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

STOCHASTIC MODELING IN FAULT TESTING OF DECOMPOSABLE SEQUENTIAL CIRCUITS THROUGH COMPUTER SIMULATION

BY

Seong Yeon Choi

A THESIS SUBMITTED TO THE
SCHOOL OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF APPLIED SCIENCES

OTTAWA-CARLETON INSTITUTE FOR ELECTRICAL ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING
FACULTY OF ENGINEERING

ISBN 0-315-85835-4

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of the thesis. I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Seong Yeon Choi

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by other means, in total or part, at the request of other institutions or individuals for the purpose of scholarly research.

Seong Yeon Choi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to my thesis supervisor, Dr. Sunil R. Das, for his continuous encouragement, patience and guidance throughout this work.

I have to extend my appreciation to Dr. A. Nayak for giving me guidance in the absence of Dr. Das, who had to visit Stanford University for his research during his sabbatical year in 1991.

I am also indebted to Dr. H. W. Lee for consultations on the modeling and simulation problems.

Furthermore, I want to thank all the faculty members in the Department of Electrical Engineering of the University of Ottawa for providing the academic, especially computing, environment for the work of this thesis.

I would like to thank my wife, Diane Okyong, whose patience and support have made this effort possible.

Finally, I wish to thank my brother, Ho Yeon Choi, in London, Ontario for praying for me.

ABSTRACT

The increasing complexity of today's digital devices has rendered the problem of fault detection, fault analysis, and test generation extremely difficult. Test generation for sequential circuits has been a difficult task. This is due to the large search space to be considered in test pattern generation. Different approaches have been taken in the past to solve the problem of fault detection and test generation in sequential circuits. A popular approach, called the scan design, is often used where the test generation problem in sequential circuits is transformed into one of combinational circuits. Unfortunately, this approach is mostly restricted to synchronous sequential circuits free of critical races. Moreover, when a circuit is very large and complex, the test generation can be quite involved, making the ad hoc approaches ineffective. Therefore, alternative methods should be considered.

In this thesis, the detection of permanent faults in sequential circuits by random testing is analyzed utilizing the circuit partitioning approach together with a continuous parameter Markov model. Given a large decomposable sequential circuit, it is partitioned into several smaller partitions using either serial or parallel decomposition. For each partition with certain stuck faults specified, the original state table and its error version are derived from an analysis of the partition under fault-free and faulty conditions, respectively. Then by simulation of these two tables on a computer, the parameters of the desired Markov model are obtained. For a specified degree of confidence, it is easy to derive the parameters of the Markov model and to calculate the required lengths of random test patterns.

CHAPTER 1
INTRODUCTION

1.1 Overview

In recent years, due to the development of integrated circuit technology, the digital devices have become very complex as numerous basic gates and flip-flops can now be incorporated into a single device. With the growth of the complexity of digital devices, problems such as fault testing, fault analysis, test generation, and interpretation of the test results are now a time consuming, and expensive part of the manufacturing and maintenance process.

The testing process has grown in complexity as IC complexity grew. Initially it seemed reasonable to continue an approach that applied functional test vectors to verify the intended logic of the circuits. However, problems began to appear as circuits grew to large scale integrated circuit (LSI) levels of a hundred gates or more. The number of required test vectors increased dramatically, thus, increasing the testing time and cost for each IC.

One structural approach to address this problem is known as design for testability (DFT). The first generation of DFT was called ad hoc technology, where controllability and observability could be enhanced by using flip-flops with resetting capabilities, logically cutting the circuit to control feedback lines, or adding test points to the original circuit [1, 47, 48]. The second generation of DFT had more structured techniques and used two modes: the normal mode and test mode [1, 47, 48]. In the test mode, flip-flops were chained together to form a shift register while the circuit was tested. However, these techniques still require external tester for stimuli at the primary input terminal and response evaluation at the primary output terminals [37]. The third generation of DFT is often called built-in test (BIT), built-in self-test (BIST), self-test, autonomous test or self-verification. The main objective of these techniques is to reduce or to remove the cost of external testers. In these techniques, an additional hardware is built into the original circuit to do some or all of the tester functions such as test generation and test response evaluations. These techniques are gaining acceptance in the industry but they are restricted to the combinational circuits as they add to the circuit overhead, slowing down device speed and making it

difficult to estimate fault coverage for the test vectors generated. These problems in BIST increase the cost of device, making it unsuitable for high performance applications.

There are two methods of fault detection in a digital circuit; they are: deterministic methods and probabilistic methods. For the fault testing in combinational circuits, a number of basic analytic and heuristic methods have been developed. They include the fault table method [48], the path sensitization and equivalent normal form (ENF) method [48], the Karnaugh map method [13], the Structure and Parity Observing Output Function (SPOOF) method [17], the Boolean difference method [19], and the D-Algorithm [60].

The fault detection for sequential circuits is considered to be much more difficult than the one for combinational circuits. A single test vector can detect a fault in one time step in a combinational circuit; however, for fault detection in sequential circuit, this is not the case because a sequential circuit is one whose output depends on the value of the previous state. To test a sequential circuit deterministically, the circuit should be at a certain state, and it requires a sequence of test vectors for a single fault detection. Therefore, a great amount of time will be devoted to the calculation of complex test sequences to identify and to verify the whole circuit. To generate test vectors for a sequential circuit, the test engineer or the designer should have the exact knowledge of circuit behavior which is another cost sensitive matter. Therefore, the deterministic methods do not seem to be suitable for fault detection in large-state sequential circuits.

The sequential circuit requires a very large number of test vectors to ensure an exhaustive test since all combinations of states must be evaluated. The number of input test vectors S_S required to exhaustively test a sequential circuit is not $S_C=2^N$ but $S_S \geq 2^{N+M}$, where N is the number of inputs and M is the number of feedback elements (flip-flops) employed in the circuit [40]. In a circuit containing many sequential elements, the total number of test vectors must consider the total number of states. Each flip-flop in a circuit now adds a factor of two to the number of input test vectors. For example, an LSI circuit with N=25 and M=50 requires $S_S=3.8*10^{22}$ total test vectors. A 1-μs-per-test vector rate requires a testing time of $1.2*10^9$ years [40].

As shown in the abovementioned example, exhaustive testing is not a promising method for fault detection in large-state sequential circuits. As an alternative approach, probabilistic methods or random testing methods can be chosen since they are simple, less

time consuming and less expensive. The random input patterns can be easily generated by a computer or a linear feedback shift register which can produce 0s or 1s with any given probability. Conventionally, in random test generation methods, the effect of a failure in a logic network is propagated to the network output by applying random stimuli to the primary inputs of the network. By using a simulator, the outputs of the faulty and fault-free networks are then compared. If the comparison fails, the applied random input pattern is retained as a test. For complete fault detection, the experiment is continued until every network fault has been detected by at least one input pattern.

1.2 The Problem and Approach

For testing a large state sequential circuit, random testing is quite appropriate. It is a fact that the efficiency and the cost of testing a circuit is directly proportional to the length of the test input sequences. When a sequential circuit has a large number of states, the number of test sequences also increases since all the states have to be tested [29]. Therefore, we need an effective and economic testing technique.

In order to reduce the testing time and cost of testing large-state sequential circuits, a well known technique called 'divide and conquer' can be applied in conjunction with the probabilistic methods or random testing. Decomposition theory can be used to partition a large-state sequential circuit into a set of smaller-state circuits (component circuits). Once the decomposition process has been completed, each component circuit can be tested separately using the random testing technique. A precise mathematical model is required for the random testing approach. One should have a certain degree of confidence level in testing after applying a large number of random input patterns.

Given a large sequential circuit, the approach proposed herein is to decompose the circuit into several smaller partitions using the state substitution property. Each partition becomes a sequential circuit with a smaller number of states than the original circuit, thus making it easy to analyze each partition separately. Once all the partitions of the original circuit are obtained, we then focus on the problem of detecting permanent stuck faults in the partitions by random testing, using a three-state continuous parameter Markov model.

To prove the usefulness of the circuit partition approach, we simulated several well known sequential circuits and determined the testing time for detecting some typical faults

in the case where we partition the circuit and in the case where we do not partition the circuit. It turned out that the testing time was significantly lower in the case where we partitioned the circuit as compared to the other case.

1.3 Contributions

A random testing approach utilizing the circuit decomposition theory together with the continuous Markov model is presented in the thesis for the detection of permanent stuck faults in sequential circuits.

The principles and types of decomposition techniques for sequential machines have been reviewed and applied to random testing. As a tool of fault detection in sequential machines, a continuous parameter Markov model of a faulty sequential machine and its differential equations are defined and derived with a complete mathematical analysis. Testing time calculation, which has restrictions as indicated in [24], has been solved by using a numerical method to solve the nonlinear equations. The new testing time calculation approach can now be considered more accurate than the one previously known.

Based on the proposed approach, a fault detection procedure and an algorithm are developed. The benefits of the proposed approach against its counterparts are described. Experimentation on various sequential circuits backs the claim that a significant saving in testing time can be achieved if we can partition a large circuit and then test each of its components as opposed to testing the circuit in its original form. This approach holds for both combinational and sequential circuits. In this thesis, we have considered the most difficult case: the sequential circuits.

1.4 Organization of the Thesis

The thesis is organized as follows. Chapter 2 gives an overview of the fault detection methods in sequential circuits. Advantages and disadvantages of both the deterministic methods and the probabilistic methods are described in this chapter.

In Chapter 3, the principles and types of decomposition for sequential machines are discussed. In this chapter, important properties of decomposition theory are presented.

Based on the decomposition theory, two types of decompositions such as serial and parallel decomposition are discussed.

In Chapter 4, as a tool for fault detection and analysis in sequential machines, a tri-state continuous Markov model is presented. In this chapter, differential equations for the model are derived, and the methods for calculating Markov parameters and the state probabilities are discussed. Testing time calculation for both the closed form solution and the numerical approximation is given.

Chapter 5 presents a random testing approach for the detection of permanent faults in sequential circuits utilizing the decomposition theory (Chapter 3) and the continuous Markov model (Chapter 4). In this chapter, a brief outline of the existing approaches including the basic circuit partitioning approach and the deterministic approach in decomposable sequential machines are given. A random testing strategy using a continuous parameter Markov model is also discussed in this chapter. Discussions on fault detection procedure, data structure for sequential machines, and algorithm development are presented. The benefits of the proposed approach against its counterparts are described followed by experimentations.

Finally, the thesis is summarized in Chapter 6 which is then followed by an Appendix which contains the simulation program used for experimentation.

CHAPTER 2

TESTING OF SEQUENTIAL CIRCUITS: A SURVEY

2.1 Introduction

The fault detection or test generation for sequential circuits is much more difficult than the one for combinational circuits since the behavior of a sequential circuit depends both on the present inputs and the past inputs. Therefore, an entire sequence of inputs must be applied to detect a single fault in a sequential circuit. The fault detection in sequential circuits can be broadly classified as either deterministic or probabilistic. In this chapter, various fault detection strategies based on deterministic and probabilistic methods for sequential circuits are surveyed. Furthermore, the issue of automatic test pattern generation is also discussed in this chapter.

The chapter is organized as follows. Section 2.2 gives an overview of the common deterministic methods such as the iterative array approach and the verification-based approach. Section 2.3 discusses the probabilistic methods for fault detection. A brief description on the issue of automatic test pattern generation is given in Section 2.4.

2.2 Deterministic Methods

In using deterministic methods for fault detection in sequential circuits, we analyze the behavior of the circuits in the presence of a fault and generate a test sequence to detect that specific fault. These methods have been described in detail [16, 47, 50].

Two common deterministic methods on fault detection in sequential circuits [50] are the iterative array approach and the state table verification approach. These two methods will be reviewed in this section.

2.2.1 Iterative Array Approach

A synchronous sequential circuit can be converted into a series of identical combinational circuits cascading all together to give the same operation mode for the original circuit. The behavior of a sequential circuit depends on the primary inputs and the internal states. The internal states maintain their values for a certain period of time. For this period, the sequential circuit can be modelled as a combinational circuit. This combinational model for the sequential circuit is constructed by replacing the feedback signals with previous time copies of the circuits. Then we can apply the combinational test generation techniques such as the D-algorithm, the path sensitization method, and the Boolean difference method to the model. As an example, consider a simple latch shown in Fig. 2.1.

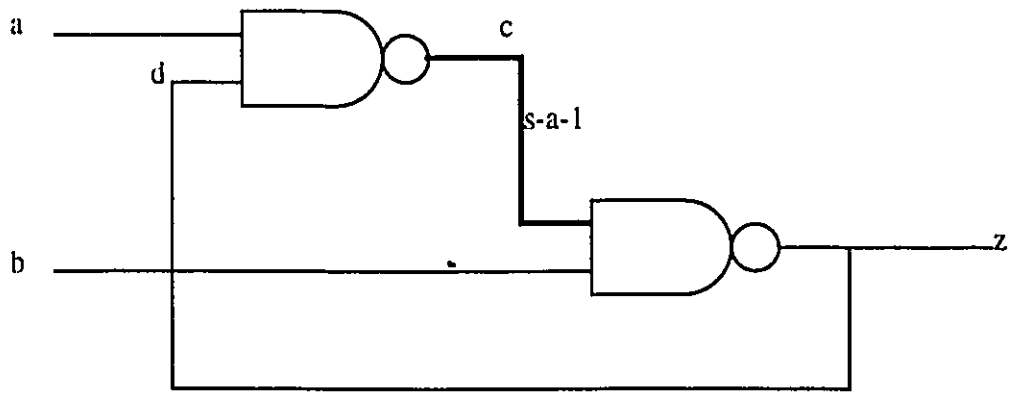


Fig. 2.1 NAND latch.

To test the latch operation when stuck-at-1 fault on the signal line c, a test pattern ab=10 followed by 11 is required. In other words, we have to hold the feedback signal line d=z at 1 and then the primary input line b at 1 to sensitize the path for the fault. If we cut the feedback signal, we can obtain a combinational model for the latch circuit as shown in Fig. 2.2, where a copy of the circuit is added to generate the feedback signal d, and the test generation begins from the current frame copy.

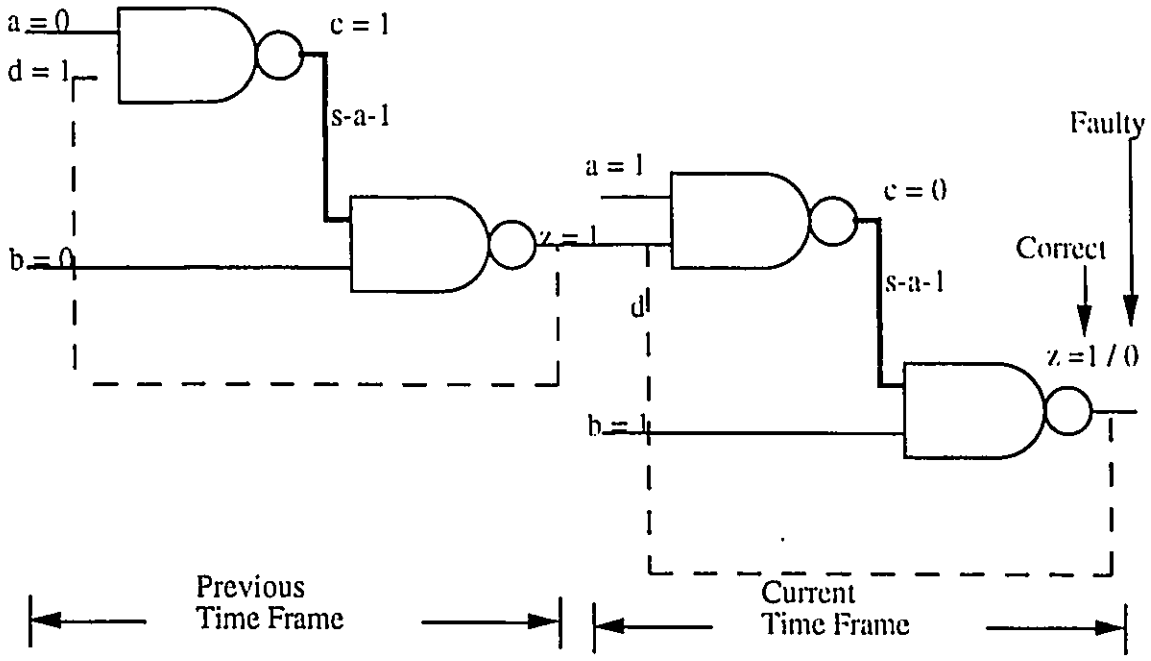


Fig.2.2 A combinational model of NAND latch.

To generalize the iterative approach, let us consider a schematic representation of a sequential circuit as shown in Fig. 2.3. The primary output is dependant on the primary inputs and the present state of memory. Using the concept discussed above, we can obtain a combinational model as shown in Fig 2.4. Feedback lines are cut and combinational parts are duplicated in each array. There are n copies of a combinational circuit to replace sequential circuit. The copy i is the ith copy of the combinational portion and corresponds to the time frame i. I_i is the primary inputs to copy i and the pseudo inputs SI_i obtained from the feedback signals produced by copy (i-1). And also, O_i and SO_i are the primary outputs and pseudo outputs of the time frame i.

