A.6. Program Listing for Example 6.6

```
C*************************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 6
C
C  THIS EXAMPLE IS AN IMPLEMENTATION OF MULTIPLE-OUTPUT SYNDROME
C    SIGNATURE. IT HAS 4 PRIMARY INPUTS AND 2 OUTPUTS.
C
C*************************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.
```

```
C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N               NUMBER OF INPUTS
C  R               TOTAL NUMBER OF INPUT COMBINATIONS
C  C               NUMBER OF COLUMNS IN INPUT ARRAY
C  X(N)            ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)         NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L       COUNTERS
C  XN              FAULTY INPUT LINE
C  XFAULT          FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C  XX(N)           FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                  FAULT 'XFAULT' IN INPUT 'XN')
C  F               OUTPUT FUNCTION
C  PF(K)           PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF            SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM            ACTUAL(FAULT-FREE) SYNDROME
C  FSUM            FAULTY SYNDROME
C  DSUM            DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)        ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)        ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)        ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF              ACTUAL(FAULT-FREE) OUTPUT
C  FF              FAULTY OUTPUT


      PROGRAM TP8N

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(4),FSUBS(4),DSUBS(4),I,J,M,
     1        L,N
      LOGICAL X(4),XNEW(4),XX(4),XFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP8N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP8N.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N

      WRITE(6,200) N,R,C
200   FORMAT('.', 'N =', I2, ' R =', I10, ' C =', I2)

      DO 5 L = 1,2*N
      READ(7,100) XN,XFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
```

```
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        XX(J) = XNEW(J)
90        X(N) = XNEW(N)
          XX(XN) = XFAULT

        CALL CALF(N,X,AF)
        CALL CALF(N,XX,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
```

```
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1            ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N
      LOGICAL Y(N),F,F1,F2

        F1 = (.NOT.Y(1)) .AND. (.NOT.Y(2)) .AND. (.NOT.Y(3)) .OR.
     1       (.NOT.Y(1)) .AND. Y(2) .AND. Y(4) .OR.
     1       Y(1) .AND. (.NOT.Y(2)) .AND. (.NOT.Y(3)) .AND. Y(4) .OR.
     1       (.NOT.Y(1)) .AND. (.NOT.Y(2)) .AND. Y(3) .AND. Y(4)

        F2 = Y(2) .AND. (.NOT.Y(3)) .AND. (.NOT.Y(4)) .OR.
     1       Y(1) .AND. (.NOT.Y(2)) .AND. Y(4) .OR.
     1       Y(1) .AND. Y(2) .AND. Y(3) .AND. (.NOT.Y(4)) .OR.
     1       (.NOT.Y(1)) .AND. Y(2) .AND. Y(3) .AND. Y(4)

        IF ((.NOT. F1) .AND.(.NOT. F2)) THEN
          F = .FALSE.
        ELSE
          IF (F1 .AND. F2) THEN
            F = .FALSE.
          ELSE
            F = .TRUE.
          ENDIF
        ENDIF

      RETURN
      END
```

A.7. Program Listing for "C17"(Table 7.2)

```
C*****************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 7
C
C  BENCHMARK CIRCUIT "C17". IT HAS 5 INPUTS AND 2 OUTPUTS. CIRCUIT
C    DESCRIPTION IS GIVEN IN APPENDIX B.
C
C*****************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L      COUNTERS
C  XN             FAULTY INPUT LINE
C  XFAULT         FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C  XX(N)          FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                 FAULT 'XFAULT' IN INPUT 'XN')
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


      PROGRAM TP9N

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(5),FSUBS(5),DSUBS(5),I,J,M,
     1        L,N
      LOGICAL X(5),XNEW(5),XX(5),XFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP9N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP9N.OUT', STATUS = 'OLD')
```

```
      READ(7,100) N
      C = N
      R = 2**N

      WRITE(6,200) N,R,C
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2)

      DO 5 L = 1,2*N
      READ(7,100) XN,XFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        XX(J) = XNEW(J)
90        X(N) = XNEW(N)
          XX(XN) = XFAULT

        CALL CALF(N,X,AF)
        CALL CALF(N,XX,FF)
```

```
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1            ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N
      LOGICAL Y(N),F,GATE10,GATE11,GATE16,GATE19,F1,F2
C
C  PRIMARY INPUTS 1,2,3,4 AND 5 CORRESPONDS TO GATE1, GATE2, GATE3,
```

```
C     GATE6 AND GATE7 RESPECTIVELY IN THE CIRCUIT DESCRIPTION OF "C17".
C     GATE10, GATE11, GATE16, GATE19 ARE INTERMEDIATE OUTPUTS AND F1
C     AND F2 CORRESPONDS TO PRIMARY OUTPUTS GATE22 AND GATE23
C
            GATE10 = .NOT.(Y(1) .AND. Y(3))
            GATE11 = .NOT.(Y(3) .AND. Y(4))
            GATE16 = .NOT.(Y(2) .AND. GATE11)
            GATE19 = .NOT.(GATE11 .AND. Y(5))

            F1 = .NOT.(GATE10 .AND. GATE16)
            F2 = .NOT.(GATE16 .AND. GATE19)

C  NOW TAKE THE EXCLUSIVE-OR SUM OF THE TWO OUTPUTS.

            IF ((.NOT.F1) .AND. (.NOT.F2)) THEN
              F = .FALSE.
            ELSE
              IF (F1 .AND. F2) THEN
                F = .FALSE.
              ELSE
                F = .TRUE.
              ENDIF
            ENDIF

      RETURN
      END
```

A.8. Program Listing for "CKT"(Table 7.3)

```
C*********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 8
C
C  BENCHMARK CIRCUIT "CKT". IT HAS 6 INPUTS AND 1 OUTPUT. CIRCUIT
C    DESCRIPTION IS GIVEN IN APPENDIX B.
C
C*********************************************************************

C     THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
```

```
C  X(N)          ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)       NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L     COUNTERS
C  XN            FAULTY INPUT LINE
C  XFAULT        FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C  XX(N)         FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                FAULT 'XFAULT' IN INPUT 'XN')
C  F             OUTPUT FUNCTION
C  PF(K)         PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF          SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM          ACTUAL(FAULT-FREE) SYNDROME
C  FSUM          FAULTY SYNDROME
C  DSUM          DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)      ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)      ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)      ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF            ACTUAL(FAULT-FREE) OUTPUT
C  FF            FAULTY OUTPUT


      PROGRAM TP10N

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(6),FSUBS(6),DSUBS(6),I,J,M,
     1        L,N
      LOGICAL X(6),XNEW(6),XX(6),XFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP10N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP10N.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N

      WRITE(6,200) N,R,C
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2)

      DO 5 L = 1,2*N
      READ(7,100) XN,XFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0
```

```
      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        XX(J) = XNEW(J)
90        X(N) = XNEW(N)
          XX(XN) = XFAULT

        CALL CALF(N,X,AF)
        CALL CALF(N,XX,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
```

```
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1             ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N
      LOGICAL Y(N),F,IOUT7,IOUT8,IOUT9,IOUT10,IOUT11,IOUT12
C
C     1,2,3,4,5 AND 6 ARE PRIMARY INPUTS AND IOUT7,IOUT8,IOUT9,IOUT10,
C     IOUT11 AND IOUT12 ARE INTERMEDIATE OUTPUTS AS DESCRIBED IN
C     THE CIRCUIT DESCRIPTION OF CKT. F CORRESPONDS TO THE PRIMARY
C     OUTPUT 13
C
        IOUT7 = .NOT. (Y(2) .AND. Y(3))
        IOUT8 = .NOT. (Y(1) .AND. IOUT7)
        IOUT9 = .NOT. (Y(4) .AND. IOUT7)
        IOUT10 = .NOT. (Y(5) .AND. IOUT7)
        IOUT11 = .NOT. (Y(6) .AND. IOUT8)
        IOUT12 = .NOT. (IOUT9 .AND. IOUT11)

        F = .NOT. (IOUT10 .OR. IOUT12)

      RETURN
      END
```

A.9. Program Listing for "C432"(Table 7.4)

```
C*********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 9
C
C  BENCHMARK CIRCUIT "C 432" WITH 18 INPUTS AND 1 OUTPUT (GAT 223)
C
C*********************************************************************
```

```
C     THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C   PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C   FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C   THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C   CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C   PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C   THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C   FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C   DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C   HAVE TO TRY SOME OTHER TESTING METHOD.

C   THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C   N               NUMBER OF INPUTS
C   R               TOTAL NUMBER OF INPUT COMBINATIONS
C   C               NUMBER OF COLUMNS IN INPUT ARRAY
C   X(N)            ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C   XNEW(N)         NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C   I,J,M,K           COUNTERS
C   XN              FAULTY INPUT LINE
C   XFAULT          FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C   XX(N)           FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                   FAULT 'XFAULT' IN INPUT 'XN'
C   F               OUTPUT FUNCTION
C   PF(K)           PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C   CALF            SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C   ASUM            ACTUAL(FAULT-FREE) SYNDROME
C   FSUM            FAULTY SYNDROME
C   DSUM            DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C   ASUBS(N)        ARRAY OF ACTUAL SUBSYNDROMES
C   FSUBS(N)        ARRAY OF FAULTY SUBSYNDROMES
C   DSUBS(N)        ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C   AF              ACTUAL(FAULT-FREE) OUTPUT
C   FF              FAULTY OUTPUT


      PROGRAM TP12N1

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(18),FSUBS(18),DSUBS(18),I,J,M,
     1        N
      LOGICAL X(18),XNEW(18),XX(18),XFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP12N1.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP12N1.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N
      WRITE(6,200) N, R, C
200   FORMAT(' ', 'N=', I2, '  R=', I10, '  C=', I2)

      DO 5 L = 1,2*N
      READ (7,100) XN,XFAULT
100   FORMAT (I2,L1)

C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
```

```
      IR = R/100

      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX

        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        XX(J) = XNEW(J)
90        X(N) = XNEW(N)
          XX(XN) = XFAULT

        CALL CALF(N,X,AF)
        CALL CALF(N,XX,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1
```

```
1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1           ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N
      LOGICAL Y(N),F,GAT118,GAT122,GAT126,GAT130,GAT134,GAT138,GAT142,
     1    GAT146,GAT150,GAT154,GAT159,GAT162,GAT165,GAT168,GAT171,
     1    GAT174,GAT177,GAT180,GAT199,GAT223

C
C  CALCULATION OF INTERMEDIATE OUTPUTS. INPUTS 1 TO 18 REFFER TO THE
C  PRIMARY INPUTS 1, 4, 11, 17, 24, 30, 37, 43, 50, 56, 63, 69, 76, 82,
C  89, 95, 102, 108 IN THE DESCRIPTION OF "C 432" GIVEN IN APPENDIX B.
C
      GAT118 = .NOT. Y(1)
      GAT122 = .NOT. Y(3)
      GAT126 = .NOT. Y(5)
      GAT130 = .NOT. Y(7)
      GAT134 = .NOT. Y(9)
      GAT138 = .NOT. Y(11)
      GAT142 = .NOT. Y(13)
```

```
      GAT146 = .NOT. Y(15)
      GAT150 = .NOT. Y(17)
      GAT154 = .NOT. (GAT118 .AND. Y(2))
      GAT159 = .NOT. (GAT122 .AND. Y(4))
      GAT162 = .NOT. (GAT126 .AND. Y(6))
      GAT165 = .NOT. (GAT130 .AND. Y(8))
      GAT168 = .NOT. (GAT134 .AND. Y(10))
      GAT171 = .NOT. (GAT138 .AND. Y(12))
      GAT174 = .NOT. (GAT142 .AND. Y(14))
      GAT177 = .NOT. (GAT146 .AND. Y(16))
      GAT180 = .NOT. (GAT150 .AND. Y(18))
      GAT199 = GAT154 .AND. GAT159 .AND. GAT162 .AND. GAT165 .AND.
     1         GAT168 .AND. GAT171 .AND. GAT174 .AND. GAT177 .AND.
     1         GAT180
      GAT223 = .NOT. GAT199

      F = GAT223

      RETURN
      END
```