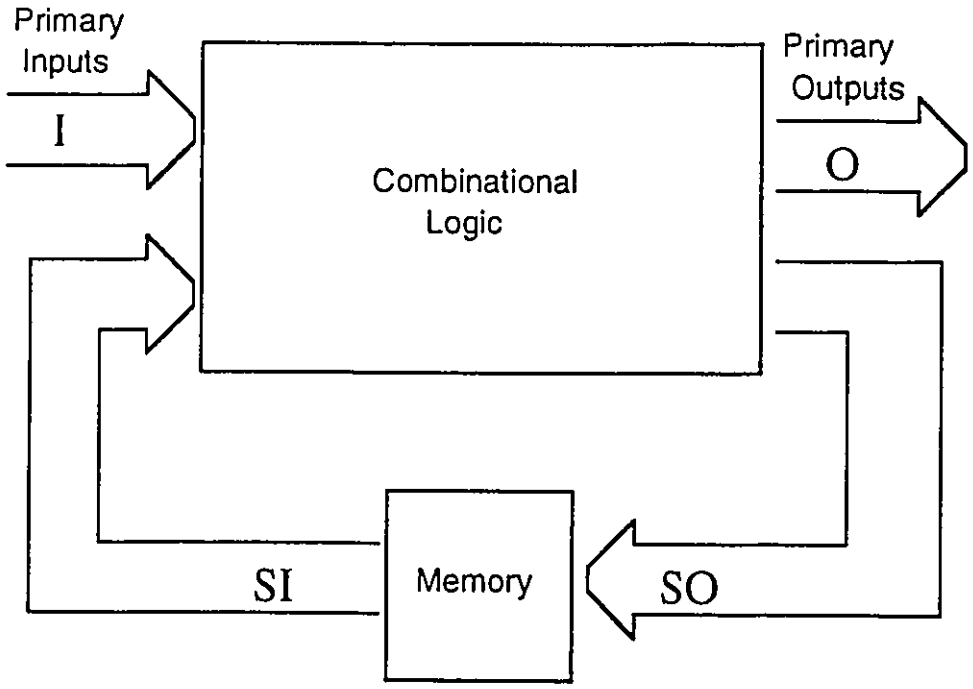


Fig.2.3 A model of sequential circuit.

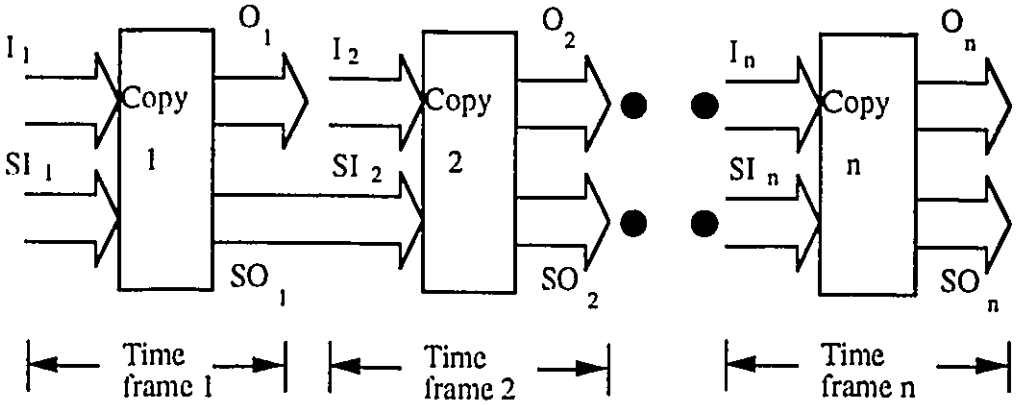


Fig 2.4 Iterative combinational model.

Once the combinational model is derived for the sequential circuit, various techniques used for test generation for the combinational circuit can be applied to the model.

Other works related to the iterative array approach are the Extended D-Algorithm [16], the Nine-Value Algorithm [55], SOFTG [2], and the Backtrace Algorithms [51]. However, the following problems exist with the iterative approach:

1. As the number of time frames increases, the number of copies of the circuit becomes large, and also the complexity of the model increases.

2. A race situation may occur due to the unspecified signals (Timing problem).

2.2.2 Verification-Based Approach

In this approach, we verify whether or not a given circuit is operating correctly in accordance with its state table. In other words, the performance of a sequential circuit described by a state table is verified by applying an "experiment", an input sequence to the primary input terminals, and observing the outputs.

The requirement for these approach is to find a checking experiment, which is an experiment designed to take the machine through all its state transitions in such way that a definite conclusion can be reached as to whether or not the machine is operating correctly. The checking experiment is often called fault detection experiment or machine identification experiment.

2.2.2.1 Checking Experiments

The checking experiments are classified as follows:

1) *Adaptive experiments*, in which the input at any instant of time depends on the previous outputs. An experiment is adaptive if the applied input sequence consists of a succession of subsequences, each of which is determined on the basis of the responses previously produced.

2) *Preset experiments*, in which the entire input sequence is predetermined independently of the outcome of the experiment. An experiment is preset if the input sequence or sequences to be applied to the machine are completely specified in advance.

The checking experiments of a sequential circuit are implemented under the following assumptions [22, 44] regarding its structure.

1) Fully specified and deterministic
2) Strongly connected
3) The fault does not cause an increase in the number of states.

The goal of testing using a checking experiment is to verify that a sequential machine is described by a particular state table, that of the fault-free machine. During the execution of the checking experiments, if the outputs of the machine are different from the those of a fault-free machine, we can conclude that the machine is faulty. Actually, this approach is functional testing. This testing can detect the fault but cannot locate the fault in the machine.

2.2.2.2 Designing Checking Experiments

Before going into the details of designing checking experiments, we will briefly give the definitions and terminologies involved.

An *experiment* performed on a sequential machine consists in applying one or more input sequences, observing the corresponding output sequences, and drawing the conclusion about the internal behavior of the machine.

One major class of experiments involves machines whose state tables are known but whose initial states are not. Such experiments are of three types. For a v-state sequential machine, a *distinguishing experiment* consists in the application of an input sequence to the machine which will result in the production of v different output sequences, and thus uniquely specifying the machine's unknown initial state.

A *homing experiment* is is an experiment that is used to determine the machine's final state, irrespective of the initial state, by applying an input sequence and observing the corresponding output response produced.

A *synchronizing experiment* which is a very special case of the homing experiment consists in the application of an input sequence which is guaranteed to leave the machine in a specific, predefined final state regardless of the responses and the unknown initial states. Reset is a widely used synchronizing experiment.

A collection of states of a specific machine M which is known to contain the present state of M is referred to as the *uncertainty*.

An uncertainty is called *trivial*, if it contains a single state, and is called *multiple*, if it contains two or more identical states.

An *uncertainty vector* is a collection of uncertainties, which consists of a total of p elements from the set S, where p is the size of the initial uncertainty and S is the total number of states.

An uncertainty vector is called *trivial*, if it contains only simple uncertainties, and is called *homogeneous*, if its elements are either simple, or multiple, or are both.

The uncertainty that follows from the application of an input sequence I_k to the initial uncertainty is called the I_k-*successor*.

The *successor tree*, which is defined for a specific machine M and a given initial uncertainty, is a graphical structure that displays the I_k-successor uncertainties for every possible I_k.

The successor tree for final state identification is called a *homing tree*. The *nodes* of this tree correspond to the possible states that the machine can be in, after the application of the input sequences that lead to these nodes. Each branch in the successor tree is associated with a node which represents an uncertainty vector.

There are are three phases to the design of checking experiments:

1) *Initialization Phase*. In this phase, the machine is brought to a known state.

2) *State Identification Phase*. In this phase, the existence of each state of machine as specified by its state table is verified.

3) *Transition Verification Phase*. In this phase, every state transition as specified by state table is checked.

The design algorithm for the checking experiments is outlined in the following steps:

Step 1: For a given state table of a sequential machine, draw a successor tree of the machine, and find a (some) homing sequence(s), distinguishing sequence(s), and synchronizing sequence(s).

Step 2: Apply a homing sequence to the machine, observe the response as the sequence is applied. This should uniquely identify the final state of the machine.

Step 3: Set the initial state S_i for the distinguishing sequence. If the final state of homing sequence was not S_i, apply a transfer sequence to move the state to S_i.

Step 4: Apply a distinguishing sequence and verify that all the responses and the final states are distinct.

Step 5: Apply the distinguishing sequence respectively verifying the existence of each state of machine as specified by its state table.

Step 6: Using the distinguishing sequence, check every state transition as specified by state table.

Finding a homing sequence for a given sequential circuit

1. Draw a homing tree, a successor tree, for every possible input sequence I_k, which shows I_k-successor uncertainty.
2. Terminate a node if any of following conditions is met.
   i) The node is associated with an uncertainty vector whose nonhomogeneous components are associated with a node at a preceding level.
   ii) The node is associated with a trivial or a homogeneous uncertainty vector.

Finding a distinguishing sequence

1. Draw the distinguishing tree, a successor tree, for every possible input sequence I_k ,which shows I_k-successor uncertainty.
2. Terminate a node if any of following conditions is met.
   i) The node is associated with an uncertainty vector whose nonhomogeneous components are associated with a node at a preceding level.
   ii) The node is associated with a trivial or a homogeneous uncertainty vector.
   iii) The node is associated with an uncertainty vector containing a homogeneous nontrivial component.

Finding a synchronizing sequence

1. Draw the synchronizing tree, a successor tree, for every possible input sequence I_k ,which shows I_k-successor uncertainty.
2. Terminate a node if any of following conditions is met.
   i) The node is associated with an uncertainty vector that is associated with a node at a preceding level.
   ii) The node in the nth level is associated with an uncertainty containing just a single element.

2.3 Probabilistic Methods

A major drawback of deterministic methods of test generation is that these methods may become prohibitively expensive for large and complex logic circuit. The random test generation methods which come under the category of probabilistic test generation can overcome some of the disadvantages inherent in the deterministic methods.

In recent years, interest in random testing of digital circuits has been growing steadily, primarily in view of the increasing size and complexity of the digital circuits, and the apparent simplicity of these methods [2, 5, 9, 12, 29, 30, 42, 46, 57, 58, 66, 67, 70]. Some researchers have, however, expressed reservations with regards to the overall effectiveness of random testing methods; they claim that the methods work quite well for some circuits, but may fail for others. They have tried this phenomenon to explain with the help of either the parameters such as latency and detection surface, or the models of circuit structure. These analyses indicate that random testing has certain limitations. A simple method by which one can, without doing a logic analysis, determine if a circuit can be tested by random inputs can be found in [5].

2.3.1 Test Procedures

The existing random testing procedures are carried out in two steps. In the first step, known as test generation, a set of input patterns is selected from random patterns such that this set will detect all of the single fault(s) in the circuit. The second step of procedure requires application of identical inputs to the circuit under test (CUT) and a fault-free circuit (or its digital simulator), and the outputs are then compared. In either case, one needs to know the number of random input patterns that will complete the testing with a high probability. For an n-input circuit, if the number is less than 2^n, then random testing is possibly recommended. The required number of random input patterns can be readily found through logic analysis. But for very large circuits, the efforts involved in such a preset analysis may be of the same order as that needed in generating tests by other algorithm. However, it must be remembered that it is only in the case of large circuits that the random test generation methods may offer some advantages over the other methods.

The random input patterns are easily generated by using a computer which can produce 0s or 1s with any given probability. Several authors have emphasized the need for selecting this input probability such that the probability of sensitizing the longest path in the circuit is maximized. Using a NAND model, it has been shown [2] that the probability p of setting an input line to 1 in a random pattern should satisfy the following equation: $p=1-p^n$, which gives the required condition to make the probability of a 1 at the output of an n-input NAND gate equal to the probability of 1 at its inputs.

Conventionally, in the random test generation methods, the effect of a failure in a logic network is propagated to the network output by applying random stimuli to the primary inputs of the network. By using a simulator, the outputs of the faulty and fault-free networks are then compared, as shown in Fig.2.5. If the comparison fails, the applied random input pattern is retained as a test. For complete fault detection, the experiment is continued until every network fault has been detected by at least one input pattern.

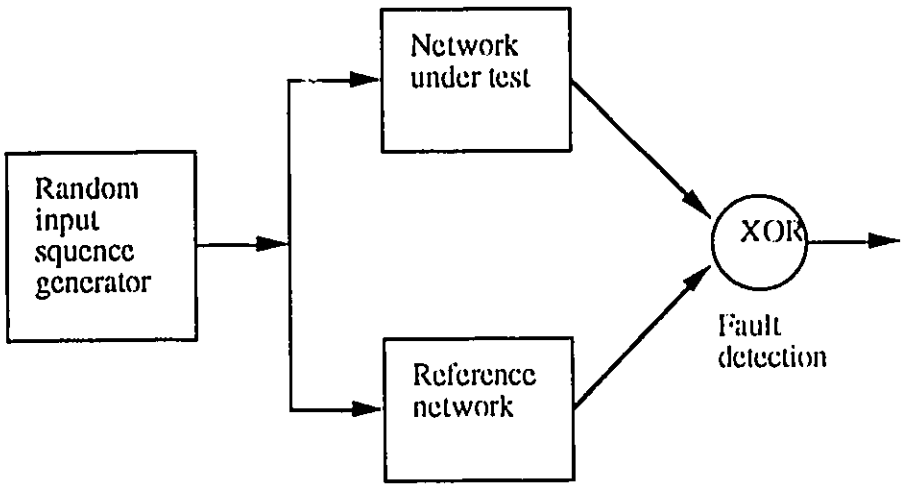


Fig. 2.5 Principle of random testing.

The accuracy of the random testing method depends on the length of the bit stream, the number of test vectors, and thus it is important to investigate the relationship between the bit stream length and the associated degree of confidence. Note here that in random testing, it is rather important to know the number of random input patterns that will complete the overall circuit testing with a high measure of confidence.

2.3.2 Models of Random Testing

The use of probabilistic models to characterize the behavior of digital circuits in the presence of faults has been discussed in the literature. Some of these models have been introduced for the detection of intermittent faults, while some others have been proposed for the detection of permanent or nontransient faults by random patterns. An intermittent fault is a fault which, while existing in the system, may be active at one instant of time causing a system malfunction, but may be inactive at another instant allowing the system to operate correctly.

A fault in a digital system is to said to be permanent or nontransient, if and only if, without having the fault fixed or repaired, it will remain there permanently. For describing the behavior of circuits with intermittent faults, Breuer [12] introduced a discrete parameter Markov model of first order, whereas Kamal and Page [42] presented a discrete parameter Markov model of zero order, which was subsequently used by Savir [61], and also by Koren and Kohavi [46]. Su *et al.* [70] proposed a continuous parameter Markov model with two states to characterize the behavior of digital systems in the presence of intermittent faults. This continuous parameter Markov model is a generalization of the discrete parameter model; it is believed to be more realistic, convenient to use, and to remove some of the restrictions faced when using a discrete model.

A fault in a digital circuit need not cause an immediate error in the circuit output; there is typically a delay between the occurrence of a fault and the first error to appear in the circuit output. This delay is the error latency of the fault. The error latency is an attribute of the fault; it depends on the circuit, the fault, and the input pattern applied to the circuit. Shedletsky and McCluskey [67] analyzed random testing of digital circuits using the error-latency model (ELM). This model proved very useful for the analysis of random testing procedures in sequential circuits. In earlier works [66], it was shown that the error latency in combinational circuits in certain cases may be comparable to the mean time between failures (MTBFs). Such a large error latency basically indicates the presence of correct output sequences long after a fault has occurred in a circuit.

When random inputs generated by a stationary multinomial process are applied to a deterministic sequential circuit, the probability of the circuit of being in a given state at time

t depends only on the state of the circuit at time t-1, and the probabilities of the input vectors at time t-1. The operation of the circuit under these conditions may be described by a finite, first order Markov process [35]. The Markov process is also ergodic and stationary if the circuit is strongly connected, and the input probabilities do not change with time [67].

A fault occurring in a sequential circuit transforms the state table from the correct version into a faulty version. Any model to determine the error latency of a particular fault must take into consideration the correct behavior of the fault-free circuit as well as the actual behavior of the faulty circuit. A normal procedure is to form a product state table, called an ELM state table, using the state tables of the fault-free and faulty circuits. The state diagram corresponding to the ELM state table is next transformed into a Markov chain by assigning probabilities to the different state transitions. An additional absorbing state S_4 is added for the detection of the fault. By calculating the probability distribution function of the error latency of the fault, the relation between the desired quality of test and necessary length of the random test pattern can be obtained. The ELM table is very efficient for small or medium size digital circuits; however, for large digital circuits, the ELM approach may prove somewhat difficult because of the complexity involved in generating the product state table, and forming their corresponding transition matrices. The authors of ELM demonstrated, using the exact analysis provided by the ELM, that some of the previous methods used in the analysis of random testing procedures led to answers substantially different from the correct ones.