A.10. Program Listing for "C880"(Table 7.5)

```
C*****************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 10
C
C  BENCHMARK CIRCUIT "C 880" (PARTITIONED) WITH 13 INPUTS  AND 9 OUTPUTS
C  (GAT388,GAT389,GAT390,GAT391,GAT418,GAT419,GAT420,GAT421,GAT422)
C*****************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K          COUNTERS
C  XN             FAULTY INPUT LINE
C  XFAULT         FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C  XX(N)          FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
```

```
C                FAULT 'XFAULT' IN INPUT 'XN')
C  F             OUTPUT FUNCTION
C  PF(K)         PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF          SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM          ACTUAL(FAULT-FREE) SYNDROME
C  FSUM          FAULTY SYNDROME
C  DSUM          DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)      ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)      ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)      ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF            ACTUAL(FAULT-FREE) OUTPUT
C  FF            FAULTY OUTPUT


      PROGRAM TP13N

      INTECER R,C,XN,ASUM,FSUM,DSUM,ASUBS(13),FSUBS(13),DSUBS(13),I,J,M,
     1        N
      LOGICAL X(13),XNEW(13),XX(13),XFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP13N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP13N.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N
      WRITE(6,200) N, R, C
200   FORMAT(' ', 'N=', I2, '  R=', I10, '  C=', I2)

      DO 5 L = 1,2*N
      READ (7,100) XN,XFAULT
100   FORMAT (I2,L1)

C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
      IR = R/100

      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
```

```
C
      DO 1000 I = 1,R
      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
220     FORMAT(' ',I5,' % PROGRAM COMPLETE')
C
C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        XX(J) = XNEW(J)
90        X(N) = XNEW(N)
          XX(XN) = XFAULT

        CALL CALF(N,X,AF)
        CALL CALF(N,XX,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
```

```
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1           ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N,U
      LOGICAL Y(N),F,PF(9),GAT269,GAT270,GAT273,GAT290,GAT291,GAT292,
     1    GAT293,GAT295,GAT296,GAT297,GAT342,GAT344,GAT351,GAT353,GAT354

C
C  CALCULATION OF INTERMEDIATE OUTPUTS. INPUTS 1 TO 13 REFFER TO THE
C  PRIMARY INPUTS 1, 8, 13, 17, 26, 29, 36, 42, 59, 75, 80, 85, 86
C  RESPECTIVELY IN THE DESCRIPTION OF "C 880" GIVEN IN APPENDIX B.
C
      GAT269 = .NOT. (Y(1) .AND. Y(2) .AND. Y(3) .AND. Y(4))
      GAT270 = .NOT. (Y(1) .AND. Y(5) .AND. Y(3) .AND. Y(4))
      GAT273 = Y(6) .AND. Y(7) .AND. Y(8)
      GAT290 = Y(6) .AND. Y(10) .AND. Y(8)
      GAT291 = Y(6) .AND. Y(7) .AND. Y(11)
      GAT292 = Y(6) .AND. Y(7) .AND. Y(8)
      GAT293 = Y(9) .AND. Y(10) .AND. Y(11)
      GAT295 = Y(9) .AND. Y(7) .AND. Y(11)
      GAT296 = Y(9) .AND. Y(7) .AND. Y(8)
      GAT297 = Y(12) .AND. Y(13)
      GAT342 = .NOT. GAT269
      GAT344 = GAT270 .OR. GAT273
      GAT351 = .NOT. GAT293
      GAT353 = .NOT. GAT295
      GAT354 = .NOT. GAT296

C
C  ALL THE 9 PRIMARY OUTPUTS ARE THE BUFFERED OUTPUTS OF
C  PF(1) TO PF(9) RESPECTIVELY
C
      PF(1) = GAT290
      PF(2) = GAT291
      PF(3) = GAT292
      PF(4) = GAT297
      PF(5) = GAT342
      PF(6) = GAT344
      PF(7) = GAT351
      PF(8) = GAT353
      PF(9) = GAT354

C  NOW TAKE THE EXCLUSIVE-OR SUM OF ALL THE PRIMARY OUTPUTS DESCRIBED
C  ABOVE AS PF(1) TO PF(9).
```

```
      DO 96 U = 1,8
            IF ((.NOT.PF(1)) .AND. (.NOT.PF(U+1))) THEN
              F = .FALSE.
            ELSE
              IF (PF(1) .AND. PF(U+1)) THEN
                F = .FALSE.
              ELSE
                F = .TRUE.
              ENDIF
            ENDIF
            PF(1) = F
96    CONTINUE

      RETURN
      END
```

A.11. Program Listing for "C880"(Table 7.6)

```
C*********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 11
C
C  BENCHMARK CIRCUIT "C 880" (PARTITIONED) WITH 10 INPUTS  AND 1 OUTPUT
C  (GAT767)
C*********************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K          COUNTERS
C  XN             FAULTY INPUT LINE
C  XFAULT         FAULT IN PRIMARY INPUT (STUCK-AT 0 OR 1)
C  XX(N)          FAULTY INPUT COMBINATION(OBTAINED BY INSERTING THE
C                 FAULT 'XFAULT' IN INPUT 'XN')
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
```

```
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


      PROGRAM TP14N

      INTEGER R,C,XN,ASUM,FSUM,DSUM,ASUBS(10),FSUBS(10),DSUBS(10),I,J,M,
     1        N
      LOGICAL X(10),XNEW(10),XX(10),XFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP14N.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP14N.OUT', STATUS = 'OLD')

      READ(7,100) N
      C = N
      R = 2**N
      WRITE(6,200) N, R, C
200   FORMAT(' ', 'N=', I2, '  R=', I10, '  C=', I2)

      DO 5 L = 1,2*N
      READ (7,100) XN,XFAULT
100   FORMAT (I2,L1)

C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
      IR = R/100

      WRITE(6,210)XN,XFAULT
210   FORMAT(/' ', 'INPUT ', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
        XNEW(I) = .FALSE.
10       XX(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R
      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
220     FORMAT(' ',I5,' % PROGRAM COMPLETE')
```

```
C
C       GENERATE NEW INPUT ARRAY XNEW AND FAULTY INPUT ARRAY XX
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30          XX(J) = XNEW(J)
90          X(N) = XNEW(N)
            XX(XN) = XFAULT

        CALL CALF(N,X,AF)
        CALL CALF(N,XX,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1             ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N
```

```
          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,F)

      INTEGER N
      LOGICAL Y(N),F,GAT301,GAT302,GAT303,GAT304,GAT305,GAT306,GAT307,
     1   GAT308,GAT357,GAT360,GAT363,GAT366,GAT404,GAT405,GAT406,GAT407,
     1   GAT408,GAT409,GAT425,GAT426,GAT460,GAT463,GAT498,GAT499,GAT500,
     1   GAT501,GAT530,GAT533,GAT550,GAT551,GAT552,GAT588,GAT660

C
C  CALCULATION OF INTERMEDIATE OUTPUTS. INPUTS 1 TO 10 REFFER TO THE
C  PRIMARY INPUTS 91, 96, 101, 106, 111, 116, 121, 126, 130, 135
C  RESPECTIVELY IN THE DESCRIPTION OF "C 880" GIVEN IN APPENDIX B.
C
      GAT301 = .NOT. (Y(1) .AND. Y(2))
      GAT302 = Y(1) .OR. Y(2)
      GAT303 = .NOT. (Y(3) .AND. Y(4))
      GAT304 = Y(3) .OR. Y(4)
      GAT305 = .NOT. (Y(5) .AND. Y(6))
      GAT306 = Y(5) .OR. Y(6)
      GAT307 = .NOT. (Y(7) .AND. Y(8))
      GAT308 = Y(7) .OR. Y(8)
      GAT357 = .NOT. (GAT301 .AND. GAT302)
      GAT360 = .NOT. (GAT303 .AND. GAT304)
      GAT363 = .NOT. (GAT305 .AND. GAT306)
      GAT366 = .NOT. (GAT307 .AND. GAT308)
      GAT404 = .NOT. GAT357
      GAT405 = .NOT. GAT360
      GAT406 = GAT357 .AND. GAT360
      GAT407 = .NOT. GAT363
      GAT408 = .NOT. GAT366
      GAT409 = GAT363 .AND. GAT366
      GAT425 = GAT404 .AND. GAT405
      GAT426 = GAT407 .AND. GAT408
      GAT460 = .NOT. (GAT406 .OR. GAT425)
      GAT463 = .NOT. (GAT409 .OR. GAT426)
      GAT498 = .NOT. (Y(9) .AND. GAT460)
      GAT499 = Y(9) .OR. GAT460
      GAT500 = .NOT. (GAT463 .AND. Y(10))
      GAT501 = GAT463 .OR. Y(10)
      GAT530 = .NOT. (GAT498 .AND. GAT499)
      GAT533 = .NOT. (GAT500 .AND. GAT501)
      GAT550 = .NOT. GAT530
      GAT551 = .NOT. GAT533
      GAT552 = GAT530 .AND. GAT533
      GAT588 = GAT550 .AND. GAT551
      GAT660 = .NOT. (GAT552 .OR. GAT588)
```

```
C
C  THE PRIMARY OUTPUT GAT767 IS THE BUFFERED OUTPUT OF GAT660
C
      F = GAT660

      RETURN
      END
```

A.12. Program Listing for "C17"(Table 7.7)

```
C*********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 1
C
C  BENCHMARK CIRCUIT "C17". IT HAS 5 INPUTS AND 2 OUTPUTS. CIRCUIT
C    DESCRIPTION IS GIVEN IN APPENDIX B.(TESTING FOR INTERNAL NODE
C    FAULTS)
C
C*********************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  IN             NUMBER OF INTERNAL NODES
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L      COUNTERS
C  GN             FAULTY INTERNAL NODE
C  GFAULT         FAULT IN INTERNAL NODE (STUCK-AT 0 OR 1)
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT
```

```
      PROGRAM TP5NN

      INTEGER R,C,GN,ASUM,FSUM,DSUM,ASUBS(5),FSUBS(5),DSUBS(5),I,J,M,
     1        L,N,IN
      LOGICAL X(5),XNEW(5),GFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP5NN.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP5NN.OUT', STATUS = 'OLD')

      READ(7,100) N
      READ(7,100) IN
      C = N
      R = 2**N

      WRITE(6,200) N,R,C,IN
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2, ' INTERNAL NODES
     1      =', I3)

      DO 5 L = 1,2*IN
      READ(7,100) GN,GFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
      WRITE(6,210)GN,GFAULT
210   FORMAT(/' ', 'INTERNAL NODE G', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
10      XNEW(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
```

```
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        CONTINUE
90        X(N) = XNEW(N)

        CALL CALF(N,X,0,GFAULT,AF)
        CALL CALF(N,X,GN,GFAULT,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1           ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
```

```
      END


      SUBROUTINE CALF(N,Y,GN,GFAULT,F)

      INTEGER N,GN
      LOGICAL Y(N),F,F1,F2,G(4),GFAULT
C
C     PRIMARY INPUTS 1,2,3,4 AND 5 CORRESPOND TO GATE1, GATE2, GATE3,
C     GATE6 AND GATE7 RESPECTIVELY. G(1) TO G(5) REFER TO GATE10,
C     GATE11, GATE16 AND GATE19 DESCRIBED AS INTERMEDIATE OUTPUTS IN
C     THE CIRCUIT DESCRIPTION OF "C17". F1 AND F2 CORRESPOND TO PRIMARY
C     OUTPUTS OF GATE22 AND GATE23.
C
        G(1) = .NOT. (Y(1) .AND. Y(3))
        G(2) = .NOT. (Y(3) .AND. Y(4))
        G(3) = .NOT. (Y(2) .AND. G(2))
        G(4) = .NOT. (G(2) .AND. Y(5))

        IF (GN .EQ. 0) THEN

          F1 = .NOT. (G(1) .AND. G(3))
          F2 = .NOT. (G(3) .AND. G(4))

        ELSEIF ((GN .EQ. 1) .OR.(GN .EQ. 3) .OR. (GN .EQ. 4)) THEN

          G(GN) = GFAULT
          F1 = .NOT. (G(1) .AND. G(3))
          F2 = .NOT. (G(3) .AND. G(4))

        ELSEIF (GN .EQ. 2) THEN

          G(GN) = GFAULT
          F1 = .NOT. (G(1) .AND. (.NOT. (Y(2) .AND. G(2))))
          F2 = .NOT. (.NOT. (Y(2) .AND. G(2)) .AND. (.NOT. (G(2) .AND.
     1          Y(5))))

        ENDIF

C  NOW TAKE THE EXCLUSIVE-OR SUM OF THE TWO OUTPUTS.

        IF ((.NOT.F1) .AND. (.NOT.F2)) THEN
          F = .FALSE.
        ELSE
          IF (F1 .AND. F2) THEN
            F = .FALSE.
          ELSE
            F = .TRUE.
          ENDIF
        ENDIF

        RETURN
        END
```