2.4 Automatic Test Pattern Generation (ATPG)

The problem of automatic test pattern generation (ATPG) in the test environment demanded new approaches and new software tools to allow the test process to be done accurately and quickly. Generation of large numbers of test vectors manually had become expensive, consuming a large fraction of the design cycle time as IC's grew in size. In response, automatic test pattern generation software tools were developed that partially freed designers from this task and sought to increase the accuracy or fault coverage of test vector sets. Computer programs known as fault simulators were also developed to determine if the test vectors written for a particular circuit did indeed test all inside the circuit. Commercial testers grew in complexity and became multimillion-dollar computer

systems capable of stimulating a complex IC and evaluating its response [37]. By applying test vectors at megahertz clock frequencies, testers could evaluate smaller ICs in milliseconds, allowing high-volume test rates. Finally, special test circuits were designed to enhance testability. Some test circuits called scan-path circuits were incorporated within the system logic on the chip allowing inspection of internal nodes on the circuit that were not easily accessible by the system I/O pins. Other test circuits were incorporated on the chip and allowed the IC to test itself in a technique known as built-in self-test (BIST) [1]. Collectively, these methods that placed special test circuits on the IC became known as design-for-testability (DFT) techniques [1, 47, 48]. DFT techniques are the modern basis of a structured approach for addressing the VLSI circuit testability problems.

A comprehensive test set that evaluates an SSI circuit can be easily created by manual inspection, but LSI and VLSI circuit complexity requires computer-assisted methods of test generation. The large numbers of gates in VLSI circuits have pushed computer automatic test generation times to weeks or months of computation. Gate counts on the order of hundreds of thousands of logic gates require numbers of test vectors that outstrip the time reasonablely available for production testing.

Another test generation problem is that computer algorithms providing ATPG work well for combinational logic but rather poorly for sequential logic circuits. Sequential circuits demand too much computer memory and computation since many more time states must be evaluated. One solution has been the imposition of a structured scan-path design [1, 47, 48] such that sequential and combinational logic are segregated on the chip during the test process. It has been shown that the test set volume for these structured scan designs grows proportionately with the gate count. Circuit partitioning and built-in test designs are methods that address solutions to these problems.

Designers and test engineers use a computer-aided design (CAD) tool called a fault simulator to evaluate their test vector sets on a given circuit. Fault simulators provide an indication of which faults in a given circuit are not detectable by the test set. It is known that computer computation time for fault simulation grows at least as fast as the square of the gate count, leading to circuits that cannot be fault simulated without using partitioning techniques [40]. In summary, increased VLSI circuit size and the presence of sequential circuits impose limitations on how test vectors can be automatically generated and how these same circuits may be fault simulated. A list of fault simulators can be found in [16].

Very small circuits do not present test generation problem because one can simply use an exhaustive pattern set to test a combinational circuit. However, as circuit sizes and pin counts have grown beyond acceptable limits for exhaustive testing, other methods have been developed for detecting specific circuit faults.

Test generation for combinational circuits can be difficult, and in sequential circuits it is even more. The most common solution to ATPG for sequential circuits is to use scan path method that creates a test mode in which the sequential circuit becomes an equivalent combinational circuit. Not all circuits, though, are amenable to scan path methods, and so there is still a need for sequential test generation.

Early approaches in sequential ATPG proposed modification to successful combinational ATPG algorithms, like the D-Algorithm [60], to account for the time dependency of node logic values [40]. Present sequential ATPG systems still rely heavily on extensions of combinational ATPG algorithms (combinational circuits are essentially a subset of sequential circuits; so the carryover of test generation algorithms should not be surprising).

Sequential ATPG algorithms must address circuit issues that affect the way in which the logic may be controlled. Selections for node settings must not only perform sensitization and propagation in time and for the combinational logic, but other complexities, like circuits timing delays, must be considered. Reverse time processing (RTP) and indexed backtracking have been developed as measures to overcome the added complexity of the test pattern search process for sequential circuits [37]. RTP starts with the desired outcome (a fault signal being propagated to a primary output) and proceeds backwards in time to find logic states that may cause the desired outcome. For instance, a node logic value will first be determined at one step in the process, and then, the state of internal memory elements will be determined to support the desired node logic value (along with any combinational input requirements). The next step is to determine the necessary clocking and input stimulus required to set the internal memory elements into the previously determined state. The process is carried out until a consistent sequence of input patterns is developed that supports the desired outcome (or until it is determined that no input pattern set can support the desired outcome). Indexed backtracking is the process of associating

search decision information with various search branching points along the search path. When an inconsistency arises, the search algorithm can more quickly find the most likely search branch responsible for the inconsistent results (thus minimizing the amount of solution search space explicitly searched).

2.5 Chapter Summary

The fault detection in sequential circuits can be broadly classified as either deterministic methods or probabilistic methods. In this chapter, some important fault detection strategies based on deterministic and probabilistic methods for sequential circuits were surveyed. Furthermore, the issue of automatic test pattern generation was also discussed in this chapter.

CHAPTER 3

DECOMPOSITION OF SEQUENTIAL MACHINES

3.1 Introduction

The complex very large scale integrated (VLSI) circuits require a structured and hierarchical design methodology to cope with design complexity [53]. One of the current trends in digital system design to meet this requirement is modularity. This is due to the fact that the modularity design methodology provides us many advantages such as reduction of dependencies between the system components, easy installation, testability (easy to locate the fault in the system level), maintainability, and system expandability. We can take a good example for this modularity design concept of integrated circuits from the programmable logic array (PLA) for the combinational circuits and programmable logic sequences (PLS) for the sequential circuits.

One of the approaches of modular design is the decomposition of a sequential machine M into two or more components. The main objective is to decompose a large-state machine (composite machine) into a set of fewer-state machines (submachines) with minimal interdependency between its subsystems [44], which will reduce complexity.

This chapter deals with the principles and types of decomposition for sequential machines. Important properties of decomposition theory are discussed in detail in Section 3.2. In Section 3.3, based on the decomposition theory, two types of decompositions, such as serial and parallel decomposition are discussed with examples.

3.2 Basic Theory for Decomposition of Sequential Machines

For many years, research on the decomposition theory of sequential machines has been going on. The main concern is how to realize a complex sequential machine as a combination of smaller component machines. For decomposition theory, two mathematical models for sequential machines are defined. Then, as a mathematical tool for decomposition of sequential machines, the substitution property partition (SP partition), which leads to the necessary and sufficient conditions for decompositions of sequential machines, is presented.

3.2.1 Machines

A sequential machine can be defined as a circuit whose output depends not only on the present input of the machine, but also on the past history as stored in the states. Fig 3.1 shows the schematic representation of a sequential machine M which contains a memory unit and a combinational logic unit, where X_1 to X_p and y_1 to y_q are denoted as inputs and outputs for the combinational logic unit and for the memory unit respectively, and Y_1 to Y_q and Z_1 to Z_r are outputs and inputs for the combinational logic unit and for the memory unit, respectively.

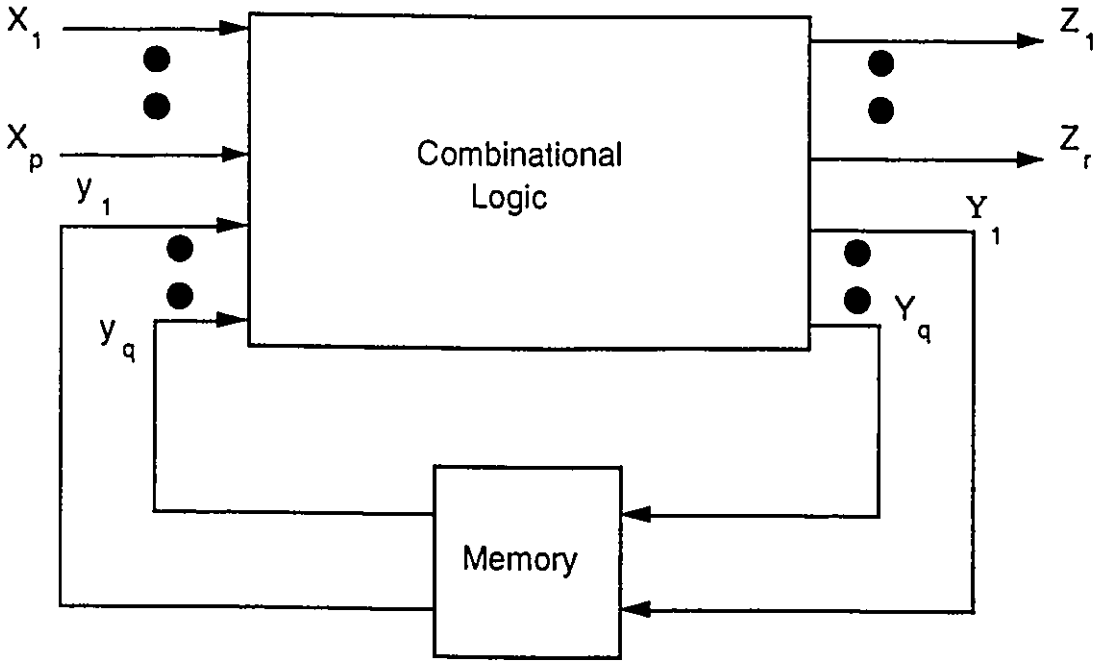


Fig 3.1 Schematic representation of the machine M

According to the mathematical properties, a machine can be classified into two models: Mealy model and Moore model.

The Mealy model machine M can be defined formally by a quintuple

$M = < I,S,O,f,g >,$

where $I = \{x_1, .., x_p\}$: input alphabet.

$S = \{y_1, .., y_q\}$: state alphabet.

$O = \{z_1, .., z_r\}$: output alphabet.

$f: S \times I \rightarrow O$: output mapping function for a Mealy machine.

$g: S \times I \rightarrow S$: next state mapping function.

In a Mealy model machine, or simply Mealy machine, the next state is dependent on the present state and the present input, and the present output is also dependent on the present state and the present input. Thus, the output variables are functions of the input variables as well as the state variables.

The Moore model machine M can be defined formally by a quintuple

$M = < I,S,O,f,g >,$

where $I = \{x_1, .., x_p\}$: input alphabet.

$S = \{y_1, .., y_q\}$: state alphabet.

$O = \{z_1, .., z_r\}$: output alphabet.

$f: S \rightarrow O$: output mapping function for a Moore machine.

$g: S \times I \rightarrow S$: next state mapping function.

In a Moore model machine, or simply Moore machine, the next state is dependent on the present state and the present input, but the present output is dependent only on the present state. Thus, the output variables are functions of the state variables only. The difference between a Mealy model machine and a Moore model machine can be clearly illustrated in a diagrammatic form. The output mapping function f and the next state mapping function g for any two consecutive time instants, i.e., the present time instant t_k

and the next time instant t_{k+1}, for both the Mealy machine and the Moore machine, are shown in Fig. 3.2.

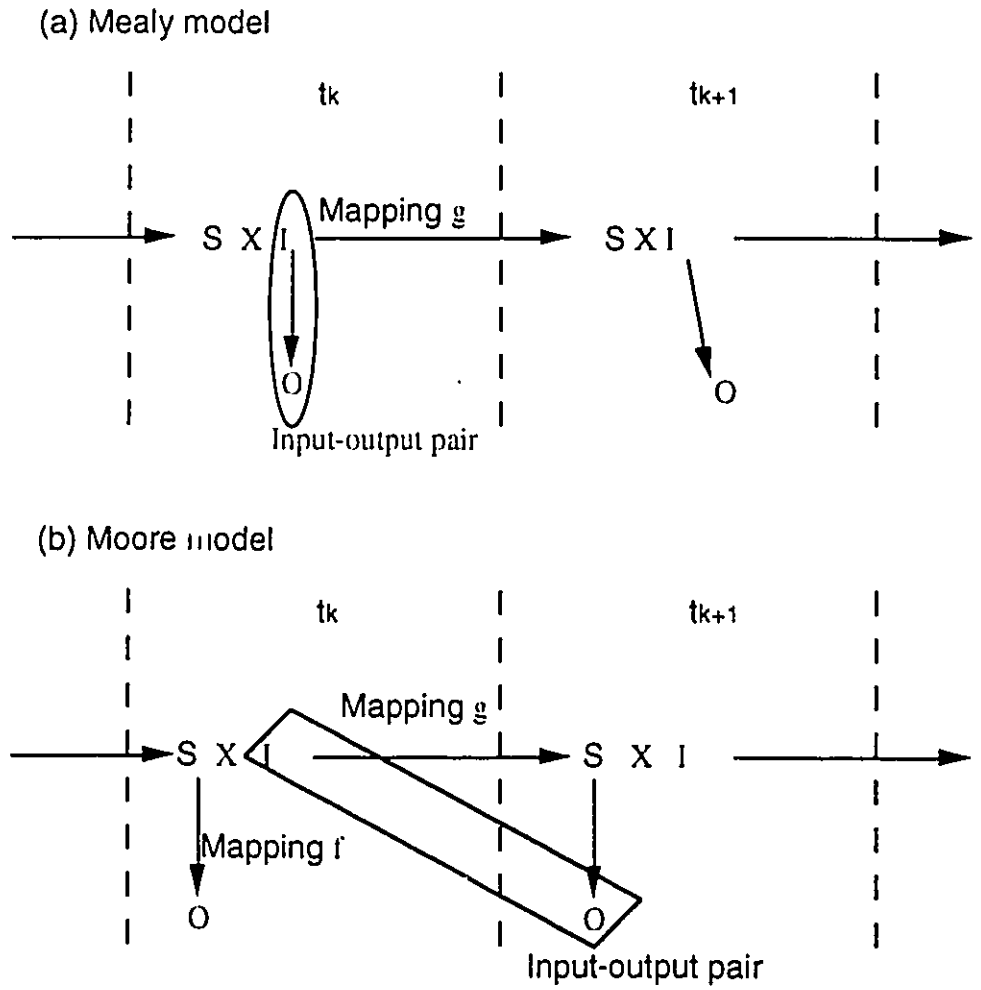


Fig. 3.2 Diagrammatic illustration of the difference between Mealy model and Moore model.

With the definitions given above, the sequential machines considered in this section are deterministic, synchronous, completely specified, and can be represented by state diagrams with transition assigned outputs, or by equivalent state tables. The machines are also assumed to be strongly connected such that there exists a path, not necessarily of length 1, from any state S_j to any other state S_k, where, $j \in \{1,2,..,m\}$, $k \in \{1,2,..,m\}$ and $j \neq k$.

3.2.2 Substitution Property (SP) Partitions

In order to find decompositions of sequential machines, we have to investigate the substitution property (SP) partitions. Hartmanis *et al.* [39] have made a systematic study on the properties of SP partitions of sequential machines. First, definitions of some basic terminologies will be clarified for SP partition theory.

Definition 1: A *partition* on the set of states of a machine M is the grouping of states into a number of disjoint subsets such that their union is the entire set of states. Let S be the entire set of states.

Definition 2: A *block* is a subset of states in a partition.

The partition π on S is defined as follows:

$$\pi = \{B_i \mid S \supseteq B_i \text{ and } B_i \cap B_j = \emptyset, \text{ for } i \neq j \text{ and } \bigcup_i B_i = S\}$$

where B_i and B_j are blocks.

Definition 3: The *zero partition*, π_0 is the partition into n blocks for a set of n states. The *unity partition* or *identity partition*, π_u is the partition into one block. π_0 and π_u are called *trivial partitions.*

Example 3.1: Consider a machine $M = \{S_0, S_1, S_2\}$, whose states are S_0, S_1, and S_2. Then all the possible partitions are,

$$\pi_0 = \pi_1 = \{\overline{S_0}; \overline{S_1}; \overline{S_2}\}$$
$$\pi_2 = \{\overline{S_0,S_1}; \overline{S_2}\}$$
$$\pi_3 = \{\overline{S_0,S_2}; \overline{S_1}\}$$
$$\pi_4 = \{\overline{S_0}; \overline{S_1,S_2}\}$$
$$\pi_u = \pi_5 = \{\overline{S_0,S_1,S_2}\}.$$

where a bar drawn over states denotes that those states are in the same block of a partition, and blocks are separated by semicolons. The partitions, $\pi_1 = \pi_0$ and $\pi_5 = \pi_u$ are trivial partitions.

Definition 4: The *sum* , π_s, of two partitions, π_a and π_b, is a partition, for which two states are in the same block of π_s if and only if they are elements of the set of the entire states, S, and they are contained in the same block of either π_a or π_b. The sum is written as: $\pi_s = \pi_a + \pi_b$ and computed taking the union of all the blocks of π_a and π_b.

Definition 5: The *product* , π_p, of two partitions, π_a and π_b, is a partition, for which two states are in the same block of π_p if and only if they are in the same block of both π_a and π_b. The product is written as: $\pi_p = \pi_a \cdot \pi_b$ and computed by intersecting the blocks of π_a and π_b.