A.13. Program Listing for "CKT"(Table 7.8)

```
C*****************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 2
C
C  BENCHMARK CIRCUIT "CKT". IT HAS 6 INPUTS AND 1 OUTPUT. CIRCUIT
C    DESCRIPTION IS GIVEN IN APPENDIX B.(TESTING FOR INTERNAL NODE
C    FAULTS)
C
C*****************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  IN             NUMBER OF INTERNAL NODES
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L      COUNTERS
C  GN             FAULTY INTERNAL NODE
C  GFAULT         FAULT IN INTERNAL NODE (STUCK-AT 0 OR 1)
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


      PROGRAM TP4NN

      INTEGER R,C,GN,ASUM,FSUM,DSUM,ASUBS(6),FSUBS(6),DSUBS(6),I,J,M,
     1        L,N,IN
      LOGICAL X(6),XNEW(6),GFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP4NN.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP4NN.OUT', STATUS = 'OLD')
```

```
      READ(7,100) N
      READ(7,100) IN
      C = N
      R = 2**N

      WRITE(6,200) N,R,C,IN
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2, ' INTERNAL NODES
     1       =', I3)

      DO 5 L = 1,2*IN
      READ(7,100) GN,GFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
C     IR = R/100
C
      WRITE(6,210)GN,GFAULT
210   FORMAT(/' ', 'INTERNAL NODE G', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
10      XNEW(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

C      IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
C 220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        CONTINUE
90        X(N) = XNEW(N)

        CALL CALF(N,X,0,GFAULT,AF)
        CALL CALF(N,X,GN,GFAULT,FF)
```

```
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1            ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,GN,GFAULT,F)

      INTEGER N,GN
      LOGICAL Y(N),F,G(6),GFAULT
C
C     1,2,3,4,5 AND 6 ARE PRIMARY INPUTS AND G(1) TO G(6) ARE
```

```
C     INTERMEDIATE OUTPUTS DESCRIBED AS IOUT7,IOUT8,IOUT9,IOUT10,
C     IOUT11 AND IOUT12 IN THE CIRCUIT DESCRIPTION OF CKT. F CORRESPONDS
C     TO THE PRIMARY OUTPUT 13
C
        G(1) = .NOT. (Y(2) .AND. Y(3))
        G(2) = .NOT. (Y(1) .AND. G(1))
        G(3) = .NOT. (Y(4) .AND. G(1))
        G(4) = .NOT. (Y(5) .AND. G(1))
        G(5) = .NOT. (Y(6) .AND. G(2))
        G(6) = .NOT. (G(3) .AND. G(5))

        IF (GN .EQ. 0) THEN

          F = .NOT. (G(4) .OR. G(6))

        ELSEIF (GN .EQ. 1) THEN

          G(GN) = GFAULT
          F = .NOT. (.NOT. (Y(5) .AND. G(1)) .OR. (.NOT. (.NOT. (Y(4)
     1         .AND. G(1)) .AND. (.NOT. (Y(6) .AND. (.NOT. (Y(1) .AND.
     1        G(1)))))))

        ELSEIF (GN .EQ. 2) THEN

          G(GN) = GFAULT
          F = .NOT. (G(4) .OR. (.NOT. (G(3) .AND. (.NOT. (Y(6)
     1         .AND. G(2))))))

        ELSEIF (GN .EQ. 3) THEN

          G(GN) = GFAULT
          F = .NOT. (G(4) .OR. (.NOT. (G(3) .AND. G(5))))

        ELSEIF ((GN .EQ. 4) .OR. (GN .EQ. 6)) THEN

          G(GN) = GFAULT
          F = .NOT. (G(4) .OR. G(6))

        ELSEIF (GN .EQ. 5) THEN

          G(GN) = GFAULT
          F = .NOT. (G(4) .OR. (.NOT. (G(3) .AND. G(5))))

        ENDIF

        RETURN
        END
```

A.14. Program Listing for "C432"(Table 7.9)

```
C*****************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 3
C
C  BENCHMARK CIRCUIT "C432". IN PARTITIONED CIRCUIT ONLY ONE SUBCIRCUIT
C  HAS 18 INPUTS(WITHIN THE LIMIT OF 20 INPUTS WE HAVE PUT).IT HAS 1
C  OUTPUT. CIRCUIT DESCRIPTION IS GIVEN IN APPENDIX B.(TESTING FOR
C  INTERNAL NODE FAULTS)
C
C*****************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  IN             NUMBER OF INTERNAL NODES
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L      COUNTERS
C  GN             FAULTY INTERNAL NODE
C  GFAULT         FAULT IN INTERNAL NODE (STUCK-AT 0 OR 1)
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


      PROGRAM TP6NN

      INTEGER R,C,GN,ASUM,FSUM,DSUM,ASUBS(18),FSUBS(18),DSUBS(18),I,J,M,
     1        L,N,IN
      LOGICAL X(18),XNEW(18),GFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP6NN.DAT', STATUS = 'OLD')
```

```
      OPEN(UNIT = 6, FILE = 'TP6NN.OUT', STATUS = 'OLD')

      READ(7,100) N
      READ(7,100) IN
      C = N
      R = 2**N

      WRITE(6,200) N,R,C,IN
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2, ' INTERNAL NODES
     1      =', I3)

      DO 5 L = 1,2*IN
      READ(7,100) GN,GFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
      IR = R/100
C
      WRITE(6,210)GN,GFAULT
210   FORMAT(/' ', 'INTERNAL NODE G', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
10      XNEW(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

       IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
  220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        CONTINUE
90        X(N) = XNEW(N)

        CALL CALF(N,X,0,GFAULT,AF)
```

```
        CALL CALF(N,X,GN,GFAULT,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1            ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,GN,GFAULT,F)

      INTEGER N,GN
      LOGICAL Y(N),F,G(18),GFAULT
C
```

```
C  GAT118,GAT122,GAT126,GAT130,GAT134,GAT138,GAT142,GAT146,GAT150,
C  GAT154,GAT159,GAT162,GAT165,GAT168,GAT171,GAT174,GAT177,AND GAT180,
C  REFER TO THE INTERMEDIATE OUTPUTS AND ARE GIVEN HERE AS G(1) TO
C  G(18). INPUTS 1 TO 18 REFFER TO THE PRIMARY INPUTS 1, 4, 11, 17, 24,
C  30, 37, 43, 50, 56, 63, 69, 76, 82, 89, 95, 102, 108 IN THE
C  DESCRIPTION OF "C 432" GIVEN IN APPENDIX B.
C
      G(1) = .NOT. Y(1)
      G(2) = .NOT. Y(3)
      G(3) = .NOT. Y(5)
      G(4) = .NOT. Y(7)
      G(5) = .NOT. Y(9)
      G(6) = .NOT. Y(11)
      G(7) = .NOT. Y(13)
      G(8) = .NOT. Y(15)
      G(9) = .NOT. Y(17)
      G(10) = .NOT. (G(1) .AND. Y(2))
      G(11) = .NOT. (G(2) .AND. Y(4))
      G(12) = .NOT. (G(3) .AND. Y(6))
      G(13) = .NOT. (G(4) .AND. Y(8))
      G(14) = .NOT. (G(5) .AND. Y(10))
      G(15) = .NOT. (G(6) .AND. Y(12))
      G(16) = .NOT. (G(7) .AND. Y(14))
      G(17) = .NOT. (G(8) .AND. Y(16))
      G(18) = .NOT. (G(9) .AND. Y(18))


        IF (GN .EQ. 0) THEN

          F = .NOT.(G(10) .AND. G(11) .AND. G(12) .AND. G(13) .AND.
     1         G(14) .AND. G(15) .AND. G(16) .AND. G(17) .AND.
     1         G(18))

        ELSEIF (GN .EQ. 1) THEN

          G(GN) = GFAULT
          F = .NOT.(.NOT. (G(1) .AND. Y(2)) .AND. G(11) .AND. G(12)
     1         .AND. G(13) .AND. G(14) .AND. G(15) .AND. G(16) .AND.
     1        G(17) .AND. G(18))

        ELSEIF (GN .EQ. 2) THEN

          G(GN) = GFAULT
          F = .NOT.(G(10) .AND. (.NOT. (G(2) .AND. Y(4))) .AND. G(12)
     1         .AND. G(13) .AND. G(14) .AND. G(15) .AND. G(16) .AND.
     1        G(17) .AND. G(18))

        ELSEIF (GN .EQ. 3) THEN

          G(GN) = GFAULT
          F = .NOT.(G(10) .AND. G(11) .AND. (.NOT. (G(3) .AND. Y(6)))
     1         .AND. G(13) .AND. G(14) .AND. G(15) .AND. G(16) .AND.
     1        G(17) .AND. G(18))

        ELSEIF (GN .EQ. 4) THEN

          G(GN) = GFAULT
          F = .NOT.(G(10) .AND. G(11) .AND. G(12) .AND. (.NOT. (G(4)
```

```
1         .AND. Y(8))) .AND. G(14) .AND. G(15) .AND. G(16) .AND.
1         G(17) .AND. G(18))


    ELSEIF (GN .EQ. 5) THEN

      G(GN) = GFAULT
      F = .NOT.(G(10) .AND. G(11) .AND. G(12) .AND. G(13) .AND.
1         (.NOT. (G(5) .AND. Y(10))) .AND. G(15) .AND. G(16)
1         .AND. G(17) .AND. G(18))

    ELSEIF (GN .EQ. 6) THEN

      G(GN) = GFAULT
      F = .NOT.(G(10) .AND. G(11) .AND. G(12) .AND. G(13) .AND.
1         G(14) .AND. (.NOT. (G(6) .AND. Y(12))) .AND. G(16)
1         .AND. G(17) .AND. G(18))

    ELSEIF (GN .EQ. 7) THEN

      G(GN) = GFAULT
      F = .NOT.(G(10) .AND. G(11) .AND. G(12) .AND. G(13) .AND.
1         G(14) .AND. G(15) .AND. (.NOT. (G(7) .AND. Y(14)))
1         .AND. G(17) .AND. G(18))

    ELSEIF (GN .EQ. 8) THEN

      G(GN) = GFAULT
      F = .NOT.(G(10) .AND. G(11) .AND. G(12) .AND. G(13) .AND.
1         G(14) .AND. G(15) .AND. G(16) .AND. (.NOT. (G(8) .AND.
1         Y(16))) .AND. G(18))

    ELSEIF (GN .EQ. 9) THEN

      G(GN) = GFAULT
      F = .NOT.(G(10) .AND. G(11) .AND. G(12) .AND. G(13) .AND.
1         G(14) .AND. G(15) .AND. G(16)
1         .AND. G(17) .AND. (.NOT. (G(9) .AND. Y(18))))

    ELSEIF ((GN .EQ. 10) .OR. (GN .EQ. 11) .OR. (GN .EQ. 12) .OR.
1          (GN .EQ. 13) .OR. (GN .EQ. 14) .OR. (GN .EQ. 15) .OR.
1          (GN .EQ. 16) .OR. (GN .EQ. 17) .OR. (GN .EQ. 18)) THEN

      G(GN) = GFAULT
      F = .NOT.(G(10) .AND. G(11) .AND. G(12) .AND. G(13) .AND.
1         G(14) .AND. G(15) .AND. G(16) .AND. G(17) .AND. G(18))

  ENDIF

  RETURN
  END
```