Example 3.2: Using state partitions from Example 3.1,

$$\pi_2 + \pi_3 = \{\overline{S_0,S_1}\,;\,\overline{S_2}\} + \{\overline{S_0,S_2}\,;\,\overline{S_1}\}$$
$$= \{\overline{S_0,S_1,S_2}\}.$$

$$\pi_2 \cdot \pi_3 = \{\overline{S_0,S_1}\,;\,\overline{S_2}\} \cdot \{\overline{S_0,S_2}\,;\,\overline{S_1}\}$$
$$= \{\overline{S_0};\,\overline{S_1};\,\overline{S_2}\}$$

Definition 6: A partition π on the set of states of machine M is said to have the *substitution property* with respect to the machine M if, for any two states S_j and S_k belonging to the the same block of π and any input I_k, the next states $g(I_i, S_j)$ and $g(I_i, S_k)$ again belongs to a common block of π. A partition having the substitution property is called a *substitution property (SP) partition.*

Example 3.3: Consider a machine M_1 whose state table is given in Table 3.1.

Present State	Next State	
	x=0	x=1
S_0	S_3	S_2
S_1	S_5	S_2
S_2	S_4	S_1
S_3	S_1	S_4
S_4	S_0	S_3
S_5	S_2	S_3

Table 3.1 State table of machine M_1 (x is the input.).

The partition, $\pi_1 = \{\overline{S_0,S_1,S_2}; \overline{S_3,S_4,S_5}\}$ has the substitution property and is a SP partition, because all the next states for every pair in each block:

$g(0,S_0)=S_3 \quad g(0,S_1)=S_5 \quad g(0,S_2)=S_4$

$g(1,S_0)=S_1 \quad g(1,S_1)=S_2 \quad g(1,S_2)=S_2$

and

$g(0,S_3)=S_1 \quad g(0,S_4)=S_0 \quad g(0,S_5)=S_2$

$g(1,S_3)=S_4 \quad g(1,S_4)=S_3 \quad g(1,S_5)=S_3$

belong to the two blocks $\overline{S_0,S_1,S_2}$ and $\overline{S_3,S_4,S_5}$.

Theorem 1: If π_1 and π_2 are two SP partitions on the set of states of a sequential machine M, then $\pi_1.\pi_2$ and $\pi_1+\pi_2$ have the substitution property for M.

Proof: Let S_i and S_j be two states in a common block of partition $\pi_1.\pi_2$. Then the next states of S_i and S_j, for the same input symbol, must be contained in a common block of π_1, or contained in a common block of π_2, because of the substitution property of π_1 and π_2. This implies that the successors of S_i and S_j are in a common block partition $\pi_1.\pi_2$, thus conforming to the definition of substitution property.

Let S_i and S_j be two states contained in a common block of $\pi_1+\pi_2$. Then the next states of S_i and S_j, for the same input symbol, must be contained in a common block of π_1, or contained in a common block of π_2, or contained in two different blocks of π_1 and/or π_2 which are chain-connected. This implies that these next states are contained in a common block of $\pi_1 + \pi_2$, again conforming to the definition of substitution property. Q.E.D.

This theorem can be used to compute additional SP partitions from known ones without using the state table.

3.2.3 Computation of SP partitions

There are two steps to find all the SP partitions for a machine M [39]:

i) Compute the smallest SP partitions.

ii) Compute all the possible sums of SP partition pairs. These sums generate additional SP partitions.

To compute the smallest SP partitions (the smallest partitions with substitution property) for a given machine M, we can start by selecting or grouping temporarily any two distinct states, say S_1 and S_2. Then we have to identify the pairs of states S_{1k} and S_{2k} to which M will go from S_1 and S_2, respectively, when we apply the kth input, k = 1, ..., p. To this set of pairs, we add those pairs which can be identified by the transitive law; that is, if S_i and S_j are identified and so are S_j and S_k, we have to identify S_i, S_j, and S_k. We now repeat the process by looking up the new identifications induced by the new pairs. If after R steps, the (R+1)th step does not yield any new identifications, we have constructed a smallest SP partition π on S(states) which has the substitution property on M. Note that if there does not exist a nontrivial partition with the substitution property, then this process will stop after identifying all states of M. For a machine M with q states, we have to try q(q - 1)/2 distinct pairs of states to compute all the smallest SP partitions of M. Having obtained the smallest SP partitions $\pi_1, .., \pi_n$, we can generate all possible SP partitions by following Step ii).

Example 3.4: To illustrate how to compute all the SP partitions for a certain machine M, we consider the following state table of a Moore machine M.

Present State	Next State		Present Output
	x=0	x=1	Z
S_1	S_4	S_5	0
S_2	S_6	S_2	0
S_3	S_4	S_6	0
S_4	S_1	S_4	0
S_5	S_2	S_1	0
S_6	S_2	S_3	1

Table 3.2 State table of machine M.

We shall try the identification of all the 6(6 - 1)/2 = 15 distinct pairs of states, starting with the pair (S_1, S_2) and ending with the pair (S_5, S_6).

The identification of (S_1, S_2) implies the identification of (S_4, S_6) and (S_2, S_5). It should be noted that the implication graph technique [21] used for the determination of compatible states can also be used here. That is, starting with a pair of states(S_i, S_j), we can form a chain or graph of implication of pairs of states, called pairwise implications. For a q-state machine, there are altogether q(q-1)/2 pairs to be checked. If q is large we may have to start from several pairs of states for convenience and thus have several separate chains of graphs of pairwise implications. If a pair (S_i, S_j) has already appeared in a previous implication graph, then the pairwise implication graph starting from (S_i, S_j) need not be repeated. If in the pairwise implication graph under construction we come across a pair of states (S_i, S_j) that has already appeared in a previous implication graph, then the implication graph under construction should be connected to the previous implication graph through this pair (S_i, S_j) to form a larger graph. Thus, in the overall graph any pair of states like (S_i, S_j) should appear exactly once.

For the six state machine M of this example, we start from the state pair (S_1, S_2). The complete implication graph starting from (S_1, S_2) is shown in Fig 3.3, where for simplicity, the states S_1, S_2 are denoted by 1, 2,... .

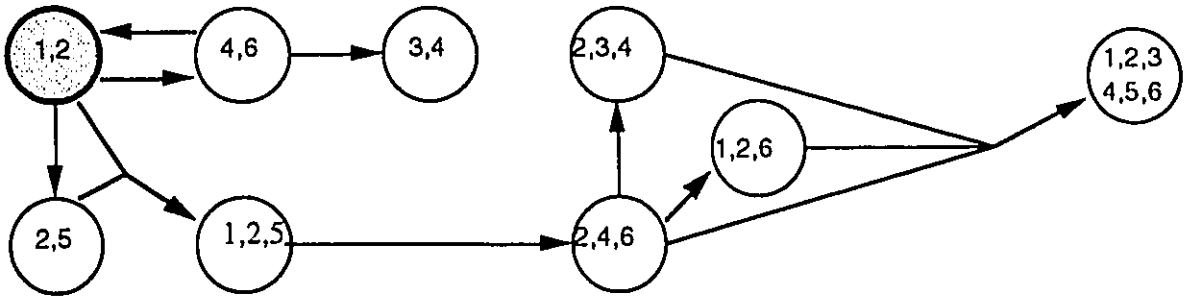


Fig.3.3 The implication graph starting with the identification of a pair of states (S_1, S_2).

Because of the transitive property, the identification of (S_1, S_2) and (S_2, S_5) implies the identification of (S_1, S_2, S_5), and the identification of (S_2, S_4, S_6), (S_1, S_2, S_6), and (S_1, S_2, S_6) implies the identification of (S_1, S_2, S_5). It can be seen from Fig. 3.3 that the implication graph ends up with the identification of all the states (S_1, .. , S_6).

Therefore, there does not exist a nontrivial SP partition which identifies a pair of states (S_1, S_2).

Next we consider the pair (S_1, S_3). The identification of (S_1, S_3) implies the identification of (S_5, S_6), and the identification of (S_5, S_6) implies the identification of (S_1, S_3) as shown in Fig.3.4.

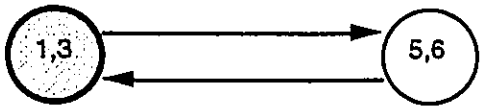


Fig 3.4 The implication graph starting with the identification of a pair of states (S_1, S_3).

Therefore, we have a nontrivial partition π_1 with the substitution property as follows.

$$\pi_1 = \{\overline{S_1 S_3}\ ;\ \overline{S_2};\ \overline{S_4};\ \overline{S_5 S_6}\}.$$

The implication graphs starting with the identification of (S_1, S_5) and (S_1, S_6), are shown in Fig. 3.5 and Fig.3.6, respectively.

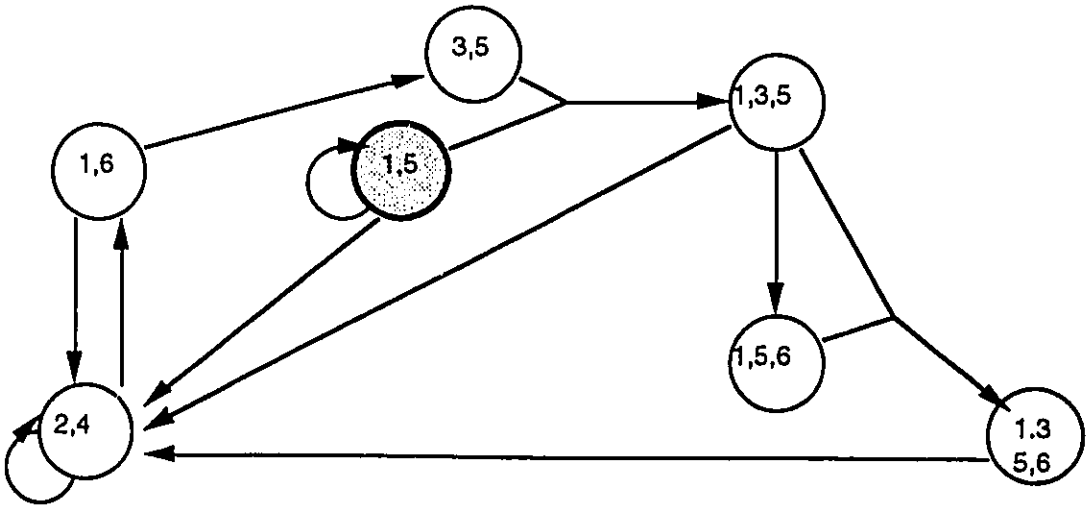


Fig.3.5 The implication graph starting with the identification of a pair of states (S_1, S_5).

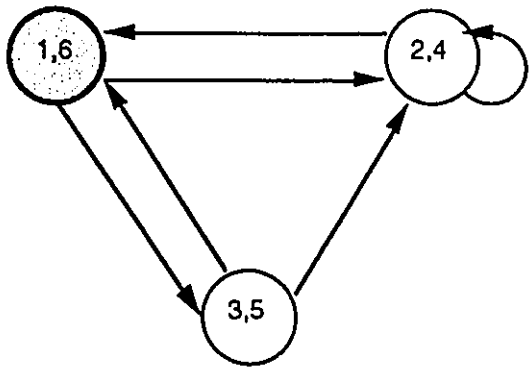


Fig 3.6 The implication graph starting with the identification of a pair of states (S_1, S_6).

They also result in partitions with the substitution property, π_2 and π_3, respectively, as shown below.

$$\pi_2 = \{\overline{S_1, S_3, S_5, S_6}; \overline{S_2, S_4}\}$$

$$\pi_3 = \{\overline{S_1, S_6}; \overline{S_2, S_4}; \overline{S_3, S_5}\}.$$

The implication graphs starting with the identification of other pairs of states all result in the trivial partition π_u (the unity partition) with a single block. These implication graphs are omitted.

Having obtained the SP partitions π_1, π_2 and π_3, we sum these SP partitions to investigate if there remains any additional SP partition. Because all sums of two SP partitions end up with the trivial partition π_u in this particular example, no new partition can be generated. Therefore $\{\pi_1, \pi_2, \pi_3\}$ is the complete set of partitions with the substitution property.

3.3 Decomposition of Machines

By decomposition of a sequential machine M, we mean the determination of a set of smaller machines which can be combined together to form a composite machine identical in behavior to the given machine M. In other words, decomposing a sequential machine M is to break down the overall specification of M (composite machine) into several smaller machines M_1 to M_n (component machines), and the component machines can be combined together to form a composite machine identical in behavior to the given machine. These smaller machines will each have, in general, fewer states than the original machine. We say this decomposition is nontrivial if it obeys the following definition.

Definition 7: The decomposition is said to be *nontrivial* if and only if every component machine has fewer states than the given machine M.

In order to decompose sequential machines, we have to find component machines and their interconnections. For finding component machines, we can use the SP partition theory discussed before. A composite machine can then be built by interconnecting the smaller machines either in series (cascade) form or in parallel form. The interconnection of component machines determine the classification of decompositions.

The decomposition can be classified under two general categories: serial decomposition and parallel decomposition as illustrated in Fig. 3.7.

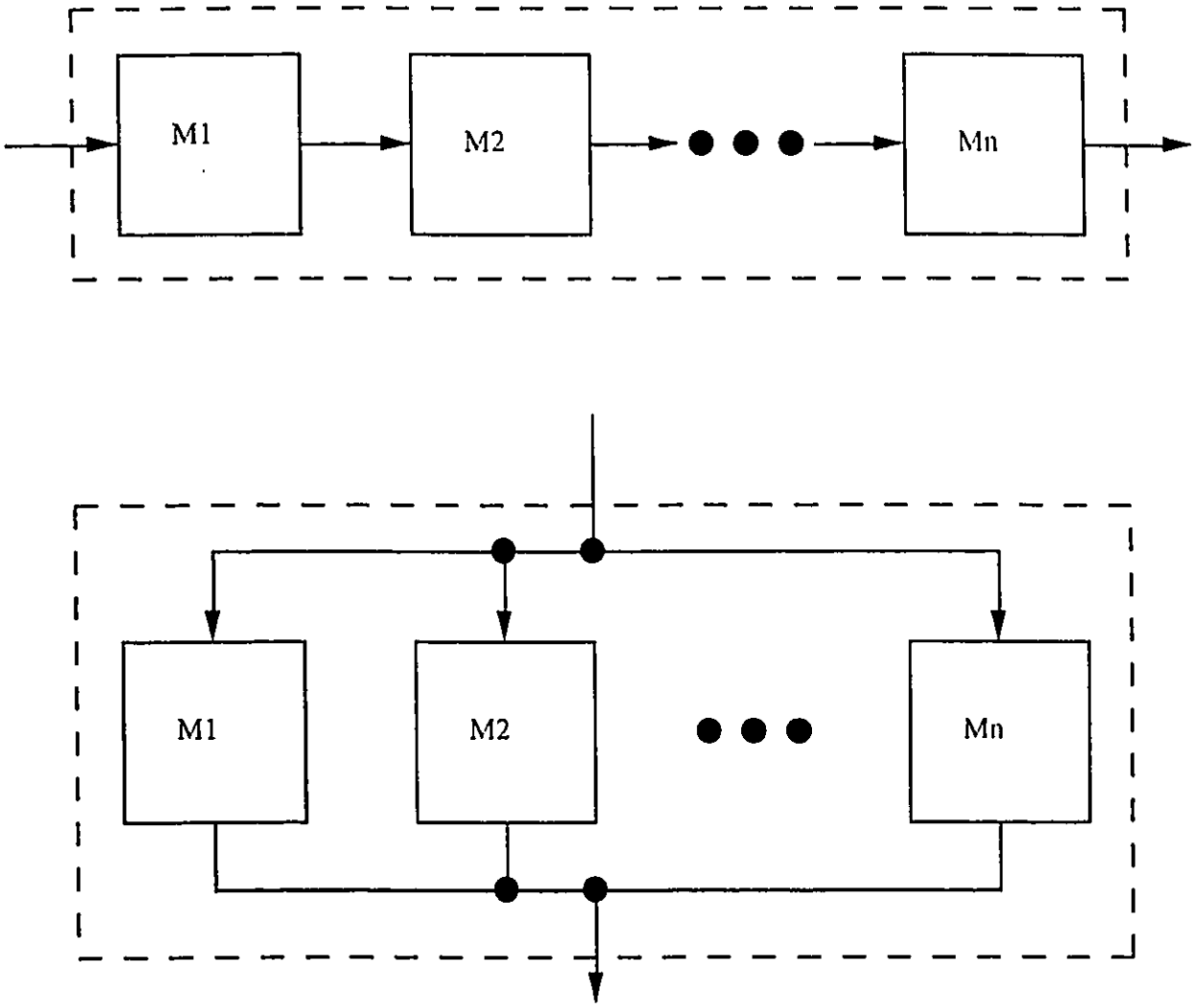


Fig.3.7 General schematic representation of serial decomposition and parallel decomposition.

3.3.1 Serial Decomposition

The serial decomposition of a machine M is the realization of M with a series (cascade) of component machines $M_1 \ldots M_n$. A general schematic representation of a serial decomposition of a machine M as shown in Fig. 3.7 consists of n number of components. In a serial connection which consist of two machines M_1 and M_2, the output of one machine, say M_1, forms the input of the other machine M_2. A schematic representation of series connection of two machines M_1 and M_2 is shown in Fig.3.8.