A.15. Program Listing for "C880"(Table 7.10)

```
C*****************************************************************

C  NITA GOEL
C
C  EXAMPLE # 4
C
C  BENCHMARK CIRCUIT "C880". A SUBCIRCUIT THAT HAS 13 INPUTS AND 9
C  OUTPUTS IS CONSIDERED. CIRCUIT DESCRIPTION IS GIVEN IN APPENDIX B.
C  (TESTING FOR INTERNAL NODE FAULTS)
C
C*****************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  IN             NUMBER OF INTERNAL NODES
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L      COUNTERS
C  GN             FAULTY INTERNAL NODE
C  GFAULT         FAULT IN INTERNAL NODE (STUCK-AT 0 OR 1)
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


      PROGRAM TP7NN

      INTEGER R,C,GN,ASUM,FSUM,DSUM,ASUBS(13),FSUBS(13),DSUBS(13),I,J,M,
     1        L,N,IN
      LOGICAL X(13),XNEW(13),GFAULT,AF,FF

      OPEN(UNIT = 5)
```

```
      OPEN(UNIT = 7, FILE = 'T97NN.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'T97NN.OUT', STATUS = 'OLD')

      READ(7,100) N
      READ(7,100) IN
      C = N
      R = 2**N

      WRITE(6,200) N,R,C,IN
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2, ' INTERNAL NODES
     1      =', I3)

      DO 5 L = 1,2*IN
      READ(7,100) GN,GFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
      IR = R/100
C
      WRITE(6,210)GN,GFAULT
210   FORMAT(/' ', 'INTERNAL NODE G', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
10      XNEW(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

       IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
  220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        CONTINUE
90        X(N) = XNEW(N)
```

```
        CALL CALF(N,X,0,GFAULT,AF)
        CALL CALF(N,X,GN,GFAULT,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1             ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,GN,GFAULT,F)

      INTEGER N,GN,U
      LOGICAL Y(N),F,G(6),GFAULT, PF(9)
```

```
C
C  CALCULATION OF INTERMEDIATE OUTPUTS. INPUTS 1 TO 13 REFFER TO THE
C  PRIMARY INPUTS 1, 8, 13, 17, 26, 29, 36, 42, 59, 75, 80, 85, 86
C  RESPECTIVELY IN THE DESCRIPTION OF "C 880" GIVEN IN APPENDIX B AND
C  G(1) TO G(6) REFER TO THE INTERNAL OUTPUTS GAT269,GAT270,GAT273,
C  GAT293,GAT295,GAT296 RESPECTIVELY.
C
      G(1) = .NOT. (Y(1) .AND. Y(2) .AND. Y(3) .AND. Y(4))
      G(2) = .NOT. (Y(1) .AND. Y(5) .AND. Y(3) .AND. Y(4))
      G(3) = Y(6) .AND. Y(7) .AND. Y(8)
      G(4) = Y(9) .AND. Y(10) .AND. Y(11)
      G(5) = Y(9) .AND. Y(7) .AND. Y(11)
      G(6) = Y(9) .AND. Y(7) .AND. Y(8)

C
C  PF(1) TO PF(9) REFER TO THE PRIMARY OUTPUTS GAT388,GAT389,GAT390,
C  GAT391,GAT418,GAT419,GAT420,GAT421,GAT422 RESPECTIVELY WHICH ARE THE
C  BUFFERED OUTPUTS OF GAT290,GAT291,GAT292,GAT297,GAT342,GAT344,GAT351,
C  GAT353,GAT354.
C

      IF (GN .EQ. 0) THEN

          PF(1) = Y(6) .AND. Y(10) .AND. Y(8)
          PF(2) = Y(6) .AND. Y(7) .AND. Y(11)
          PF(3) = Y(6) .AND. Y(7) .AND. Y(8)
          PF(4) = Y(12) .AND. Y(13)
          PF(5) = .NOT. G(1)
          PF(6) = G(2) .OR. G(3)
          PF(7) = .NOT. G(4)
          PF(8) = .NOT. G(5)
          PF(9) = .NOT. G(6)

      ELSEIF (GN.EQ.1 .OR. GN.EQ.2 .OR. GN.EQ.3 .OR. GN.EQ.4 .OR.
     1        GN.EQ.5 .OR. GN.EQ.6) THEN

          G(GN) = GFAULT
          PF(1) = Y(6) .AND. Y(10) .AND. Y(8)
          PF(2) = Y(6) .AND. Y(7) .AND. Y(11)
          PF(3) = Y(6) .AND. Y(7) .AND. Y(8)
          PF(4) = Y(12) .AND. Y(13)
          PF(5) = .NOT. G(1)
          PF(6) = G(2) .OR. G(3)
          PF(7) = .NOT. G(4)
          PF(8) = .NOT. G(5)
          PF(9) = .NOT. G(6)

      ENDIF

C  NOW TAKE THE EXCLUSIVE-OR SUM OF ALL THE PRIMARY OUTPUTS DESCRIBED
C  ABOVE AS PF(1) TO PF(9).

      DO 96 U = 1,8
            IF ((.NOT.PF(1)) .AND. (.NOT.PF(U+1))) THEN
              F = .FALSE.
            ELSE
              IF (PF(1) .AND. PF(U+1)) THEN
```

```
                F = .FALSE.
              ELSE
                F = .TRUE.
              ENDIF
            ENDIF
            PF(1) = F
96    CONTINUE

      RETURN
      END
```

A.16. Program Listing for "C880"(Table 7.11)

```
C*********************************************************************
C
C  NITA GOEL
C
C  EXAMPLE # 5
C
C  BENCHMARK CIRCUIT "C880". A SUBCIRCUIT WHICH HAS I0 INPUTS(WITHIN
C  THE LIMIT OF 20 INPUTS WE HAVE PUT) AND 1 OUTPUT IS CONSIDERED.
C  CIRCUIT DESCRIPTION IS GIVEN IN APPENDIX B.(TESTING FOR INTERNAL NODE
C  FAULTS)
C
C*********************************************************************

C    THE FOLLOWING PROGRAM DESCRIBES HOW TO GENERATE EXHAUSTIVE TEST
C  PATTERNS GIVEN THE NUMBER OF INPUTS. IT THEN CALCULATES THE FAULT-
C  FREE SYNDROME AS WELL AS THE FAULTY SYNDROME OF THE FUNCTION. IF
C  THE FAULT-FREE SYNDROME IS DIFFERENT FROM THE FAULTY SYNDROME THE
C  CIRCUIT IS 'SYNDROME TESTABLE'; OTHERWISE, IT IS NOT AND IT THEN
C  PROCEEDS FOR THE 'SYNDROME SIGNATURE' TESTING.
C  THE PROGRAM NEXT CALCULATES THE 'SYNDROME SIGNATURE' OF THE FAULT-
C  FREE CIRCUIT AND THE FAULTY CIRCUIT. IF THE TWO SIGNATURES ARE
C  DIFFERENT, THE CIRCUIT IS 'SYNDROME SIGNATURE TESTABLE'AND IF NOT, WE
C  HAVE TO TRY SOME OTHER TESTING METHOD.

C  THE FOLLOWING NOTATIONS ARE USED IN DESDCRIBING THE PROGRAM:

C  N              NUMBER OF INPUTS
C  R              TOTAL NUMBER OF INPUT COMBINATIONS
C  C              NUMBER OF COLUMNS IN INPUT ARRAY
C  IN             NUMBER OF INTERNAL NODES
C  X(N)           ONE INPUT COMBINATION OUT OF 2**N COMBINATIONS
C  XNEW(N)        NEW INPUT ARRAY (TEMPORARY ARRAY ONLY)
C  I,J,M,K,L      COUNTERS
C  GN             FAULTY INTERNAL NODE
C  GFAULT         FAULT IN INTERNAL NODE (STUCK-AT 0 OR 1)
C  F              OUTPUT FUNCTION
C  PF(K)          PRIMARY OUTPUT ARRAY FOR MULTIPLE-OUTPUT CKTS
C  CALF           SUBROUTINE TO CALCULATE OUTPUT FUNCTION F
C  ASUM           ACTUAL(FAULT-FREE) SYNDROME
C  FSUM           FAULTY SYNDROME
C  DSUM           DIFFERENCE IN ACTUAL AND FAULTY SYNDROMES
C  ASUBS(N)       ARRAY OF ACTUAL SUBSYNDROMES
C  FSUBS(N)       ARRAY OF FAULTY SUBSYNDROMES
```

```
C  DSUBS(N)       ARRAY OF DIFFERENCE IN ACTUAL AND FAYLTY SUBSYNDROMES
C  AF             ACTUAL(FAULT-FREE) OUTPUT
C  FF             FAULTY OUTPUT


      PROGRAM TP8NN

      INTEGER R,C,GN,ASUM,FSUM,DSUM,ASUBS(10),FSUBS(10),DSUBS(10),I,J,M,
     1        L,N,IN
      LOGICAL X(10),XNEW(10),GFAULT,AF,FF

      OPEN(UNIT = 5)
      OPEN(UNIT = 7, FILE = 'TP8NN.DAT', STATUS = 'OLD')
      OPEN(UNIT = 6, FILE = 'TP8NN.OUT', STATUS = 'OLD')

      READ(7,100) N
      READ(7,100) IN
      C = N
      R = 2**N

      WRITE(6,200) N,R,C,IN
200   FORMAT(' ', 'N =', I2, ' R =', I10, ' C =', I2, ' INTERNAL NODES
     1      =', I3)

      DO 5 L = 1,2*IN
      READ(7,100) GN,GFAULT
100   FORMAT (I2,L1)
C
C     STATEMENT BELOW IS USEFUL FOR LARGE INPUT CIRCUITS.
      IR = R/100
C
      WRITE(6,210)GN,GFAULT
210   FORMAT(/' ', 'INTERNAL NODE G', I2,' IS STUCK AT ', L1)

C
C  INITIALIZE INPUT ARRAYS
C
      DO 10 I = 1, N
        X(I) = .FALSE.
10      XNEW(I) = .FALSE.

      ASUM = 0
      FSUM = 0

      DO 20 M = 1,N
        ASUBS(M) = 0
        FSUBS(M) = 0
20    CONTINUE

C
C  MAIN CALCULATION LOOP
C
      DO 1000 I = 1,R

       IF(((I/IR)*IR - I).EQ.0) WRITE(5,220) I/IR
  220     FORMAT(' ',I5,' % PROGRAM COMPLETE')

C       GENERATE NEW INPUT ARRAY XNEW
```

```
C
        IF (I.EQ.1) GO TO 90
          DO 30 J = 1, N
            IF (J.EQ.1) THEN
              XNEW(J) = .NOT.X(J)
            ELSE
              IF ((.NOT.XNEW(J-1)).AND.X(J-1)) XNEW(J)=.NOT.X(J)
                X(J-1) = XNEW(J-1)
            ENDIF
30        CONTINUE
90        X(N) = XNEW(N)

        CALL CALF(N,X,0,GFAULT,AF)
        CALL CALF(N,X,GN,GFAULT,FF)
C
C       CALCULATION OF FAULT-FREE AND FAULTY SYNDROMES
C
        IF (AF) ASUM = ASUM+1
        IF (FF) FSUM = FSUM+1
C
C       CALCULATION OF FAULTY-FREE AND FAULTY SUBSYNDROMES. SUBSYNDROMES
C       CAN BE OBTAINED BY SETTING THE ith VARIABLE IN F EQUAL TO b
C       (0 OR 1).  HERE IT IS DONE BY SETTING IT 0.
C
        DO 40 M = 1,N
          IF (.NOT. X(M) .AND. AF) ASUBS(M) = ASUBS(M) +1
40        IF (.NOT. X(M) .AND. FF) FSUBS(M) = FSUBS(M) +1

1000  CONTINUE
C
C
C     DIVISION OF FINAL ASUM AND FSUM BY 2**N (ACCORDING TO THE
C     DEFINITION OF SYNDROME) IS NOT NECESSARY HERE, AS WE ARE DOING
C     COMPARISON OF THE TWO SYNDROMES. FOR LARGER VALUES OF N DIVISON
C     BY 2**N WILL INTRODUCE  ROUND OF ERRORS. THE SIMILAR REASON IS
C     USED IN CACULATING THE SUBSYNDROMES IN WHICH ASUBS(N) AND FSUBS(N)
C     HAVE TO BE DIVIDED BY 2**(N-1).
C

      DSUM = ABS(ASUM - FSUM)

      IF(ASUM .NE. FSUM) THEN
          WRITE(6,260)ASUM,FSUM,DSUM
260       FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES  : ',3I12)
          WRITE(6,265)
265       FORMAT(' ','THE CIRCUIT IS SYNDROME TESTABLE')
      ELSE
        WRITE(6,270)ASUM,FSUM,DSUM
270     FORMAT(' ','ACT, FAUL AND DIFF SYNDROMES    : ',3I12)

        WRITE(6,280)
280     FORMAT(' ','THE CIRCUIT IS NOT SYNDROME TESTABLE,  TRY',
     1            ' SYNDROME SIGNATURE TESTING')
        DO 50 M = 1,N

          DSUBS(M) = ABS(ASUBS(M) - FSUBS(M))

50        WRITE(6,290)ASUBS(M),FSUBS(M),DSUBS(M)
```

```
290       FORMAT(' ','ACT, FAUL AND DIFF SUBSYNDROMES: ',3I12)
      ENDIF
5     CONTINUE

      STOP
      END


      SUBROUTINE CALF(N,Y,GN,GFAULT,F)

      INTEGER N,GN
      LOGICAL Y(N),F,G(32),GFAULT

C     CALCULATION OF INTERMEDIATE OUTPUTS. G(1) TO G(32) REFER TO THE
C     INTERMEDIATE OUTPUTS GAT301,GAT302,GAT303,GAT304,GAT305,GAT306,
C     GAT307,GAT308,GAT357,GAT360,GAT363,GAT366,GAT404,GAT405,GAT406,
C     GAT407,GAT408,GAT409,GAT425,GAT426,GAT460,GAT463,GAT498,GAT499,
C     GAT500,GAT501,GAT530,GAT533,GAT550,GAT551,GAT552,GAT588
C     RESPECTIVELY AND INPUTS 1 TO 10 REFFER TO THE PRIMARY INPUTS 91,
C     96, 101, 106, 111, 116, 121, 126, 130, 135 RESPECTIVELY IN THE
C     DESCRIPTION OF "C 880" GIVEN IN APPENDIX B.
C
      G(1) = .NOT. (Y(1) .AND. Y(2))
      G(2) = Y(1) .OR. Y(2)
      G(3) = .NOT. (Y(3) .AND. Y(4))
      G(4) = Y(3) .OR. Y(4)
      G(5) = .NOT. (Y(5) .AND. Y(6))
      G(6) = Y(5) .OR. Y(6)
      G(7) = .NOT. (Y(7) .AND. Y(8))
      G(8) = Y(7) .OR. Y(8)
      G(9) = .NOT. (G(1) .AND. G(2))
      G(10) = .NOT. (G(3) .AND. G(4))
      G(11) = .NOT. (G(5) .AND. G(6))
      G(12) = .NOT. (G(7) .AND. G(8))
      G(13) = .NOT. G(9)
      G(14) = .NOT. G(10)
      G(15) = G(9) .AND. G(10)
      G(16) = .NOT. G(11)
      G(17) = .NOT. G(12)
      G(18) = G(11) .AND. G(12)
      G(19) = G(13) .AND. G(14)
      G(20) = G(16) .AND. G(17)
      G(21) = .NOT. (G(15) .OR. G(19))
      G(22) = .NOT. (G(18) .OR. G(20))
      G(23) = .NOT. (Y(9) .AND. G(21))
      G(24) = Y(9) .OR. G(21)
      G(25) = .NOT. (G(22) .AND. Y(10))
      G(26) = G(22) .OR. Y(10)
      G(27) = .NOT. (G(23) .AND. G(24))
      G(28) = .NOT. (G(25) .AND. G(26))
      G(29) = .NOT. G(27)
      G(30) = .NOT. G(28)
      G(31) = G(27) .AND. G(28)
      G(32) = G(29) .AND. G(30)

        IF (GN .EQ. 0) THEN

          F = .NOT.(G(31) .OR. G(32))
```

```
      ELSEIF (GN.EQ.1 .OR. GN.EQ.2 .OR. GN.EQ.3 .OR. GN.EQ.4 .OR.
     1         GN.EQ.28) THEN

        G(GN) = GFAULT
        G(31) = .NOT.(.NOT.(Y(9).AND.(.NOT.((.NOT.(G(1).AND.G(2)).AND.
     1       (.NOT.(G(3).AND.G(4))))).OR.(.NOT.(.NOT.(G(1).AND.G(2)))
     1       .AND.(.NOT.(.NOT.(G(3).AND.G(4)))))))).AND.(Y(9).OR.
     1       (.NOT.((.NOT.(G(1).AND.G(2)).AND.(.NOT.(G(3).AND.G(4))))
     1       .OR.(.NOT.(.NOT.(G(1).AND.G(2))).AND.(.NOT.(.NOT.(G(3)
     1       .AND.G(4))))))).AND.G(28)
        G(32) = .NOT.(.NOT.(.NOT.(Y(9).AND.(.NOT.((.NOT.(G(1).AND.
     1       G(2)).AND.(.NOT.(G(3).AND.G(4))))).OR.(.NOT.(.NOT.(G(1)
     1       .AND.G(2))).AND.(.NOT.(.NOT.(G(3).AND.G(4))))))))).AND.
     1       (Y(9).OR.(.NOT.((.NOT.(G(1).AND.G(2)).AND.(.NOT.(G(3).AND.
     1       G(4))))).OR.(.NOT.(.NOT.(G(1).AND.G(2))).AND.(.NOT.(.NOT.
     1       (G(3).AND.G(4))))))))).AND.(.NOT.G(28))

        F = .NOT.(G(31).OR.G(32))

      ELSEIF (GN.EQ.5 .OR. GN.EQ.6 .OR. GN.EQ.7 .OR. GN.EQ.8 .OR.
     1         GN.EQ.27) THEN

        G(GN) = GFAULT
        G(31) = G(27).AND.(.NOT.(.NOT.(.NOT.((.NOT.(G(5).AND.G(6))
     1       .AND.(.NOT.(G(7).AND.G(8))))).OR.(.NOT.(.NOT.(G(5).AND.
     1       G(6))).AND.(.NOT.(.NOT.(G(7).AND.G(8)))))).AND.Y(10))
     1       .AND.(.NOT.((.NOT.(G(5).AND.G(6)).AND.(.NOT.(G(7).AND.
     1       G(8))))).OR.(.NOT.(.NOT.(G(5).AND.G(6))).AND.(.NOT.(.NOT.
     1       (G(7).AND.G(8)))))).OR.Y(10)))
        G(32) = .NOT.G(27).AND.(.NOT.(.NOT.(.NOT.(.NOT.((.NOT.(G(5)
     1       .AND.G(6)).AND.(.NOT.(G(7).AND.G(8))))).OR.(.NOT.(.NOT.
     1       (G(5).AND.G(6))).AND.(.NOT.(.NOT.(G(7).AND.G(8)))))).AND.
     1       Y(10)).AND.(.NOT.((.NOT.(G(5).AND.G(6)).AND.(.NOT.(G(7)
     1       .AND.G(8))))).OR.(.NOT.(.NOT.(G(5).AND.G(6))).AND.(.NOT.
     1       (.NOT.(G(7).AND.G(8)))))).OR.Y(10)))))

        F = .NOT.(G(31).OR.G(32))

      ELSEIF (GN.EQ.9 .OR. GN.EQ.10) THEN

        G(GN) = GFAULT
        G(31) = .NOT.(.NOT.(Y(9).AND.(.NOT.((G(9).AND.G(10)).OR.(.NOT.
     1       G(9).AND.(.NOT.G(10)))))).AND.(Y(9).OR.(.NOT.((G(9).AND.
     1       G(10)).OR.(.NOT.G(9).AND.(.NOT.G(10))))))).AND.G(28)
        G(32) = .NOT.(.NOT.(.NOT.(Y(9).AND.(.NOT.((G(9).AND.G(10)).OR.
     1       (.NOT.G(9).AND.(.NOT.G(10)))))).AND.(Y(9).OR.(.NOT.((G(9)
     1       .AND.G(10)).OR.(.NOT.G(9).AND.(.NOT.G(10))))))).AND.
     1       (.NOT.G(28))

        F = .NOT.(G(31).OR.G(32))

      ELSEIF (GN.EQ.11 .OR. GN.EQ.12) THEN

        G(GN) = GFAULT
        G(31) = G(27).AND.(.NOT.(.NOT.(.NOT.((G(11).AND.G(12)).OR.
     1       (.NOT.G(11).AND.(.NOT.G(12))))).AND.Y(10)).AND.(.NOT.
     1       ((G(11).AND.G(12)).OR.(.NOT.G(11).AND.(.NOT.G(12))))
```

```
1         .OR.Y(10))))
      G(32) = .NOT.G(27).AND.(.NOT.(.NOT.(.NOT.(.NOT.((G(11).AND.
1         G(12)).OR.(.NOT.G(11).AND.(.NOT.G(12)))).AND.Y(10)).AND.
1         (.NOT.((G(11).AND.G(12)).OR.(.NOT.G(11).AND.(.NOT.G(12))))
1         .OR.Y(10)))))

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.13 .OR. GN.EQ.14 .OR. GN.EQ.15) THEN

      G(GN) = GFAULT
      G(31) = .NOT.(.NOT.(Y(9).AND.(.NOT.(G(15).OR.(G(13).AND.
1         G(14))))).AND.(Y(9).OR.(.NOT.(G(15).OR.(G(13).AND.
1         G(14)))))).AND.G(28)
      G(32) = .NOT.(.NOT.(.NOT.(Y(9).AND.(.NOT.(G(15).OR.(G(13).AND.
1         G(14))))).AND.(Y(9).OR.(.NOT.(G(15).OR.(G(13).AND.
1         G(14)))))).AND.(.NOT.G(28))

      F =.NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.16 .OR. GN.EQ.17 .OR. GN.EQ.18) THEN

      G(GN) = GFAULT
      G(31) = G(27).AND.(.NOT.(.NOT.(.NOT.(G(18).OR.(G(16).AND.
1         G(17))).AND.Y(10)).AND.(.NOT.(G(18).OR.(G(16).AND.G(17)))
1        .OR.Y(10))))
      G(32) =.NOT.G(27).AND.(.NOT.(.NOT.(.NOT.(.NOT.(G(18).OR.(G(16)
1         .AND.G(17))).AND.Y(10)).AND.(.NOT.(G(18).OR.(G(16).AND.
1         G(17))).OR.Y(10)))))

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.19) THEN

      G(GN) = GFAULT
      G(31) = .NOT.(.NOT.(Y(9).AND.(.NOT.(G(15).OR.G(19)))))
1        .AND.(Y(9).OR.(.NOT.(G(15).OR.G(19))))).AND.G(28)
      G(32) = .NOT.(.NOT.(.NOT.(Y(9).AND.(.NOT.(G(15).OR.G(19))))
1        .AND.(Y(9).OR.(.NOT.(G(15).OR.G(19)))))).AND.(.NOT.G(28))

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.20) THEN

      G(GN) = GFAULT
      G(31) = G(27).AND.(.NOT.(.NOT.(.NOT.(G(18).OR.G(20)).AND.
1         Y(10)).AND.(.NOT.(G(18).OR.G(20)).OR.Y(10))))
      G(32) = .NOT.G(27).AND.(.NOT.(.NOT.(.NOT.(.NOT.(G(18).OR.
1         G(20)).AND.Y(10)).AND.(.NOT.(G(18).OR.G(20)).OR.Y(10)))))

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.21) THEN

      G(GN) = GFAULT
      G(31) = .NOT.(.NOT.(Y(9).AND.G(21)).AND.(Y(9).OR.G(21)))
1         .AND.G(28)
      G(32) = .NOT.(.NOT.(.NOT.(Y(9).AND.G(21)).AND.(Y(9).OR.
```

```
1         G(21))))).AND.(.NOT.G(28))

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.22) THEN

      G(GN) = GFAULT
      G(31) = G(27).AND.(.NOT.(.NOT.(G(22).AND.Y(10)).AND.(G(22)
1        .OR.Y(10))))
      G(32) = .NOT.G(27).AND.(.NOT.(.NOT.(.NOT.(G(22).AND.Y(10))
1      .AND.(G(22).OR.Y(10)))))

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.23 .OR. GN.EQ.24) THEN

      G(GN) = GFAULT
      G(31) = .NOT.(G(23).AND.G(24)).AND.G(28)
      G(32) = .NOT.(.NOT.(G(23).AND.G(24))).AND.(.NOT.G(28))

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.25 .OR. GN.EQ.26) THEN

      G(GN) = GFAULT
      G(31) = G(27).AND.(.NOT.(G(25).AND.G(26)))
      G(32) = .NOT.G(27).AND.(.NOT.(.NOT.(G(25).AND.G(26))))

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.29 .OR. GN.EQ.30) THEN

      G(GN) = GFAULT
      G(31) = G(27).AND.G(28)
      G(32) = G(29).AND.G(30)

      F = .NOT.(G(31).OR.G(32))

    ELSEIF (GN.EQ.31 .OR. GN.EQ.32) THEN

      G(GN) = GFAULT
      F = .NOT.(G(31).OR.G(32))

  ENDIF

  RETURN
  END
```