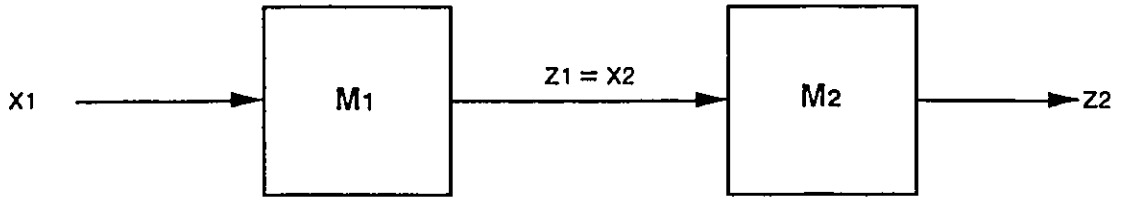


Fig. 3.8 Serial connection of two machines M_1 and M_2.

Let the component machines $M_1 = < I_1, S_1, O_1, f_1, g_1 >$ and $M_2 = < I_2, S_2, O_2, f_2, g_2 >$ be in a serial connection, where I_i, S_i, O_i, f_i, and g_i denote the input alphabet, the state alphabet, the output alphabet, the output mapping function, and the next state mapping function, respectively for machine M_i. Since M_1 and M_2 are connected in series (denoted as $M_1 \Rightarrow M_2$), $O_1 = I_2$. Then the composite machine M can be expressed as:

$$M = (M_1 \Rightarrow M_2) = < I_1, S_1 \times S_2, O_2, f, g >$$

where $f(S, I) = f(S_1 \times S_2, I_1) = f_2(S_2, f_1(S_1, I_1))$

and $g(S,I) = g(S_1 \times S_2, I_1) = (g_1(S_1, I_1), g_2(S_2, I_2))$

$= (g_1(S_1, I_1), g_2(S_2, f_1(S_1, I_1)))$.

To illustrate these points, let us consider the following example.

Example 3.5: The state tables of two machines M_1 and M_2 are shown in Tables 3.3 and 3.4, respectively.

Present State	Next State,Output=z_1	
	$x_1=0$	$x_1=1$
S_A	S_C,0	S_B,1
S_B	S_A,1	S_C,0
S_C	S_B,1	S_A,1

Table 3.3 State table of machine M_1.

Present State	Next State,Output=z_2	
	x_2=0	x_2=1
S_a	S_b,0	S_a,0
S_b	S_a,0	S_b,1

Table 3.4 State table of machine M_2.

The series connection of M_1 and M_2 results in a machine M whose state table, together with a secondary assignment, is shown in Table 3.5.

Present State		Next State,Output			
		x=0		x=1	
	$y_1y_2y_3$		$y_1y_2y_3$		$y_1y_2y_3$
S_{Aa}	0 0 0	S_{Cb}	1 1 1, 0	S_{Ba}	0 1 0, 0
S_{Ab}	0 0 1	S_{Ca}	1 1 0, 0	S_{Bb}	0 1 1, 1
S_{Ba}	0 1 0	S_{Aa}	0 0 0, 0	S_{Cb}	1 1 1, 0
S_{Bb}	0 1 1	S_{Ab}	0 0 1,1	S_{Ca}	1 1 0, 0
S_{Ca}	1 1 0	S_{Ba}	0 1 0, 0	S_{Aa}	0 0 0, 0
S_{Cb}	1 1 1	S_{Bb}	0 1 1, 1	S_{Ab}	0 0 1, 1

Table 3.5 State table of machine M (with assignment).

In Table 3.5, the states S_{Aa}, S_{Ab}, S_{Ba}, S_{Bb}, S_{Ca}, and S_{Cb} correspond to (S_A, S_a), (S_A, S_b), (S_B,S_a), (S_C, S_a), and (S_C,S_b), respectively. The next state and output entries are obtained according to the aforementioned relation. For instance,

$$f(S_{Aa}, 1) = f((S_A, S_a), 1)) = f_2(S_a, f_1(S_A, 1))$$
$$= f_2(S_A, 1) = 0,$$

and

$$g(S_{Aa}, 1) = (g_1(S_A, 1), g_2(S_a, f_1(S_A, 1))) = (S_B, g_2(S_a, 1))$$
$$= (S_B,S_a) = S_{Ba}.$$

The above example shows how a composite machine can be derived by series composition of two different component machines. Series decomposition is the reverse of series composition.

The machines $M_1 \Rightarrow M_2$ is said to be a series decomposition of machine M if and only if M can be realized by $M_1 \Rightarrow M_2$. A necessary and sufficient condition for the nontrivial series decomposition of a machine M into two submachines M_1 and M_2 is the existence of an SP partition on the set of states of M. For the machine M of Example 3.5, such a partition is

$$\pi = \{\overline{S_{Aa}, S_{Ab}}; \overline{S_{Ba}, S_{Bb}}; \overline{S_{Ca}, S_{Cb}}\}$$

From Table 3.5, it is readily seen that the next state variables $Y_1$and Y_2 are functions of the input variable x and the state variables y_1 and y_2 but are independent of y_3. That is, the next state variables Y_1 and Y_2 can be determined from Table 3.5 :

$$Y_1 = x'y_2' + xy_1'y_2$$
$$Y_2 = x'y_1' + x'y_2' + xy_1'.$$

As shown above, since the serial decomposition does not provide complete independence between the component machines, the states and outputs of M_2 (the tail machine) are dependent on those of M_1 (the head machine).

3.3.2 Parallel Decomposition

The parallel decomposition of machine M is realization of M consisting of a parallel connection of component machines $M_1, ..., M_n$. In parallel connection of two machine M_1 and M_2, the input to both the machines are the same,whereas the output of the composite machine is obtained as a function of the outputs of the machines M_1 and M_2. The parallel connection of two machines is shown in Fig. 3.9.

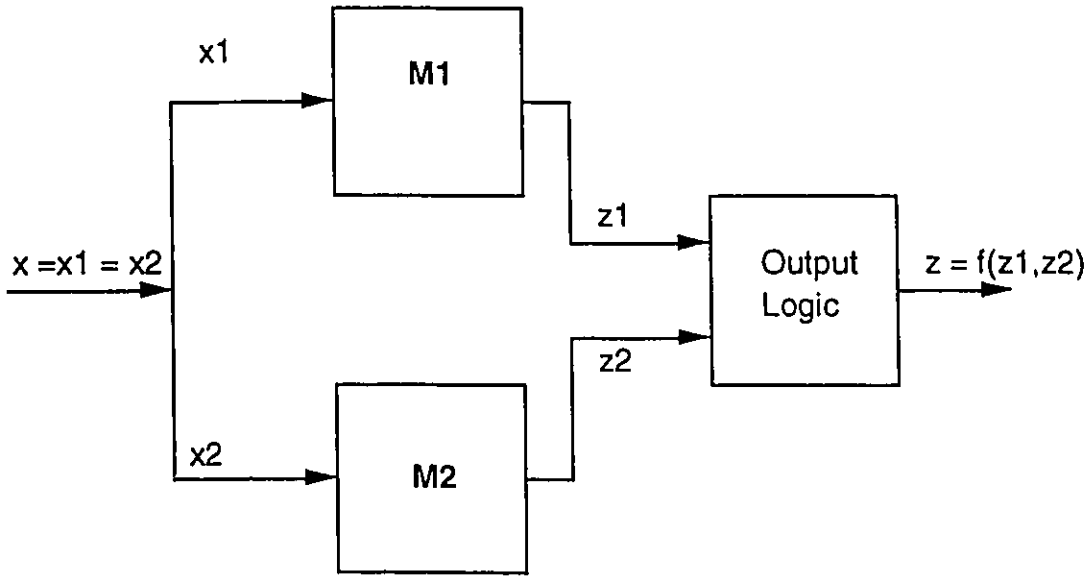


Fig.3.9 Parallel connection of two machines.

Let the component machines $M_1 = < I_1, S_1, O_1, f_1, g_1 >$ and $M_2 = < I_2, S_2, O_2, f_2, g_2 >$ be in a parallel connection. Then the composite machine M, denoted by $M_1 \parallel M_2$, can be expressed as:

$$M = (M_1 \parallel M_2) = < I, S, O, f, g >$$

where

$$I = I_1 = I_2$$

$$S = S_1 \times S_2$$

$$O = f'(O_1, O_2)$$

$$f(S, I) = f(S_1 \times S_2,\ I) = f'(f_1(S_1, I_1), f_2(S_2, I_2))$$

and $$g(S,I) = g(S_1 \times S_2,\ I) = (g_1(S_1, I_1), g_2(S_2, I_2))$$

As shown above, the outputs and the next states of component machines M_1 and M_2 are connected independently to realize the composite machine M. This property of the parallel connection may be applied to the random testing.

Parallel decomposition is the reverse of parallel connection or composition, The machine $M_1 \parallel M_2$ is a parallel decomposition of a machine M if and only if $M_1 \parallel M_2$ realizes M. The following theorem will specify the necessary conditions for parallel decomposition.

Theorem 2: A sequential machine M has a parallel decomposition of its behavior if and only if there exist two nontrivial partitions π_1 and π_2 with substitution property for M satisfying the condition:

$$\pi_1 \cdot \pi_2 = \pi_0$$

where π_0 is the zero partition.

Proof: The proof of Theorem 2 is omitted here and it is described in [39] and [44].

From the theorem, the necessary condition for the parallel decomposition of sequential machine M (the machine M is in fact decomposed into two independent machines) is the existence of two SP partitions π_1 and π_2 having a relationships of $\pi_1 \cdot \pi_2 = \pi_0$.

The following example illustrates parallel decomposition:

Example 3.6: Consider a pair of SP partitions,

$$\pi_1 = \{\overline{S_1 S_3}; \overline{S_2}; \overline{S_4}; \overline{S_5 S_6}\}$$

and

$$\pi_3 = \{\overline{S_1 S_6}; \overline{S_2 S_4}; \overline{S_3 S_5}\}$$

satisfied the the conditions of the theorem, because

$$\pi_1 \cdot \pi_3 = \pi_0$$

The SP partitions π_1 and π_3 are to be chosen for parallel decomposition of the machine M. Four new states, S_A to S_D, can be assigned to distinguish the four blocks of the partition π_1 and another three states, S_a to S_c, can be assigned to do three blocks of partitions (See Table 3.6 and Table 3.7).

Blocks	State
S_1, S_3	S_A
S_2	S_B
S_4	S_C
S_5, S_6	S_D

Table 3.6 A state assignment of π_1.

Blocks	State
S_1, S_6	S_a
S_2, S_4	S_b
S_3, S_5	S_c

Table 3.7 A state assignment of π_3.

It can be seen that the states S_a to S_c are independent of S_A to S_D, thus conforming to the theorem. Based on the new states we can decompose M into the parallel connection of two submachines M_1 and M_2, whose state tables are shown below.

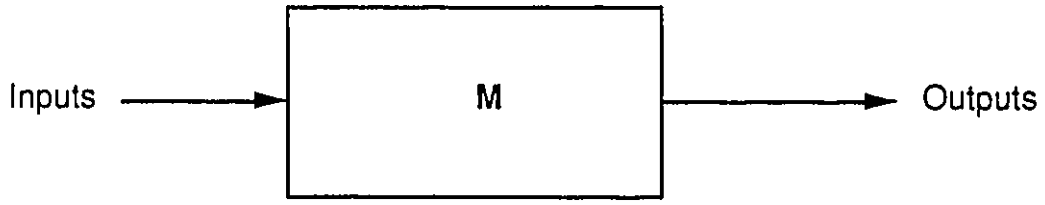
Present State	Next State		Present Output
	x=0	x=1	Z_0
S_A	S_C	S_D	0
S_B	S_D	S_B	0
S_C	S_A	S_C	0
S_D	S_B	S_A	1

Table 3.8 State table of machine M_1.

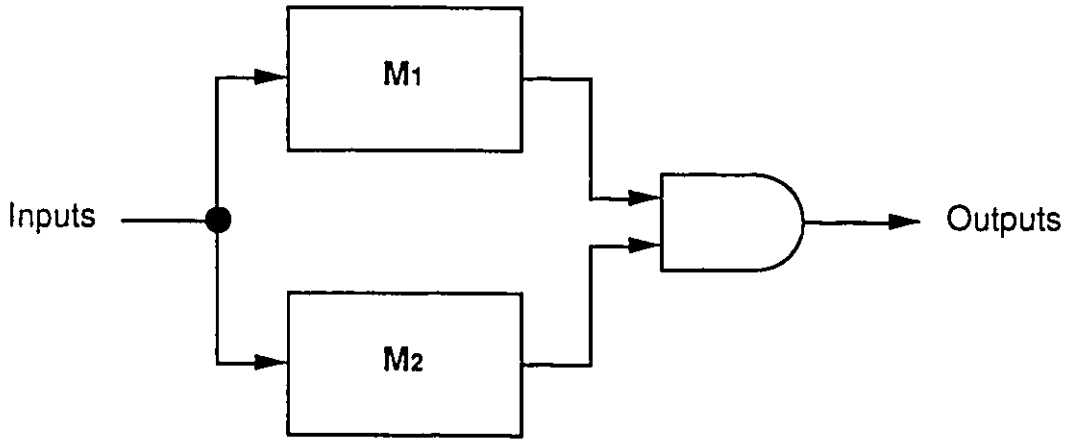
Present State	Next State		Present Output
	x=0	x=1	Z_1
S_a	S_b	S_c	1
S_b	S_a	S_b	0
S_c	S_b	S_a	0

Table 3.9 State table of machine M_2.

The output function of machine M is $Z = Z_0 . Z_1$. Note that in decomposing a machine M into two machines M_1 and M_2 in parallel, the size of the output alphabets O_1 and O_2 are not necessarily limited to 2, even if the size of the output alphabet O is 2. In other words, in general, M_1 and M_2 will each have more than one output variable. In general there do not exist two separate output variables $Z = Z_0 . Z_1$. In the above example, $Z = Z_0 . Z_1$ is a special case. Since M_2 has two state variables, in general one output variable is not sufficient. Finally the schematic representation of the Moore machine M with its parallel decomposition is shown in Fig. 3.10



(a) Machine M



(b) Interconnection of M1 and M2

Fig 3.10 Schematic representation of parallel decomposition.

3.4 Chapter Summary

In this chapter, the principles and types of decompositions of sequential circuits were discussed. The main objective is to optimally decompose a large-state machine (composite machine) into a set of fewer-state machines (submachines) in order to reduce interdependency between subsystems and to minimize the hardware. Important properties of decomposition theory were discussed in detail. Based on the decomposition theory, two types of decompositions, serial and parallel decomposition, are discussed with examples.

CHAPTER 4

MARKOV MODEL FOR FAULT DETECTION OF SEQUENTIAL CIRCUITS

4.1 Introduction

Markov models are recognized tools used in the analysis of stochastic processes whose past has no influence on the future if the present is specified. The use of Markov models to characterize the behavior of digital circuits in the presence of faults was discussed in the literature. Some of these models were introduced for the detection of intermittent faults, while others were proposed for the detection of permanent or nontransient faults by random patterns. For describing the behavior of circuits with intermittent faults, Breuer [12] introduced a discrete parameter Markov model of first order, whereas Kamal and Page [42] presented a discrete parameter Markov model of zero order, which was subsequently used by Savir [61] and Koren and Kohavi [46]. Su *et al.*[70] proposed a continuous parameter Markov model with two states to characterize the behavior of digital systems in the presence of intermittent faults. The continuous parameter Markov model is a generalization of the discrete parameter model and is believed to be more realistic and convenient to use.

A fault in a digital circuit need not cause an immediate error in the circuit output; there is typically a delay between the occurrence of a fault and the first error to appear in the circuit output. This delay is the error latency of the fault. It depends on the circuit, the fault, and the input pattern applied to the circuit. Shedletsky and McCluskey [67] analyzed random testing of digital circuits using the error-latency model (ELM). This model proved very useful for the analysis of random testing procedures in sequential circuits.

In this thesis, as a tool of fault detection and analysis in sequential machines, a tri-state continuous [29] Markov model is utilized. The continuous Markov model of faulty sequential machines is defined in Section 4.2. Section 4.3 derives differential equations for the model. In Sections 4.4 and 4.5, Markov parameters and state probabilities are derived. In Section 4.6, testing time calculations using both the closed form solutions and the numerical approaches are described.

4.2 The Continuous Markov Model of Faulty Sequential Circuits

The sequential circuits considered here are deterministic and synchronous, and can be represented by state diagrams with transition assigned outputs, or by equivalent state tables. They are also assumed to be strongly connected such that there exists a path, not necessarily of length 1, from any state to any other state.

In order to describe the behavior of a sequential circuit in the presence of a fault while subjected to random inputs, a continuous parameter Markov model with three states is defined. The three states are

state 0 : the fault exists, but causes no error output and error state transition;

state 1 : the fault causes an error output in the output terminals;

state 2 : the fault causes only error state transition, but no error output results in the output terminals.

To represent the interaction among the three states 0, 1, and 2 for the infinitesimal time step Δt in the circuit, we will define the different state transition probabilities.