APPENDIX B

DATA SHEETS

```
$c17 iscas example (to test conversion program only)
$-----------------------------------------------
$
$
$  total number of lines in the netlist .............    17
$  simplistically reduced equivalent fault set size =    22
$        lines from primary input  gates .......     5
$        lines from primary output gates .......     2
$        lines from interior gate outputs ......     4
$        lines from **     3 ** fanout stems ...     6
$
$        avg_fanin  =  2.00,     max_fanin  =  2
$        avg_fanout =  2.00,     max_fanout =  2
$
$
$
$
$
1gat                                    $... primary input
2gat                                    $... primary input
3gat                                    $... primary input
6gat                                    $... primary input
7gat                                    $... primary input
                              $... primary input
$
$
22gat                                   $... primary output
23gat                                   $... primary output
                              $... primary output
$
$
$       Output  Type    Inputs...
$       ------  ----    ---------
      10gat   nand    1gat    3gat
      11gat   nand    3gat    6gat
      16gat   nand    2gat    11gat
      19gat   nand    11gat   7gat
      22gat   nand    10gat   16gat
      23gat   nand    16gat   19gat
```

```
$CKT iscas example (to test conversion program only)
$-----------------------------------------------
$
$
1
2
```

```
3
4
5
6
7 nand 2 3
8 nand 1 7
9 nand 4 7
10 nand 5 7
11 nand 6 8
12 nand 9 11
13 nor 10 12


$  combinational logic example "c432"
$-----------------------------------------------------
$
$
$  total number of lines in the netlist ..............   432
$  simplistically reduced equivalent fault set size =    524
$        lines from primary input   gates .......    36
$        lines from primary output  gates .......     7
$        lines from interior  gate outputs ......   153
$        lines from **    89 ** fanout stems ...   236
$
$        avg_fanin  =  2.10,     max_fanin  =  9
$        avg_fanout =  2.65,     max_fanout =  9
$
$
$
$
$
1 gat                                   $... primary input
4 gat                                   $... primary input
8 gat                                   $... primary input
11 gat                                  $... primary input
14 gat                                  $... primary input
17 gat                                  $... primary input
21 gat                                  $... primary input
24 gat                                  $... primary input
27 gat                                  $... primary input
30 gat                                  $... primary input
34 gat                                  $... primary input
37 gat                                  $... primary input
40 gat                                  $... primary input
43 gat                                  $... primary input
47 gat                                  $... primary input
50 gat                                  $... primary input
53 gat                                  $... primary input
56 gat                                  $... primary input
60 gat                                  $... primary input
63 gat                                  $... primary input
66 gat                                  $... primary input
69 gat                                  $... primary input
73 gat                                  $... primary input
76 gat                                  $... primary input
79 gat                                  $... primary input
82 gat                                  $... primary input
```

```
86 gat                                          $... primary input
89 gat                                          $... primary input
92 gat                                          $... primary input
95 gat                                          $... primary input
99 gat                                          $... primary input
102 gat                                         $... primary input
105 gat                                         $... primary input
108 gat                                         $... primary input
112 gat                                         $... primary input
115 gat                                         $... primary input
                                    $... primary input
$
$
223 gat                                         $... primary output
329 gat                                         $... primary output
370 gat                                         $... primary output
421 gat                                         $... primary output
430 gat                                         $... primary output
431 gat                                         $... primary output
432 gat                                         $... primary output
                                    $... primary output
$
$
$       Output  Type    Inputs...
$       ------  ----    ---------
      118 gat not     1 gat
      119 gat not     4 gat
      122 gat not     11 gat
      123 gat not     17 gat
      126 gat not     24 gat
      127 gat not     30 gat
      130 gat not     37 gat
      131 gat not     43 gat
      134 gat not     50 gat
      135 gat not     56 gat
      138 gat not     63 gat
      139 gat not     69 gat
      142 gat not     76 gat
      143 gat not     82 gat
      146 gat not     89 gat
      147 gat not     95 gat
      150 gat not     102 gat
      151 gat not     108 gat
      154 gat nand    118 gat 4 gat
      157 gat nor     8 gat   119 gat
      158 gat nor     14 gat  119 gat
      159 gat nand    122 gat 17 gat
      162 gat nand    126 gat 30 gat
      165 gat nand    130 gat 43 gat
      168 gat nand    134 gat 56 gat
      171 gat nand    138 gat 69 gat
      174 gat nand    142 gat 82 gat
      177 gat nand    146 gat 95 gat
      180 gat nand    150 gat 108 gat
      183 gat nor     21 gat  123 gat
      184 gat nor     27 gat  123 gat
      185 gat nor     34 gat  127 gat
      186 gat nor     40 gat  127 gat
```

```
187 gat nor     47 gat  131 gat
188 gat nor     53 gat  131 gat
189 gat nor     60 gat  135 gat
190 gat nor     66 gat  135 gat
191 gat nor     73 gat  139 gat
192 gat nor     79 gat  139 gat
193 gat nor     86 gat  143 gat
194 gat nor     92 gat  143 gat
195 gat nor     99 gat  147 gat
196 gat nor     105 gat 147 gat
197 gat nor     112 gat 151 gat
198 gat nor     115 gat 151 gat
199 gat and     154 gat 159 gat 162 gat 165 gat 168 gat 171 gat
                  174 gat 177 gat 180 gat
203 gat not     199 gat
213 gat not     199 gat
223 gat not     199 gat
224 gat xor     203 gat 154 gat
227 gat xor     203 gat 159 gat
230 gat xor     203 gat 162 gat
233 gat xor     203 gat 165 gat
236 gat xor     203 gat 168 gat
239 gat xor     203 gat 171 gat
242 gat nand    1 gat   213 gat
243 gat xor     203 gat 174 gat
246 gat nand    213 gat 11 gat
247 gat xor     203 gat 177 gat
250 gat nand    213 gat 24 gat
251 gat xor     203 gat 180 gat
254 gat nand    213 gat 37 gat
255 gat nand    213 gat 50 gat
256 gat nand    213 gat 63 gat
257 gat nand    213 gat 76 gat
258 gat nand    213 gat 89 gat
259 gat nand    213 gat 102 gat
260 gat nand    224 gat 157 gat
263 gat nand    224 gat 158 gat
264 gat nand    227 gat 183 gat
267 gat nand    230 gat 185 gat
270 gat nand    233 gat 187 gat
273 gat nand    236 gat 189 gat
276 gat nand    239 gat 191 gat
279 gat nand    243 gat 193 gat
282 gat nand    247 gat 195 gat
285 gat nand    251 gat 197 gat
288 gat nand    227 gat 184 gat
289 gat nand    230 gat 186 gat
290 gat nand    233 gat 188 gat
291 gat nand    236 gat 190 gat
292 gat nand    239 gat 192 gat
293 gat nand    243 gat 194 gat
294 gat nand    247 gat 196 gat
295 gat nand    251 gat 198 gat
296 gat and     260 gat 264 gat 267 gat 270 gat 273 gat 276 gat
                  279 gat 282 gat 285 gat
300 gat not     263 gat
301 gat not     288 gat
302 gat not     289 gat
```

```
303 gat not     290 gat
304 gat not     291 gat
305 gat not     292 gat
306 gat not     293 gat
307 gat not     294 gat
308 gat not     295 gat
309 gat not     296 gat
319 gat not     296 gat
329 gat not     296 gat
330 gat xor     309 gat 260 gat
331 gat xor     309 gat 264 gat
332 gat xor     309 gat 267 gat
333 gat xor     309 gat 270 gat
334 gat nand    8 gat   319 gat
335 gat xor     309 gat 273 gat
336 gat nand    319 gat 21 gat
337 gat xor     309 gat 276 gat
338 gat nand    319 gat 34 gat
339 gat xor     309 gat 279 gat
340 gat nand    319 gat 47 gat
341 gat xor     309 gat 282 gat
342 gat nand    319 gat 60 gat
343 gat xor     309 gat 285 gat
344 gat nand    319 gat 73 gat
345 gat nand    319 gat 86 gat
346 gat nand    319 gat 99 gat
347 gat nand    319 gat 112 gat
348 gat nand    330 gat 300 gat
349 gat nand    331 gat 301 gat
350 gat nand    332 gat 302 gat
351 gat nand    333 gat 303 gat
352 gat nand    335 gat 304 gat
353 gat nand    337 gat 305 gat
354 gat nand    339 gat 306 gat
355 gat nand    341 gat 307 gat
356 gat nand    343 gat 308 gat
357 gat and     348 gat 349 gat 350 gat 351 gat 352 gat 353 gat
                  354 gat 355 gat 356 gat
360 gat not     357 gat
370 gat not     357 gat
371 gat nand    14 gat  360 gat
372 gat nand    360 gat 27 gat
373 gat nand    360 gat 40 gat
374 gat nand    360 gat 53 gat
375 gat nand    360 gat 66 gat
376 gat nand    360 gat 79 gat
377 gat nand    360 gat 92 gat
378 gat nand    360 gat 105 gat
379 gat nand    360 gat 115 gat
380 gat nand    4 gat   242 gat 334 gat 371 gat
381 gat nand    246 gat 336 gat 372 gat 17 gat
386 gat nand    250 gat 338 gat 373 gat 30 gat
393 gat nand    254 gat 340 gat 374 gat 43 gat
399 gat nand    255 gat 342 gat 375 gat 56 gat
404 gat nand    256 gat 344 gat 376 gat 69 gat
407 gat nand    257 gat 345 gat 377 gat 82 gat
411 gat nand    258 gat 346 gat 378 gat 95 gat
414 gat nand    259 gat 347 gat 379 gat 108 gat
```

```
415 gat not     380 gat
416 gat and     381 gat 386 gat 393 gat 399 gat 404 gat 407 gat
                  411 gat 414 gat
417 gat not     393 gat
418 gat not     404 gat
419 gat not     407 gat
420 gat not     411 gat
421 gat nor     415 gat 416 gat
422 gat nand    386 gat 417 gat
425 gat nand    386 gat 393 gat 418 gat 399 gat
428 gat nand    399 gat 393 gat 419 gat
429 gat nand    386 gat 393 gat 407 gat 420 gat
430 gat nand    381 gat 386 gat 422 gat 399 gat
431 gat nand    381 gat 386 gat 425 gat 428 gat
432 gat nand    381 gat 422 gat 425 gat 429 gat


$  combinational logic example "c880"
$-----------------------------------------------------
$
$
$  total number of lines in the netlist .............   880
$  simplistically reduced equivalent fault set size =    942
$       lines from primary input   gates .......    60
$       lines from primary output  gates .......    26
$       lines from interior  gate outputs ......   357
$       lines from **   125 ** fanout stems ...   437
$
$       avg_fanin  =  1.90,     max_fanin  =  4
$       avg_fanout =  3.50,     max_fanout =  8
$
$
$
$
$
1 gat                              $... primary input
8 gat                              $... primary input
13 gat                             $... primary input
17 gat                             $... primary input
26 gat                             $... primary input
29 gat                             $... primary input
36 gat                             $... primary input
42 gat                             $... primary input
51 gat                             $... primary input
55 gat                             $... primary input
59 gat                             $... primary input
68 gat                             $... primary input
72 gat                             $... primary input
73 gat                             $... primary input
74 gat                             $... primary input
75 gat                             $... primary input
80 gat                             $... primary input
85 gat                             $... primary input
86 gat                             $... primary input
87 gat                             $... primary input
88 gat                             $... primary input
89 gat                             $... primary input
```

```
90 gat                  $... primary input
91 gat                  $... primary input
96 gat                  $... primary input
101 gat                 $... primary input
106 gat                 $... primary input
111 gat                 $... primary input
116 gat                 $... primary input
121 gat                 $... primary input
126 gat                 $... primary input
130 gat                 $... primary input
135 gat                 $... primary input
138 gat                 $... primary input
143 gat                 $... primary input
146 gat                 $... primary input
149 gat                 $... primary input
152 gat                 $... primary input
153 gat                 $... primary input
156 gat                 $... primary input
159 gat                 $... primary input
165 gat                 $... primary input
171 gat                 $... primary input
177 gat                 $... primary input
183 gat                 $... primary input
189 gat                 $... primary input
195 gat                 $... primary input
201 gat                 $... primary input
207 gat                 $... primary input
210 gat                 $... primary input
219 gat                 $... primary input
228 gat                 $... primary input
237 gat                 $... primary input
246 gat                 $... primary input
255 gat                 $... primary input
259 gat                 $... primary input
260 gat                 $... primary input
261 gat                 $... primary input
267 gat                 $... primary input
268 gat                 $... primary input
                        $... primary input
$
$
388 gat                  $... primary output
389 gat                  $... primary output
390 gat                  $... primary output
391 gat                  $... primary output
418 gat                  $... primary output
419 gat                  $... primary output
420 gat                  $... primary output
421 gat                  $... primary output
422 gat                  $... primary output
423 gat                  $... primary output
446 gat                  $... primary output
447 gat                  $... primary output
448 gat                  $... primary output
449 gat                  $... primary output
450 gat                  $... primary output
767 gat                  $... primary output
768 gat                  $... primary output
```

```
850 gat                                  $... primary output
863 gat                                  $... primary output
864 gat                                  $... primary output
865 gat                                  $... primary output
866 gat                                  $... primary output
874 gat                                  $... primary output
878 gat                                  $... primary output
879 gat                                  $... primary output
880 gat                                  $... primary output
                                         $... primary output

$
$
$      Output      Type  Inputs...
$      ------      ----  ---------
       269 gat     nand  1 gat 8 gat 13 gat      17 gat
       270 gat     nand  1 gat 26 gat      13 gat      17 gat
       273 gat     and   29 gat      36 gat      42 gat
       276 gat     and   1 gat 26 gat      51 gat
       279 gat     nand  1 gat 8 gat 51 gat      17 gat
       280 gat     nand  1 gat 8 gat 13 gat      55 gat
       284 gat     nand  59 gat      42 gat      68 gat      72 gat
       285 gat     nand  29 gat      68 gat
       286 gat     nand  59 gat      68 gat      74 gat
       287 gat     and   29 gat      75 gat      80 gat
       290 gat     and   29 gat      75 gat      42 gat
       291 gat     and   29 gat      36 gat      80 gat
       292 gat     and   29 gat      36 gat      42 gat
       293 gat     and   59 gat      75 gat      80 gat
       294 gat     and   59 gat      75 gat      42 gat
       295 gat     and   59 gat      36 gat      80 gat
       296 gat     and   59 gat      36 gat      42 gat
       297 gat     and   85 gat      86 gat
       298 gat     or    87 gat      88 gat
       301 gat     nand  91 gat      96 gat
       302 gat     or    91 gat      96 gat
       303 gat     nand  101 gat     106 gat
       304 gat     or    101 gat     106 gat
       305 gat     nand  111 gat     116 gat
       306 gat     or    111 gat     116 gat
       307 gat     nand  121 gat     126 gat
       308 gat     or    121 gat     126 gat
       309 gat     and   8 gat 138 gat
       310 gat     not   268 gat
       316 gat     and   51 gat      138 gat
       317 gat     and   17 gat      138 gat
       318 gat     and   152 gat     138 gat
       319 gat     nand  59 gat      156 gat
       322 gat     nor   17 gat      42 gat
       323 gat     and   17 gat      42 gat
       324 gat     nand  159 gat     165 gat
       325 gat     or    159 gat     165 gat
       326 gat     nand  171 gat     177 gat
       327 gat     or    171 gat     177 gat
       328 gat     nand  183 gat     189 gat
       329 gat     or    183 gat     189 gat
       330 gat     nand  195 gat     201 gat
       331 gat     or    195 gat     201 gat
       332 gat     and   210 gat     91 gat
```

```
333 gat     and   210 gat     96 gat
334 gat     and   210 gat     101 gat
335 gat     and   210 gat     106 gat
336 gat     and   210 gat     111 gat
337 gat     and   255 gat     259 gat
338 gat     and   210 gat     116 gat
339 gat     and   255 gat     260 gat
340 gat     and   210 gat     121 gat
341 gat     and   255 gat     267 gat
342 gat     not   269 gat
343 gat     not   273 gat
344 gat     or    270 gat     273 gat
345 gat     not   276 gat
346 gat     not   276 gat
347 gat     not   279 gat
348 gat     nor   280 gat     284 gat
349 gat     or    280 gat     285 gat
350 gat     or    280 gat     286 gat
351 gat     not   293 gat
352 gat     not   294 gat
353 gat     not   295 gat
354 gat     not   296 gat
355 gat     nand  89 gat      298 gat
356 gat     and   90 gat      298 gat
357 gat     nand  301 gat     302 gat
360 gat     nand  303 gat     304 gat
363 gat     nand  305 gat     306 gat
366 gat     nand  307 gat     308 gat
369 gat     not   310 gat
375 gat     nor   322 gat     323 gat
376 gat     nand  324 gat     325 gat
379 gat     nand  326 gat     327 gat
382 gat     nand  328 gat     329 gat
385 gat     nand  330 gat     331 gat
388 gat     buff  290 gat
389 gat     buff  291 gat
390 gat     buff  292 gat
391 gat     buff  297 gat
392 gat     or    270 gat     343 gat
393 gat     not   345 gat
399 gat     not   346 gat
400 gat     and   348 gat     73 gat
401 gat     not   349 gat
402 gat     not   350 gat
403 gat     not   355 gat
404 gat     not   357 gat
405 gat     not   360 gat
406 gat     and   357 gat     360 gat
407 gat     not   363 gat
408 gat     not   366 gat
409 gat     and   363 gat     366 gat
410 gat     nand  347 gat     352 gat
411 gat     not   376 gat
412 gat     not   379 gat
413 gat     and   376 gat     379 gat
414 gat     not   382 gat
415 gat     not   385 gat
416 gat     and   382 gat     385 gat
```