Let $Pr[(S=j, T=t+\Delta t)|(S=i, T=t)]$ denote the probability of the circuit of staying in state i at time t, but going to state j at time $t+\Delta t$. Then we can define transition probabilities:

$\lambda_0\Delta t$: $Pr[(S=1, T=t+\Delta t)|(S=0, T=t)]$. (4.1)

$\lambda_1\Delta t$: $Pr[(S=2, T=t+\Delta t)|(S=1, T=t)]$. (4.2)

$\lambda_2\Delta t$: $Pr[(S=0, T=t+\Delta t)|(S=2, T=t)]$. (4.3)

$\mu_0\Delta t$: $Pr[(S=0, T=t+\Delta t)|(S=1, T=t)]$. (4.4)

$\mu_1\Delta t$: $Pr[(S=1, T=t+\Delta t)|(S=2, T=t)]$. (4.5)

$\mu_2\Delta t$: $Pr[(S=2, T=t+\Delta t)|(S=0, T=t)]$. (4.6)

$1-\lambda_0\Delta t-\mu_2\Delta t$: $Pr[(S=0, T=t+\Delta t)|(S=0, T=t)]$. (4.7)

$1-\lambda_1\Delta t-\mu_0\Delta t$: $Pr[(S=1, T=t+\Delta t)|(S=1, T=t)]$. (4.8)

$1-\lambda_2\Delta t-\mu_1\Delta t$: $Pr[(S=2, T=t+\Delta t)|(S=2, T=t)]$. (4.9)

Finally, the tri-state continuous parameter Markov model is given in Fig. 4.1.

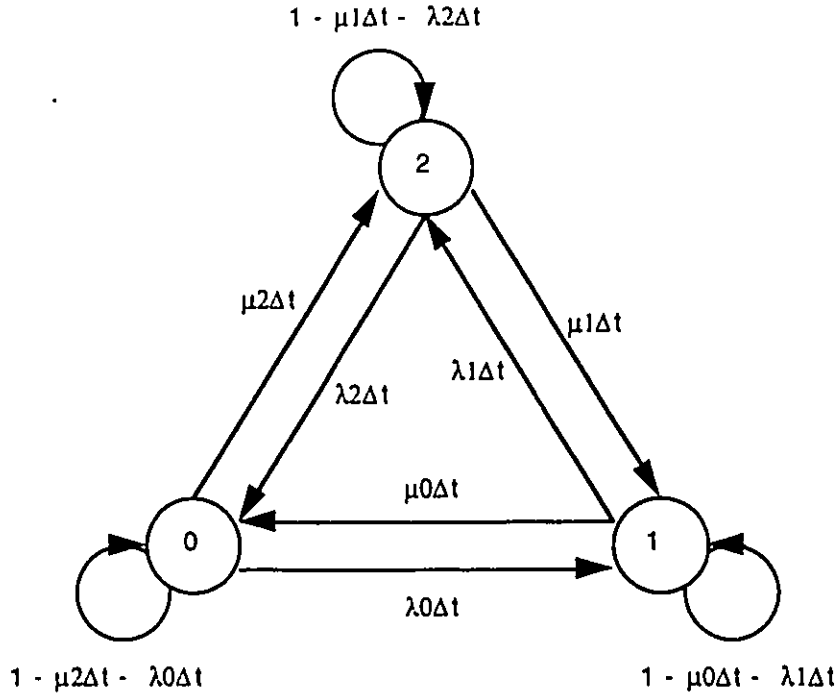


Fig. 4.1 The tri-state continuous parameter Markov model.

4.3 Derivation of Differential Equations of the Model

Using the relationship between conditional probability and joint probability, $P(A|B)=\frac{P(A,B)}{P(B)}$, the Equations (4.7), (4.4), and (4.3) can be written as:

$$Pr[(S=0,\ T=t+\Delta t)|(S=0,\ T=t)]=\frac{Pr[(S=0,\ T=t+\Delta t)\text{ and }(S=0,\ T=t)]}{Pr[(S=0,\ T=t)]}, \qquad (4.10)$$

$$Pr[(S=0,T=t+\Delta t)|(S=1,\ T=t)]=\frac{Pr[(S=0,\ T=t+\Delta t)\text{ and }(S=1,\ T=t)]}{Pr[(S=1,\ T=t)]}, \qquad (4.11)$$

$$Pr[(S=0,T=t+\Delta t)|(S=2,T=t)]=\frac{Pr[(S=0,\ T=t+\Delta t)\text{ and }(S=2,\ T=t)]}{Pr[(S=2,\ T=t)]}. \qquad (4.12).$$

These equations give the probability of staying in state 0 at T=t+Δt given that the circuit is in state 0, state 1, or state 2 at the time T=t. We define $P_i(t) = Pr[(S=i, T=t)]$, where i = 0,1,2, as the probability of the circuit being in state i at time T=t.

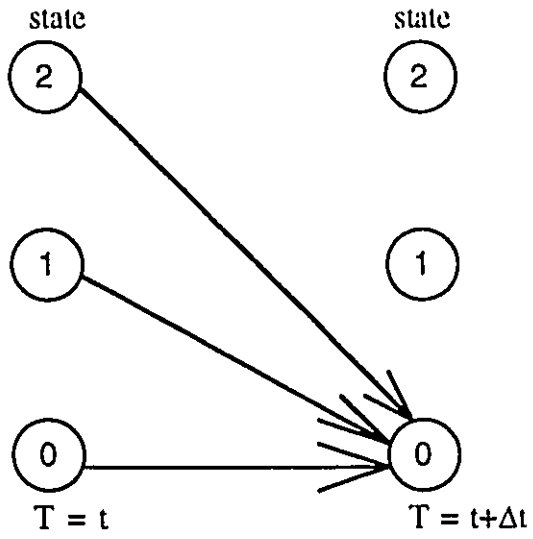


Fig. 4.2 $Pr[(S=0, T=t+\Delta t)] = P_0(t+\Delta t)$

$$= \sum_{i=0}^{2} Pr[(S=0, T=t+\Delta t) \text{ and } (S=i, T=t)]$$

Because Equations (4.10), (4.11), and (4.12) are mutually exclusive (as illustrated in Fig 4.2), the probability that the circuit is in state 0 is expressed as;

$$Pr[(S=0, T=t+\Delta t)] = P_0(t+\Delta t)$$

$$= \sum_{i=0}^{2} Pr[(S=0, T=t+\Delta t) \text{ and } (S=i, T=t)]$$

$$= Pr[(S=0, T=t)]\ Pr[(S=0, T=t+\Delta t) | (S=0, T=t)]$$
$$+ Pr[(S=1, T=t)]\ Pr[(S=0, T=t+\Delta t) | (S=0, T=t)]$$
$$+ Pr[(S=2, T=t)]\ Pr[(S=0, T=t+\Delta t) | (S=0, T=t)]$$
$$= P_0(t)\ [\ 1-\lambda_0\Delta t-\mu_2\Delta t\ ] + P_1(t)\mu_0\Delta t\ + P_2(t)\ \lambda_2\Delta t$$

Dividing by Δt both terms and taking a limit on Δt

$$\lim_{\Delta t \to 0} \frac{P_0(t+\Delta t) - P_0(t)}{\Delta t} = P_1(t)\mu_0 + P_2(t)\lambda_2 - P_0(t)[\ \lambda_0+\mu_2\ ]$$

Then we can get a differential equation,

$$\frac{dP_0(t)}{dt} = P_1(t)\mu_0 + P_2(t)\lambda_2 - P_0(t)(\mu_2 + \lambda_0) \qquad (4.13)$$

Using the same method as above, the probability of the circuit going to state 1 and state 2 at time T=t + Δt, two more differential equations can be derived as follows:

$$\frac{dP_1(t)}{dt} = P_0(t)\lambda_0 + P_2(t)\mu_1 - P_1(t)(\mu_0 + \lambda_1) \qquad (4.14)$$

$$\frac{dP_2(t)}{dt} = P_0(t)\mu_2 + P_1(t)\lambda_1 - P_2(t)(\lambda_2+\mu_1). \qquad (4.15)$$

4.4 Markov Parameters

In order to simulate the transition behavior of the circuit with respect to the model, it should be mentioned that we have nine different counts to store the information regarding the circuit after a large number of the input patterns is applied. Let C_{ab} = N(circuit going from state a to state b); a = 0,1,2; b = 0,1,2, where N denotes the total occurrence number or count .

All counts are cleared to zero initially. By using a random number generator we can generate an input value in the interval [0,1], and then apply it to the circuit with a certain constant probability. However, instead of physical inputs to the circuit, we may also simulate the stimulation with respect to the fault-free state table of the circuit and its corresponding faulty table. On comparing the status of these two state tables, on each application of a random pattern, we will have any of the following three different cases in the compared result.

Case 1 : Neither an output nor a state transition is different, and the circuit is in state 0.

Case 2 : Output is different, and the circuit is in state 1.

Case 3 : Only a state transition is different, but an output is not different, and the circuit is in state 2.

The circuit is evidently in state 0 initially, since at the beginning of test no differences ever occurred. After the first random pattern is applied to the circuit, we compare the outcomes with respect to the two different tables, and then decide upon which one of the aforementioned three cases occurs; the state of the circuit with respect to the current input pattern is thus determined.

Assume that on application of the first input pattern we have determined the circuit to be in state i, $0 \le i \le 2$; we increment C_{0i}, i.e., $C_{0i} = C_{0i} + 1$. Next we apply the second random pattern to the circuit, and determine which state it is in. If it is now in state j, $0 \le j \le 2$, then $C_{ij} = C_{ij} + 1$. This process is repeated until a large number of input patterns are applied to the circuit.

Then by Equations (4.1) to (4.6), the Markov parameters are calculated as follows,

$$\lambda_0 = \frac{C_{01}}{\text{Total number of random inputs per unit time}}$$

$$\lambda_1 = \frac{C_{12}}{\text{Total number of random inputs per unit time}}$$

$$\lambda_2 = \frac{C_{20}}{\text{Total number of random inputs per unit time}}$$

$$\mu_0 = \frac{C_{10}}{\text{Total number of random inputs per unit time}}$$

$$\mu_1 = \frac{C_{21}}{\text{Total number of random inputs per unit time}}$$

$$\mu_2 = \frac{C_{02}}{\text{Total number of random inputs per unit time}}.$$

A C language program was written to output the fault-free and faulty state tables, the count values C_{ij}s and the Markov parameters μ_is and λ_is for probability p(1) ranging from 0.0 to 0.9 in steps of 0.1. Note that P(1) is the probability of the input being 1.

Example: As an illustration, consider the synchronous sequential circuit as shown below in Fig. 4.3(a), with its corresponding state table M (see Fig. 4.3(b)).

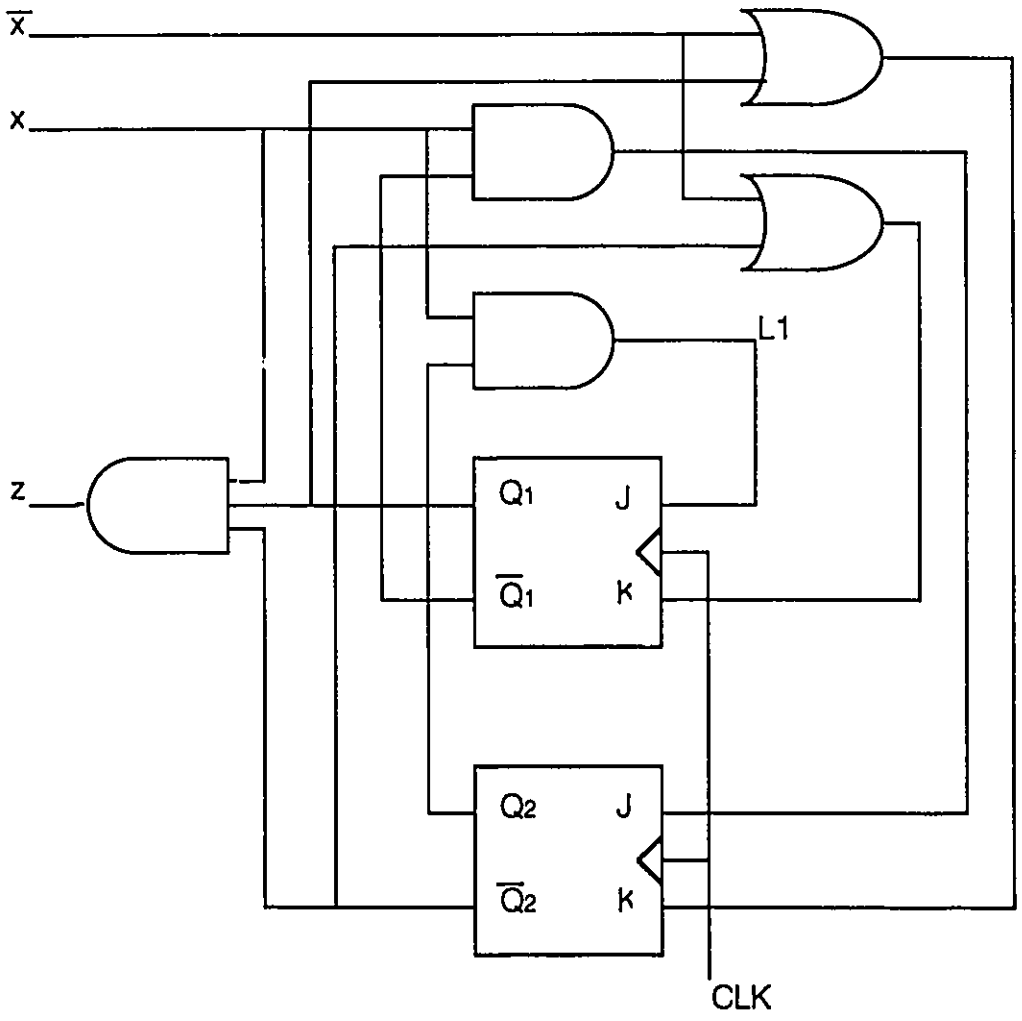


Fig.4.3(a) Example of a sequential circuit.

Present State	Next State	
	x=0	x=1
$S_0 = 00$	S_0,0	S_1,0
$S_1 = 01$	S_0,0	S_3,0
$S_2 = 10$	S_0,0	S_0,1
$S_3 = 11$	S_0,0	S_2,0

Fig.4.3(b) Fault-free state table M.

Present State	Next State	
	x=0	x=1
$S_0 = 00$	S_2,0	S_3,0
$S_1 = 01$	S_2,0	S_3,0
$S_2 = 10$	S_0,0	S_0,1
$S_3 = 11$	S_0,0	S_2,0

Fig.4.3(c) Faulty-state table M'.

For the circuit of Fig. 4.3(a), if we assume a stuck-at-1 fault in line L1 (L1/1), then the state table M' of the circuit corresponding to this fault can be obtained as shown in Fig. 4.3(c).

We now simulate the fault-free state table M and its faulty version M' on DEC 3100 workstation running a RISC based ULTRIX - 32 (UNIX) system, and apply random patterns generated by a random number generator to these simulated state tables. After 10,000 input patterns are applied, with different probabilities of a 1 input, p(1), we obtain values of C_{ij}, as shown in Table 4.1. For each value of p(1), we have chosen a different initial value (seed) for the random number generator.

P(1)	C_{00}	C_{01}	C_{02}	C_{10}	C_{11}	C_{12}	C_{20}	C_{21}	C_{22}
0	0	0	5000	0	0	0	5000	0	0
0.1	51	0	4521	50	1	408	4470	459	40
0.2	186	0	4089	161	5	658	3928	819	154
0.3	380	0	3675	315	34	775	3359	1090	372
0.4	604	0	3285	487	87	867	2797	1354	519
0.5	858	0	2913	671	187	861	2241	1533	736
0.6	*1030*	*0*	*2477*	*801*	*361*	*986*	*1676*	*1787*	*899*
0.7	1093	0	2147	815	553	1104	1331	1920	1037
0.8	1205	0	1659	867	937	1289	792	2165	1075
0.9	1365	0	1217	840	1255	1563	377	2403	980

Table 4.1 Transition counts.

The transition frequency count is a basic element of parameters inference. If we generate 1,000 input patterns per ms, we have the rates (parameters) of our model derived as follows (based on the results of P(1) = 0.6 in Table 4.1).

$\lambda_0 = 0$ $\lambda_1 = 0.986$ $\lambda_2 = 1.676$

$\mu_0 = 0.801$ $\mu_1 = 1.787$ $\mu_2 = 2.477.$

4.5 The State Probabilities

Using the initial conditions $P_0(0) = 1$, $P_1(0) = P_2(0) = 0$, we can solve the three differential equations, (4.13), (4.14) , and (4.15), which was derived in Section 4.3. That is:

$$P_0(t) = \frac{D_0}{B}u(t) + e^{-(A/2)t}\left[\frac{B-D_0}{B}\cos\sqrt{B - \frac{A^2}{4}}\,t + \frac{2BC_0 - AB - AD_0}{2B\sqrt{B - \frac{A^2}{4}}}\sin\sqrt{B - \frac{A^2}{4}}\,t\right]$$

$$P_1(t) = \frac{D_1}{B}u(t) + e^{-(A/2)t}\left[\frac{-D_1}{B}\cos\sqrt{B - \frac{A^2}{4}}\,t + \frac{2BC_1 - AD_1}{2B\sqrt{B - \frac{A^2}{4}}}\sin\sqrt{B - \frac{A^2}{4}}\,t\right]$$

$$P_2(t) = \frac{D_2}{B}u(t) + e^{-(A/2)t}\left[\frac{-D_2}{B}\cos\sqrt{B - \frac{A^2}{4}}\,t + \frac{2BC_2 - AD_2}{2B\sqrt{B - \frac{A^2}{4}}}\sin\sqrt{B - \frac{A^2}{4}}\,t\right]$$

where
$$A = \mu_0 + \mu_1 + \mu_2 + \lambda_0 + \lambda_1 + \lambda_2,$$
$$B = \mu_0\mu_1 + \mu_1\mu_2 + \mu_0\mu_2 + \lambda_0\lambda_1 + \lambda_1\lambda_2 + \lambda_0\lambda_2 + \lambda_0\mu_1 + \lambda_2\mu_0 + \mu_2\lambda_1,$$
$$C_0 = \mu_0 + \mu_1 + \lambda_1 + \lambda_2,$$
$$D_0 = \mu_0\lambda_2 + \mu_0\mu_1 + \lambda_1\lambda_2,$$
$$C_1 = \lambda_0,$$
$$D_1 = \lambda_0\lambda_2 + \lambda_0\mu_1 + \mu_1\mu_2,$$

$$C_2 = \mu_2,$$
$$D_2 = \mu_0 \mu_2 + \mu_2 \lambda_1 + \lambda_0 \lambda_1,$$

and,

$$u(t) = \begin{cases} 0 & \text{for } t < 0 \\ 1 & \text{for } t \geq 0 \end{cases}$$

With respect to the term $(B - \frac{A^2}{4})$ in the above solutions, we may have three different cases.

Case 1: $(B - \frac{A^2}{4}) = 0$

In this case, the expressions for $P_0(t)$, $P_1(t)$, and $P_2(t)$ are given as:

$$P_0(t) = \frac{D_0}{B}u(t) + e^{-(A/2)t}\left[\frac{B-D_0}{B} + \frac{2BC_0 - AB - AD_0}{2B}t\right]$$

$$P_1(t) = \frac{D_1}{B}u(t) + e^{-(A/2)t}\left[\frac{-D_1}{B} + \frac{2BC_1 - AD_1}{2B}t\right]$$

$$P_2(t) = \frac{D_2}{B}u(t) + e^{-(A/2)t}\left[\frac{-D_2}{B} + \frac{2BC_2 - AD_2}{2B}t\right]$$

Thus, for $(B - \frac{A^2}{4}) = 0$, the probabilities of the three states converge to their steady state values in a non-oscillatory manner.

Case 2: $(B - \frac{A^2}{4}) > 0$

In this case, damping exists and $P_0(t)$, $P_1(t)$, and $P_2(t)$ decay exponentially in an oscillatory manner.

Case 3: $(B - \frac{A^2}{4}) < 0$

In this case, the expressions for state probabilities can be derived as:

$$P_0(t) = \frac{D_0}{B}u(t) + K_1e^{-S_1t} + K_2e^{-S_2t}$$

$$P_1(t) = \frac{D_1}{B}u(t) + K_1'e^{-S_1t} + K_2'e^{-S_2t}$$

$$P_2(t) = \frac{D_2}{B}u(t) + K_1''e^{-S_1t} + K_2''e^{-S_2t}$$

where,

$$S_1 = \frac{A}{2} - \sqrt{\frac{A^2}{4} - B}\ ,\ S_2 = \frac{A}{2} + \sqrt{\frac{A^2}{4} - B}\ ,$$

$$K_1 = \frac{C_0 - S_1 - \frac{D_0}{S_1}}{2\sqrt{\frac{A^2}{4} - B}}\ ,\ K_2 = \frac{C_0 - S_2 - \frac{D_0}{S_2}}{-2\sqrt{\frac{A^2}{4} - B}}\ ,$$

$$K_1' = \frac{C_1 - \frac{D_1}{S_1}}{2\sqrt{\frac{A^2}{4} - B}}\ ,\ K_2' = \frac{C_1 - \frac{D_1}{S_2}}{-2\sqrt{\frac{A^2}{4} - B}}\ ,$$

$$K_1'' = \frac{C_2 - \frac{D_2}{S_1}}{2\sqrt{\frac{A^2}{4} - B}}\ ,\ K_2'' = \frac{C_2 - \frac{D_2}{S_2}}{-2\sqrt{\frac{A^2}{4} - B}}\ .$$

The probabilities of the three states converge to their steady state values in the same nonoscillatory manner as in case 1, but at a much slower rate.

From the above analysis, we can see that in order for a fault in a circuit to be detected quickly with a constant probability, it is desirable that the parameters or rates of the model should satisfy the condition specified by $(B - \frac{A^2}{4}) \geq 0$. In case this condition is not met, we have to modify our input signal probability assignment for the circuit to change the parameters of the model in a way that will increase the convergence speed of the state probabilities so that the fault may be detected as fast as possible.

To further analyze our model, we may develop transition probabilities among the states. Let $P_{ij}(t)$ denotes the probability of going from state i at time t_0 to state j at time t_0+t. We can hence write the following sets of differential equations in terms of the state transition probabilities.

$$\frac{dP_{00}(t)}{dt} = P_{01}(t)\mu_0 + P_{02}(t)\lambda_2 - P_{00}(t)(\mu_2 + \lambda_0)$$

$$\frac{dP_{01}(t)}{dt} = P_{00}(t)\mu_0 + P_{02}(t)\mu_1 - P_{01}(t)(\mu_0 + \lambda_1)$$

$$\frac{dP_{02}(t)}{dt} = P_{00}(t)\mu_2 + P_{01}(t)\lambda_1 - P_{02}(t)(\lambda_2 + \mu_1)$$

with $P_{00}(0) = 1$, $P_{01}(0) = P_{02}(0) = 0$ and $P_{00}(t) + P_{01}(t) + P_{02}(t) = 1$.

$$\frac{dP_{10}(t)}{dt} = P_{11}(t)\mu_0 + P_{12}(t)\lambda_2 - P_{10}(t)(\mu_2 + \lambda_0)$$

$$\frac{dP_{11}(t)}{dt} = P_{10}(t)\lambda_0 + P_{12}(t)\mu_1 - P_{11}(t)(\mu_0 + \lambda_1)$$

$$\frac{dP_{12}(t)}{dt} = P_{10}(t)\mu_2 + P_{11}(t)\lambda_1 - P_{12}(t)(\lambda_2 + \mu_1)$$

with $P_{11}(0) = 1$, $P_{10}(0) = P_{12}(0) = 0$ and $P_{10}(t) + P_{11}(t) + P_{12}(t) = 1$.

$$\frac{dP_{20}(t)}{dt} = P_{21}(t)\mu_0 + P_{22}(t)\lambda_2 - P_{20}(t)(\mu_2 + \lambda_0)$$

$$\frac{dP_{21}(t)}{dt} = P_{20}(t)\lambda_0 + P_{22}(t)\mu_1 - P_{21}(t)(\mu_0 + \lambda_1)$$

$$\frac{dP_{22}(t)}{dt} = P_{20}(t)\mu_2 + P_{21}(t)\lambda_1 - P_{22}(t)(\lambda_2 + \mu_1)$$

with $P_{22}(0) = 1$, $P_{20}(0) = P_{21}(0) = 0$ and $P_{20}(t) + P_{21}(t) + P_{22}(t) = 1$.

The above sets of equations are isomorphic with Equations (4.13) to (4.15) defined above. The solutions corresponding to the differential equations are given as follows, with $P_{00}(t) = P_0(t)$, $P_{01}(t) = P_1(t)$, and $P_{02}(t) = P_2(t)$, with all parameters being the same as those in $P_0(t)$, $P_1(t)$, and $P_2(t)$.

$P_{10}(t) = P_2(t)$, with $C_2' = \mu_0$, $D_2' = \lambda_1\lambda_2 + \lambda_2\mu_0 + \mu_0\mu_1$;

$P_{11}(t) = P_0(t)$, with $C_0' = \lambda_0 + \lambda_2 + \mu_1 + \mu_2$, $D_0' = \lambda_0\lambda_2 + \lambda_0\mu_1 + \mu_1\mu_2$;

$P_{12}(t) = P_1(t)$, with $C_1' = \lambda_1$, $D_1' = \lambda_0\lambda_1 + \lambda_1\mu_2 + \mu_0\mu_2$.

$P_{20}(t) = P_1(t)$, with $C_1'' = \lambda_2$, $D_1'' = \lambda_1\lambda_2 + \lambda_2\mu_0 + \mu_0\mu_1$;

$P_{21}(t) = P_2(t)$, with $C_2'' = \mu_1$, $D_2'' = \lambda_0\lambda_2 + \lambda_0\mu_1 + \mu_1\mu_2$;

$P_{22}(t) = P_0(t)$, with $C_0'' = \lambda_0 + \lambda_1 + \mu_0 + \mu_2$, $D_0'' = \lambda_0\lambda_1 + \lambda_1\mu_2 + \mu_0\mu_2$.

As in the case of state probabilities, the condition for rapid convergence to steady state values for the state transition probabilities depends on $(B - \frac{\Lambda^2}{4})$. If $(B - \frac{\Lambda^2}{4}) \geq 0$, all of the state transition probabilities converge quickly to their steady state values. The result about $P_{00}(t) = P_0(t)$, $P_{01}(t) = P_1(t)$, and $P_{02}(t) = P_2(t)$ appears to be reasonable outcome in view of the assumption that $P_0(0) = 1$.

4.6 Testing Time Calculations

Faults in a digital circuit are detected by applying test patterns to the primary inputs of the circuit, and observing the output response with respect to these input patterns. In the strategy of random testing, a large number of input patterns are generated randomly. However, if a fault is not sensitive to these generated test patterns, a wrong conclusion may be drawn regarding the existence of the fault. One way to minimize the probability of such a wrong conclusion is to apply a large number of test patterns to the circuit until either the fault is detected, or the confidence about the circuit being error-free is larger than or equal to some prespecified value. In order to increase our confidence in the testing procedure, we must minimize the probability that a fault exists, but is not detected.

During computer simulation of the Markov model, a critical problem was faced which has been solved by using a numerical method to solve the nonlinear equations. This approach may be considered more accurate than using an approximate method on the closed form equation.

Assume that the test patterns are applied to the circuit under test (CUT) continuously from time t_0 to time t_0+s, and the testing is terminated prior to t_0+s, if the fault is detected. We determine the maximum testing time s(max) so that the probability of a wrong conclusion is smaller than or equal to some precalculated value α, that is,

$P(\text{fault exists, but is not detected during the interval } [t_0,t_0+s] \mid \text{fault exists}) \leq \alpha.$

This probability may be decomposed, and expressed as:

$P(\text{fault exists, but is not detected during the interval } [t_0,t_0+s] \mid \text{fault exists}) = P(X \cap Y_1 \cap Z) + P(X \cap Y_2 \cap Z) \leq \alpha,$

where X, Y_1, Y_2, Z are the events defined as

- X: fault exists in the circuit,
- Y_1: fault causes no error output and error state transition at time t_0,
- Y_2: fault causes only error state transition at time t_0,
- Z: fault causes no error output from time t_0 to time t_0+s.

We then have:

$$P(X \cap Y_1 \cap Z) + P(X \cap Y_2 \cap Z) = P(Z|X \cap Y_1)\, P(Y_1|X)P(X) + P(Z|X \cap Y_2)P(Y_2|X)P(X)$$

With reference to our model, we can now compute the aforesaid probabilities as follows:

4.6.1 Testing Time Calculation in Case $(B - \frac{A^2}{4}) > 0$: Closed Form Approach [29]

$$P(Z|X \cap Y_1) = 1 - P_{01}(s)$$

$$= 1 - \left[\frac{D_1}{B} u(s) + e^{-(A/2)s} \left\{ \frac{-D_1}{B} \cos\sqrt{B-\frac{A^2}{4}}\, s + \frac{2BC_1 - AD_1}{2B\sqrt{B - \frac{A^2}{4}}} \sin\sqrt{B-\frac{A^2}{4}}\, s \right\} \right]$$

$$= \frac{B - D_1}{B} u(s) - e^{-(A/2)s} \left[\frac{-D_1}{B} \cos\sqrt{B-\frac{A^2}{4}}\, s + \frac{2BC_1 - A D_1}{2B\sqrt{B - \frac{A^2}{4}}} \sin\sqrt{B-\frac{A^2}{4}}\, s \right]$$

$$P(Z|X\cap Y_2) = 1 - P_{21}(s)$$

$$= 1 - \left[\frac{D_2''}{B} u(s) + e^{-(A/2)s} \left\{ \frac{-D_2''}{B} \cos\sqrt{B-\frac{A^2}{4}}\, s + \frac{2BC_2'' - AD_2''}{2B\sqrt{B - \frac{A^2}{4}}} \sin\sqrt{B-\frac{A^2}{4}}\, s \right\} \right]$$

$$= \frac{B - D_2''}{B} u(s) - e^{-(A/2)s} \left[\frac{-D_2''}{B} \cos\sqrt{B-\frac{A^2}{4}}\, s + \frac{2BC_2'' - AD_2''}{2B\sqrt{B - \frac{A^2}{4}}} \sin\sqrt{B-\frac{A^2}{4}}\, s \right]$$

To determine $P(Y_1|X)$ and $P(Y_2|X)$, assume that the fault existed for a long time prior to time t_0 so that the circuit is in a stable condition, and the steady state probabilities can be used. Hence,

$$P(Y_1|X) = \lim_{t\to\infty} P_0(t) = \frac{D_0}{B},$$

$$P(Y_2|X) = \lim_{t\to\infty} P_2(t) = \frac{D_2}{B}.$$

Finally, let the a priori probability $P(X) = P(\text{fault exists}) = P$. Then, we have:

$$P(X\cap Y_1\cap Z) + P(X\cap Y_2\cap Z)$$

$$= P\frac{D_0}{B} \left[\frac{B - D_1}{B} u(s) - e^{-(A/2)s} \frac{-D_1}{B} \cos\sqrt{B - \frac{A^2}{4}}\, s - e^{-(A/2)s} \frac{2BC_1 - AD_1}{2B\sqrt{B - \frac{A^2}{4}}} \sin\sqrt{B - \frac{A^2}{4}}\, s \right]$$

$$+ P\frac{D_2}{B}\left[\frac{B - D_2''}{B}u(s) - e^{-(A/2)s}\frac{-D_2''}{B}\cos\sqrt{B - \frac{A^2}{4}}\,s \right.$$

$$\left. - e^{-(A/2)s}\frac{2BC_2'' - AD_2''}{2B\sqrt{B - \frac{A^2}{4}}}\sin\left(\sqrt{B - \frac{A^2}{4}}\,s\right)\right]$$

$\leq \alpha.$

Now, if we put:

$$\frac{-D_i^*}{B} = g_i,$$

$$\frac{2BC_i^* - A D_i^*}{2B\sqrt{B - \frac{A^2}{4}}} = h_i, \text{ and}$$

$$\sqrt{B - \frac{A^2}{4}}\,s = \beta.$$

Then,

$$\frac{-D_i^*}{B}\cos\sqrt{B - \frac{A^2}{4}}\,s + \frac{2BC_i^* - A D_i^*}{2B\sqrt{B - \frac{A^2}{4}}}\sin\sqrt{B - \frac{A^2}{4}}\,s$$

$$= g_i\cos\beta + h_i\sin\beta$$

$$= \sqrt{g_i^2+h_i^2}\,(\sin\theta\cos\beta + \cos\theta\sin\beta)$$

$$= \sqrt{g_i^2+h_i^2}\,[\sin(\theta+\beta)]$$

where $\sin\theta = \frac{g_i}{\sqrt{g_i^2+h_i^2}}$, $\cos\theta = \frac{h_i}{\sqrt{g_i^2+h_i^2}}$.

Since,

$$-\sqrt{g_i^2 + h_i^2} \leq g_i\cos\beta + h_i\sin\beta \leq \sqrt{g_i^2 + h_i^2}$$

Then,

$$P(X \cap Y_1 \cap Z) + P(X \cap Y_2 \cap Z)$$

$$= P\frac{D_0}{B}\left[\frac{B - D_1}{B} + e^{-(A/2)s\,\Omega_1}\right] + P\frac{D_2}{B}\,B\left[\frac{B - D_2''}{B} + e^{-(A/2)s\,\Omega_2}\right]$$

$$\leq \alpha.$$

where $\Omega_i = \sqrt{g_i^2 + h_i^2}$, i = 1,2.