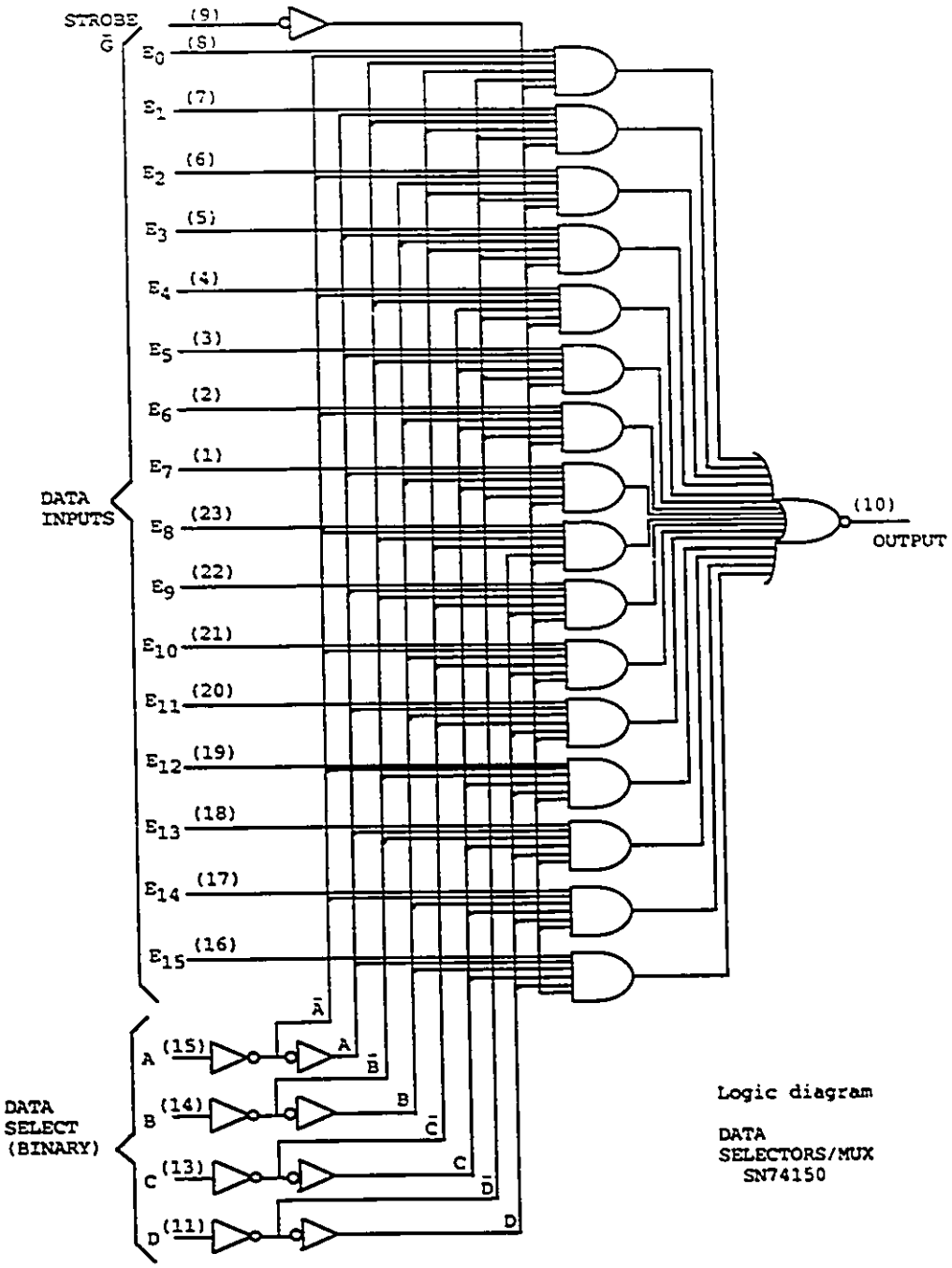
```
417 gat     and   210 gat     369 gat
418 gat     buff  342 gat
419 gat     buff  344 gat
420 gat     buff  351 gat
421 gat     buff  353 gat
422 gat     buff  354 gat
423 gat     buff  356 gat
424 gat     not   400 gat
425 gat     and   404 gat     405 gat
426 gat     and   407 gat     408 gat
427 gat     and   319 gat     393 gat     55 gat
432 gat     and   393 gat     17 gat      287 gat
437 gat     nand  393 gat     287 gat     55 gat
442 gat     nand  375 gat     59 gat      156 gat     393 gat
443 gat     nand  393 gat     319 gat     17 gat
444 gat     and   411 gat     412 gat
445 gat     and   414 gat     415 gat
446 gat     buff  392 gat
447 gat     buff  399 gat
448 gat     buff  401 gat
449 gat     buff  402 gat
450 gat     buff  403 gat
451 gat     not   424 gat
460 gat     nor   406 gat     425 gat
463 gat     nor   409 gat     426 gat
466 gat     nand  442 gat     410 gat
475 gat     and   143 gat     427 gat
476 gat     and   310 gat     432 gat
477 gat     and   146 gat     427 gat
478 gat     and   310 gat     432 gat
479 gat     and   149 gat     427 gat
480 gat     and   310 gat     432 gat
481 gat     and   153 gat     427 gat
482 gat     and   310 gat     432 gat
483 gat     nand  443 gat     1 gat
488 gat     or    369 gat     437 gat
489 gat     or    369 gat     437 gat
490 gat     or    369 gat     437 gat
491 gat     or    369 gat     437 gat
492 gat     nor   413 gat     444 gat
495 gat     nor   416 gat     445 gat
498 gat     nand  130 gat     460 gat
499 gat     or    130 gat     460 gat
500 gat     nand  463 gat     135 gat
501 gat     or    463 gat     135 gat
502 gat     and   91 gat      466 gat
503 gat     nor   475 gat     476 gat
504 gat     and   96 gat      466 gat
505 gat     nor   477 gat     478 gat
506 gat     and   101 gat     466 gat
507 gat     nor   479 gat     480 gat
508 gat     and   106 gat     466 gat
509 gat     nor   481 gat     482 gat
510 gat     and   143 gat     483 gat
511 gat     and   111 gat     466 gat
512 gat     and   146 gat     483 gat
513 gat     and   116 gat     466 gat
514 gat     and   149 gat     483 gat
```

```
515 gat     and   121 gat     466 gat
516 gat     and   153 gat     483 gat
517 gat     and   126 gat     466 gat
518 gat     nand  130 gat     492 gat
519 gat     or    130 gat     492 gat
520 gat     nand  495 gat     207 gat
521 gat     or    495 gat     207 gat
522 gat     and   451 gat     159 gat
523 gat     and   451 gat     165 gat
524 gat     and   451 gat     171 gat
525 gat     and   451 gat     177 gat
526 gat     and   451 gat     183 gat
527 gat     nand  451 gat     189 gat
528 gat     nand  451 gat     195 gat
529 gat     nand  451 gat     201 gat
530 gat     nand  498 gat     499 gat
533 gat     nand  500 gat     501 gat
536 gat     nor   309 gat     502 gat
537 gat     nor   316 gat     504 gat
538 gat     nor   317 gat     506 gat
539 gat     nor   318 gat     508 gat
540 gat     nor   510 gat     511 gat
541 gat     nor   512 gat     513 gat
542 gat     nor   514 gat     515 gat
543 gat     nor   516 gat     517 gat
544 gat     nand  518 gat     519 gat
547 gat     nand  520 gat     521 gat
550 gat     not   530 gat
551 gat     not   533 gat
552 gat     and   530 gat     533 gat
553 gat     nand  536 gat     503 gat
557 gat     nand  537 gat     505 gat
561 gat     nand  538 gat     507 gat
565 gat     nand  539 gat     509 gat
569 gat     nand  488 gat     540 gat
573 gat     nand  489 gat     541 gat
577 gat     nand  490 gat     542 gat
581 gat     nand  491 gat     543 gat
585 gat     not   544 gat
586 gat     not   547 gat
587 gat     and   544 gat     547 gat
588 gat     and   550 gat     551 gat
589 gat     and   585 gat     586 gat
590 gat     nand  553 gat     159 gat
593 gat     or    553 gat     159 gat
596 gat     and   246 gat     553 gat
597 gat     nand  557 gat     165 gat
600 gat     or    557 gat     165 gat
605 gat     and   246 gat     557 gat
606 gat     nand  561 gat     171 gat
609 gat     or    561 gat     171 gat
615 gat     and   246 gat     561 gat
616 gat     nand  565 gat     177 gat
619 gat     or    565 gat     177 gat
624 gat     and   246 gat     565 gat
625 gat     nand  569 gat     183 gat
628 gat     or    569 gat     183 gat
631 gat     and   246 gat     569 gat
```

```
632 gat     nand  573 gat     189 gat
635 gat     or    573 gat     189 gat
640 gat     and   246 gat     573 gat
641 gat     nand  577 gat     195 gat
644 gat     or    577 gat     195 gat
650 gat     and   246 gat     577 gat
651 gat     nand  581 gat     201 gat
654 gat     or    581 gat     201 gat
659 gat     and   246 gat     581 gat
660 gat     nor   552 gat     588 gat
661 gat     nor   587 gat     589 gat
662 gat     not   590 gat
665 gat     and   593 gat     590 gat
669 gat     nor   596 gat     522 gat
670 gat     not   597 gat
673 gat     and   600 gat     597 gat
677 gat     nor   605 gat     523 gat
678 gat     not   606 gat
682 gat     and   609 gat     606 gat
686 gat     nor   615 gat     524 gat
687 gat     not   616 gat
692 gat     and   619 gat     616 gat
696 gat     nor   624 gat     525 gat
697 gat     not   625 gat
700 gat     and   628 gat     625 gat
704 gat     nor   631 gat     526 gat
705 gat     not   632 gat
708 gat     and   635 gat     632 gat
712 gat     nor   337 gat     640 gat
713 gat     not   641 gat
717 gat     and   644 gat     641 gat
721 gat     nor   339 gat     650 gat
722 gat     not   651 gat
727 gat     and   654 gat     651 gat
731 gat     nor   341 gat     659 gat
732 gat     nand  654 gat     261 gat
733 gat     nand  644 gat     654 gat     261 gat
734 gat     nand  635 gat     644 gat     654 gat     261 gat
735 gat     not   662 gat
736 gat     and   228 gat     665 gat
737 gat     and   237 gat     662 gat
738 gat     not   670 gat
739 gat     and   228 gat     673 gat
740 gat     and   237 gat     670 gat
741 gat     not   678 gat
742 gat     and   228 gat     682 gat
743 gat     and   237 gat     678 gat
744 gat     not   687 gat
745 gat     and   228 gat     692 gat
746 gat     and   237 gat     687 gat
747 gat     not   697 gat
748 gat     and   228 gat     700 gat
749 gat     and   237 gat     697 gat
750 gat     not   705 gat
751 gat     and   228 gat     708 gat
752 gat     and   237 gat     705 gat
753 gat     not   713 gat
754 gat     and   228 gat     717 gat
```

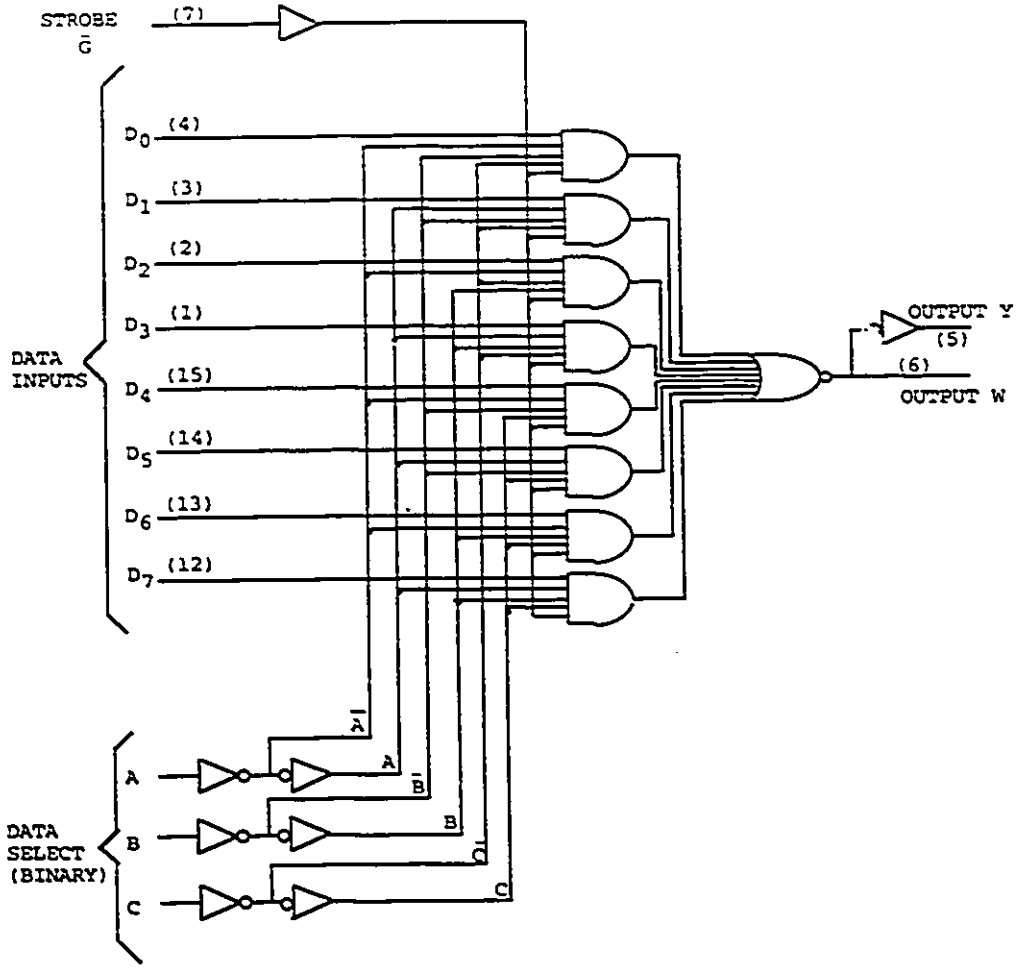
```
755 gat     and   237 gat     713 gat
756 gat     not   722 gat
757 gat     nor   727 gat     261 gat
758 gat     and   727 gat     261 gat
759 gat     and   228 gat     727 gat
760 gat     and   237 gat     722 gat
761 gat     nand  644 gat     722 gat
762 gat     nand  635 gat     713 gat
763 gat     nand  635 gat     644 gat     722 gat
764 gat     nand  609 gat     687 gat
765 gat     nand  600 gat     678 gat
766 gat     nand  600 gat     609 gat     687 gat
767 gat     buff  660 gat
768 gat     buff  661 gat
769 gat     nor   736 gat     737 gat
770 gat     nor   739 gat     740 gat
771 gat     nor   742 gat     743 gat
772 gat     nor   745 gat     746 gat
773 gat     nand  750 gat     762 gat     763 gat     734 gat
777 gat     nor   748 gat     749 gat
778 gat     nand  753 gat     761 gat     733 gat
781 gat     nor   751 gat     752 gat
782 gat     nand  756 gat     732 gat
785 gat     nor   754 gat     755 gat
786 gat     nor   757 gat     758 gat
787 gat     nor   759 gat     760 gat
788 gat     nor   700 gat     773 gat
789 gat     and   700 gat     773 gat
790 gat     nor   708 gat     778 gat
791 gat     and   708 gat     778 gat
792 gat     nor   717 gat     782 gat
793 gat     and   717 gat     782 gat
794 gat     and   219 gat     786 gat
795 gat     nand  628 gat     773 gat
796 gat     nand  795 gat     747 gat
802 gat     nor   788 gat     789 gat
803 gat     nor   790 gat     791 gat
804 gat     nor   792 gat     793 gat
805 gat     nor   340 gat     794 gat
806 gat     nor   692 gat     796 gat
807 gat     and   692 gat     796 gat
808 gat     and   219 gat     802 gat
809 gat     and   219 gat     803 gat
810 gat     and   219 gat     804 gat
811 gat     nand  805 gat     787 gat     731 gat     529 gat
812 gat     nand  619 gat     796 gat
813 gat     nand  609 gat     619 gat     796 gat
814 gat     nand  600 gat     609 gat     619 gat     796 gat
815 gat     nand  738 gat     765 gat     766 gat     814 gat
819 gat     nand  741 gat     764 gat     813 gat
822 gat     nand  744 gat     812 gat
825 gat     nor   806 gat     807 gat
826 gat     nor   335 gat     808 gat
827 gat     nor   336 gat     809 gat
828 gat     nor   338 gat     810 gat
829 gat     not   811 gat
830 gat     nor   665 gat     815 gat
831 gat     and   665 gat     815 gat
```

```
832 gat     nor   673 gat     819 gat
833 gat     and   673 gat     819 gat
834 gat     nor   682 gat     822 gat
835 gat     and   682 gat     822 gat
836 gat     and   219 gat     825 gat
837 gat     nand  826 gat     777 gat     704 gat
838 gat     nand  827 gat     781 gat     712 gat     527 gat
839 gat     nand  828 gat     785 gat     721 gat     528 gat
840 gat     not   829 gat
841 gat     nand  815 gat     593 gat
842 gat     nor   830 gat     831 gat
843 gat     nor   832 gat     833 gat
844 gat     nor   834 gat     835 gat
845 gat     nor   334 gat     836 gat
846 gat     not   837 gat
847 gat     not   838 gat
848 gat     not   839 gat
849 gat     and   735 gat     841 gat
850 gat     buff  840 gat
851 gat     and   219 gat     842 gat
852 gat     and   219 gat     843 gat
853 gat     and   219 gat     844 gat
854 gat     nand  845 gat     772 gat     696 gat
855 gat     not   846 gat
856 gat     not   847 gat
857 gat     not   848 gat
858 gat     not   849 gat
859 gat     nor   417 gat     851 gat
860 gat     nor   332 gat     852 gat
861 gat     nor   333 gat     853 gat
862 gat     not   854 gat
863 gat     buff  855 gat
864 gat     buff  856 gat
865 gat     buff  857 gat
866 gat     buff  858 gat
867 gat     nand  859 gat     769 gat     669 gat
868 gat     nand  860 gat     770 gat     677 gat
869 gat     nand  861 gat     771 gat     686 gat
870 gat     not   862 gat
871 gat     not   867 gat
872 gat     not   868 gat
873 gat     not   869 gat
874 gat     buff  870 gat
875 gat     not   871 gat
876 gat     not   872 gat
877 gat     not   873 gat
878 gat     buff  875 gat
879 gat     buff  876 gat
880 gat     buff  877 gat
```



Logic diagram

DATA SELECTORS/MUX
SN74150



Logic diagram

8-INPUT
MULTIPLEXER
SN74151

Truth Table

SN74150

INPUTS					
SELECT				STROBE	OUTPUT
D	C	B	A	$\overline{G}$	W
X	X	X	X	H	H
L	L	L	L	L	$\overline{E}_0$
L	L	L	H	L	$\overline{E}_1$
L	L	H	L	L	$\overline{E}_2$
L	L	H	H	L	$\overline{E}_3$
L	H	L	L	L	$\overline{E}_4$
L	H	L	H	L	$\overline{E}_5$
L	H	H	L	L	$\overline{E}_6$
L	H	H	H	L	$\overline{E}_7$
H	L	L	L	L	$\overline{E}_8$
H	L	L	H	L	$\overline{E}_9$
H	L	H	L	L	$\overline{E}_{10}$
H	L	H	H	L	$\overline{E}_{11}$
H	H	L	L	L	$\overline{E}_{12}$
H	H	L	H	L	$\overline{E}_{13}$
H	H	H	L	L	$\overline{E}_{14}$
H	H	H	H	L	$\overline{E}_{15}$

Truth Table

SN74151

INPUTS				OUTPUT	
SELECT			STROBE		
C	B	A	$\overline{G}$	Y	W
X	X	X	H	L	H
L	L	L	L	D_0	$\overline{D_0}$
L	L	H	L	D_1	$\overline{D_1}$
L	H	L	L	D_2	$\overline{D_2}$
L	H	H	L	D_3	$\overline{D_3}$
H	L	L	L	D_4	$\overline{D_4}$
H	L	H	L	D_5	$\overline{D_5}$
H	H	L	L	D_6	$\overline{D_6}$
H	H	H	L	D_7	$\overline{D_7}$