Hence,

$$\frac{P}{B}\,e^{-(A/2)s}\,(D_0\,\Omega_1 + D_2\,\Omega_2) \leq \alpha - \frac{P}{B^2}\,(D_0(B-D_1) + D_2(B-D_2''))$$

Consequently,

$$e^{(A/2)s} \geq \frac{(D_0\,\Omega_1 + D_2\,\Omega_2)\frac{P}{B}}{\alpha - \frac{P}{B^2}\,(D_0(B-D_1) + D_2(B-D_2''))}$$

$$= W \qquad (4.16)$$

Thus, the testing time required is

$$s = \frac{2}{A}\,\ln(W) \qquad (4.17)$$

This equation gives us an upper bound of testing time after parameters of the model and A are determined. The necessary condition for this equation to hold is

$$\alpha > \frac{P}{B^2}\,(D_0(B-D_1) + D_2(B-D_2'')) \qquad (4.18)$$

If it is not satisfied, then no test is needed, because the quality of the circuit is good enough to pass the error probability α without any testing.

For a given circuit, with each line i stuck-at-k, k = 0,1, we may have the corresponding testing time s(i,k) calculated by the above equation. However, in order to test the circuit for an unknown single fault, we must take

$$s(max) = \max_{i=1}^{\text{total \# of lines}} \max_{k=0}^{1} s(i,k)$$

If we apply test-input patterns with testing time at least s(max) to the circuit, we will have confidence level $(1-\alpha)$ that no detection error occurs. The procedure can be applied as well to test for any combination of stuck-at-0 or stuck-at-1 faults in the circuit.

4.6.2 Testing Time Calculation in Case $(B - \frac{A^2}{4}) < 0$: A Numerical Approach

During the computer simulation of various circuits using the existing testing time calculation method, we have experienced that there are many circuits which do not meet the condition $(B - \frac{A^2}{4}) > 0$. This is a critical problem for further research, since we have to stop simulation when the condition $(B - \frac{A^2}{4}) > 0$ is not met. Therefore, we must try to solve the problem in case that the condition $(B - \frac{A^2}{4}) > 0$ is not met., i.e. for the condition $(B - \frac{A^2}{4}) < 0$. We first try to approach the problem using a closed form method before using a numerical method to solve the problem.

4.6.2.1 Closed Form Approach

$$\begin{aligned} P(Z|X\cap Y_1) &= 1 - P_{01}(s) \\ &= 1 - \left[\frac{D_1}{B}u(s) + e^{-(A/2)s} \left\{ \frac{-D_1}{B} \cos\sqrt{B-\frac{A^2}{4}}\, s \right.\right. \\ &\quad \left.\left. + \frac{2BC_1 - AD_1}{2B\sqrt{B - \frac{A^2}{4}}} \sin\sqrt{B-\frac{A^2}{4}}\, s \right\} \right] \\ &= \frac{B - D_1}{B}u(s) - e^{-(A/2)s} \left[\frac{-D_1}{B} \cos\sqrt{B-\frac{A^2}{4}}\, s \right. \\ &\quad \left. + \frac{2BC_1 - A D_1}{2B\sqrt{B - \frac{A^2}{4}}} \sin\sqrt{B-\frac{A^2}{4}}\, s \right] \end{aligned}$$

Let $(B - \frac{A^2}{4}) = z^2 < 0$, where $z > 0$,

then ,

$$\cos\sqrt{B-\frac{A^2}{4}}\,s = \cos jzs = \frac{e^{zs} + e^{-zs}}{2}, \quad \sin\sqrt{B-\frac{A^2}{4}}\,s = \sin jzs = -\frac{e^{zs} - e^{-zs}}{2j}$$

and

$$\sqrt{B-\frac{A^2}{4}} = jz, \text{ where } j = \sqrt{-1}.$$

Then,

$$\begin{aligned} P(Z|X\cap Y_1) &= 1 - P_{01}(s) \\ &= 1 - \left[\frac{D_1}{B}u(s) + e^{-(A/2)s} \left\{ \frac{-D_1}{B} \frac{e^{zs} + e^{-zs}}{2} \right.\right. \\ &\quad \left.\left. + \frac{2BC_1 - AD_1}{2Bjz} \left(-\frac{e^{zs} - e^{-zs}}{2j} \right) \right\} \right] \\ &= \frac{B - D_1}{B}u(s) - e^{-(A/2)s}\left[\frac{-D_1}{B} \frac{e^{zs} + e^{-zs}}{2} \right. \\ &\quad \left. + \frac{2BC_1 - AD_1}{2Bz} \frac{e^{zs} - e^{-zs}}{2} \right]. \end{aligned}$$

In the same way,

$$\begin{aligned} P(Z|X\cap Y_2) &= 1 - P_{21}(s) \\ &= 1 - \left[\frac{D_2''}{B}u(s) + e^{-(A/2)s} \left(\frac{-D_2''}{B} \frac{e^{zs} + e^{-zs}}{2} \right) \right. \\ &\quad \left. + \frac{2BC_2'' - AD_2''}{2Bjz} \left(-\frac{e^{zs} - e^{-zs}}{2j} \right) \right] \\ &= \frac{B - D_2''}{B}u(s) - e^{-(A/2)s}\left[\frac{-D_2''}{B} \frac{e^{zs} + e^{-zs}}{2} \right. \\ &\quad \left. + \frac{2BC_2'' - AD_2''}{2Bz} \frac{e^{zs} - e^{-zs}}{2} \right]. \end{aligned}$$

Following the same steps as in Section 4.6.1,

$$P(X\cap Y_1\cap Z) + P(X\cap Y_2\cap Z)$$

$$= P\frac{D_0}{B}\left[\frac{B - D_1}{B}u(s) - e^{-(A/2)s}\left\{ \frac{-D_1}{B} \frac{e^{zs} + e^{-zs}}{2} \right.\right.$$

$$+ \frac{2BC_1 - AD_1}{2Bz} \frac{e^{zs} - e^{-zs}}{2} \Big\} \Big]$$

$$+ P\frac{D_2}{B}\left[\frac{B - D_2''}{B}u(s) - e^{-(A/2)s}\left\{\frac{-D_2''}{B}\frac{e^{zs} + e^{-zs}}{2} + \frac{2BC_2'' - AD_2''}{2Bz}\frac{e^{zs} - e^{-zs}}{2}\right\}\right]$$

$$\leq \alpha.$$

Now, if we put:

$$\frac{-D_i^*}{B} = g_i, \frac{2BC_i^* - A D_i^*}{2Bz} = h_i, \text{ and } zs = \beta.$$

Then,

$$\frac{-D_i^*}{B}\frac{e^{zs} + e^{-zs}}{2} + \frac{2BC_i - AD_i}{2Bz} + \frac{e^{zs} - e^{-zs}}{2}$$

$$= g_i \cosh\beta + h_i \sinh\beta$$

$$= \sqrt{g_i^2+h_i^2}\left[\frac{g_i}{\sqrt{g_i^2+h_i^2}}\cosh\beta + \frac{h_i}{\sqrt{g_i^2+h_i^2}}\sinh\beta\right]$$

$$= \sqrt{g_i^2+h_i^2}\ (\sin\theta\cosh\beta + \cos\theta\sinh\beta)$$

Since the equation above is nonlinear , we cannot simplify it further by applying the approximation method for the sine function used in the case of $(B - \frac{A^2}{4}) > 0$. Therefore we need another approach to solve the nonlinear equation in case of $(B - \frac{A^2}{4}) < 0$.

4.6.2.2 Numerical Approach

Again,

$$P(X \cap Y_1 \cap Z) + P(X \cap Y_2 \cap Z)$$

$$= P\frac{D_0}{B}\left[\frac{B - D_1}{B}u(s) - e^{-(A/2)s}\left\{\frac{-D_1}{B}\frac{e^{zs} + e^{-zs}}{2} + \frac{2BC_1 - AD_1}{2Bz}\frac{e^{zs} - e^{-zs}}{2}\right\}\right]$$

$$+ P\frac{D_2}{B}\left[\frac{B - D_2''}{B}u(s) - e^{-(A/2)s}\left\{\frac{-D_2''}{B}\frac{e^{zs} + e^{-zs}}{2} + \frac{2BC_2'' - AD_2''}{2Bz}\frac{e^{zs} - e^{-zs}}{2}\right\}\right]$$

$$\leq \alpha.$$

The above equation can be written as

$$D_0\left[(B - D_1) - e^{-(A/2)s}\left\{- D_1\frac{e^{zs} + e^{-zs}}{2} + \frac{2BC_1 - AD_1}{2z}\frac{e^{zs} - e^{-zs}}{2}\right\}\right]$$

$$+ D_2\left[(B - D_2'') - e^{-(A/2)s}\left\{- D_2''\frac{e^{zs} + e^{-zs}}{2} + \frac{2BC_2'' - AD_2''}{2z}\frac{e^{zs} - e^{-zs}}{2}\right\}\right]$$

$$\leq \frac{\alpha B^2}{p}$$

The left-hand side terms can be changed as follows, by using hyperbolic function definitions,

$$D_0 (B - D_1) - D_0 e^{-(A/2)s}\left\{- D_1 \cosh zs + \frac{2BC_1 - AD_1}{2z}\sinh zs\right\}$$

$$+ D_2 (B - D_2'') - D_2 e^{-(A/2)s}\left\{- D_2'' \cosh zs + \frac{2BC_2'' - AD_2''}{2z}\sinh zs\right\}$$

$$\leq \frac{\alpha B^2}{p}$$

Then we have an equation,

$$\frac{\alpha B^2}{p} = D_0 (B - D_1) + D_2 (B - D_2") + (D_0 D_1 + D_2 D_2") (\cosh zs)\ e^{-(A/2)s}$$

$$- \{\frac{2BC_1 - AD_1}{2z} + \frac{2BC_2" - AD_2"}{2z}\} (\sinh zs)\ e^{-(A/2)s}.$$

To solve for s (the testing time), we have to manipulate the above equation as:

$$\frac{\alpha B^2}{p} - D_0(B - D_1) - D_2 (B - D_2")$$
$$= (D_0 D_1 + D_2 D_2") (\cosh zs)\ e^{-(A/2)s}$$

$$- \{\frac{2BC_1 - AD_1}{2z} + \frac{2BC_2" - AD_2"}{2z}\} (\sinh zs)\ e^{-(A/2)s}.$$

Let

$$X = \frac{\alpha B^2}{p} - D_0(B - D_1) - D_2 (B - D_2"),$$

$$Y = D_0D_1 + D_2D_2", \text{ and}$$

$$Z = \frac{2BC_1 - AD_1}{2z} + \frac{2BC_2" - A D_2"}{2z}.$$

Then, the equation becomes:

$$X = Y (\cosh zs)\ e^{-(A/2)s} - Z (\sinh zs)\ e^{-(A/2)s}, \text{ and,}$$

$$X = Y\frac{e^{zs} + e^{-zs}}{2} e^{-(A/2)s} - Z\frac{e^{zs} - e^{-zs}}{2} e^{-(A/2)s}$$

Multiplying by 2 and rearranging the above equation, we can obtain

$$2X = (Y-Z)\ e^{(z - A/2)s} + (Y + Z)\ e^{-(z + A/2)s}$$

Multiplied by $e^{(A/2)s}$, we have

$$2X\, e^{(A/2)s} = (Y - Z)e^{zs} + (Y + Z)e^{-zs}$$

Taking natural log, we get

$$\ln(2X) + \frac{A}{2}s = \ln\left[(Y - Z)\, e^{zs} + (Y + Z)\, e^{-zs} \right]$$

$$s = \frac{2}{A}\left[\ln\left((Y - Z)e^{zs} + (Y + Z)e^{-zs}\right) - \ln(2X) \right].$$

As shown in the above equation, not only the right-hand side but also the left-hand side of the equation is a function of s (the testing time).

Let

$$f(s) = 2X - (Y - Z)e^{-(z - A/2)s} - (Y + Z)e^{-(z + A/2)s}$$

To solve this nonlinear equation numerically, we used the Method of Bisection [43]. The solution of the equation can be obtained by finding the root which satisfies f(s) =0. More specifically, given the interval [a,b] and different signs of f(a) and f(b), for some s in [a,b] , say s = k , if f(k) =0, then k is the root of function f(s). This method is illustrated in Fig 4.4, where the root lies between a and b.

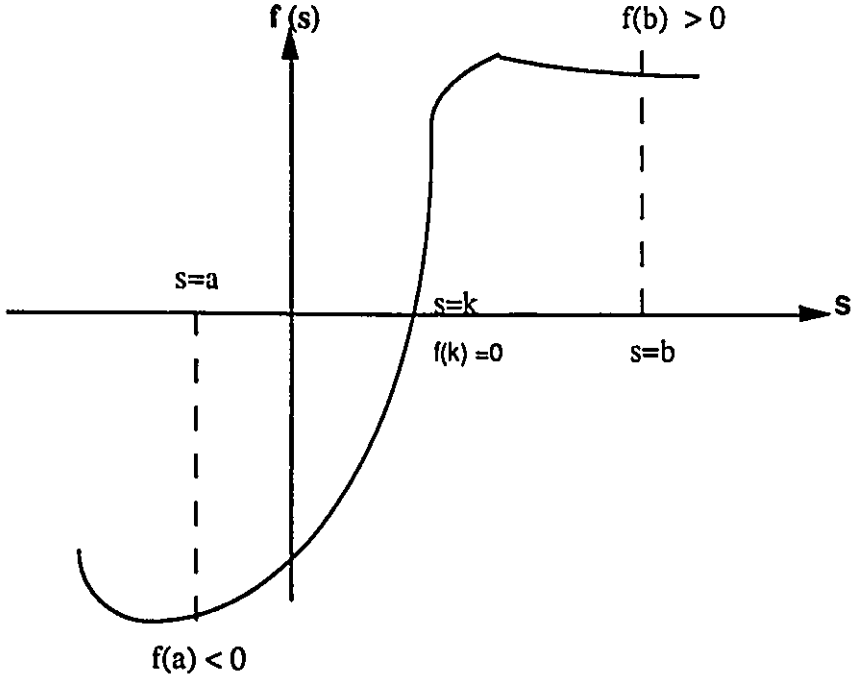


Fig.4.4 Method of bisection.

Following algorithm has been used to solve the nonlinear equation f(s).

Step 1 Get the initial values of negative s (S_{neg}), positive s (S_{pos}), and maximum number of iterations (I_{max}).

Step 2 If $f(S_{neg}$ is positive, get another S_{neg} until $f(S_{neg})$ is negative.
If $f(S_{pos})$ is negative, get another S_{pos} until $f(S_{pos})$ is positive.
Now $f(S_{neg})$ and $f(S_{pos})$ have different signs, and we can predict that the root lies between S_{neg} and S_{pos}.

Step 3 For iteration =0,1,.., I_{max}, do

$$S_{new} = \frac{S_{neg} + S_{pos}}{2}$$

Calculate $f(S_{new})$,
if $|f(S_{new})| < \varepsilon$ (preset value: close to 0), break.
if $f(S_{new}) < 0$, swap the value of S_{neg} and S_{pos}.

Using the above algorithm, a C program was written and found to be working properly. The program is next integrated into the main program which simulates CUT (Circuit Under Test) and calculates the testing time. This is discussed later.

4.6.3 A Numerical Approach for the Case of $(B - \frac{A^2}{4}) > 0$

In case of $(B - \frac{A^2}{4}) > 0$, we can also calculate testing time numerically with some manipulations on portion of the above expressions.

Let

$$U = \sqrt{-(B - \frac{A^2}{4})}\,.$$

Since $(B - \frac{A^2}{4}) > 0$, U is a complex number,

let

$$B - \frac{A^2}{4} = V.$$

Then

$$U = \sqrt{-V} = jV, \text{ where } j = \sqrt{-1}\ .$$

The expression for X and Y are changed such as,

$$X = \frac{\alpha B^2}{p} - D_0 (B - D_1) - D_2 (B - D_2''),$$

$$Y = D_0 D_1 + D_2 D_2''.$$

However, we let $B - \frac{A^2}{4} = z^2$ in case of $(B - \frac{A^2}{4}) < 0$, and we can obtain a relationship between z and V as

$$z = jV.$$

Substituting this into the expresstion for Z, we have

$$Z = \frac{2BC_1 - AD_1}{2z} + \frac{2BC_2'' - AD_2''}{2z}$$

$$= \frac{2BC_1 - AD_1}{2jV} + \frac{2BC_2'' - AD_2''}{2jV}.$$

If we let,

$$W = \frac{2BC_1 - AD_1}{2V} + \frac{2BC_2'' - AD_2''}{2V},$$

then

$$Z = \frac{W}{j}.$$

Substituting $Z = \frac{W}{j}$ and $z = jV$ into the equation f(s),

$$f(s) = 2X - (Y - Z)\, e^{(z - A/2)s} - (Y + Z) e^{-(z + A/2)s}$$

$$= 2X - (Y - \frac{W}{j})\, e^{(jV - A/2)s} - (Y + \frac{W}{j}) e^{-(z + A/2)s}$$

$$= 2X - e^{-(A/2)s} [\, (Y + jW) e^{jV s} + (Y - jW) e^{-jV s} \,]$$

Since $(Y + jW)e^{jV s}$ and $(Y - jW)e^{-jV s}$ are complex conjugates,

$$(Y + jW)e^{jVs} + (Y - jW)e^{-jV s} = 2\,\mathrm{Re}\,[\, (Y + jW)e^{jVs} \,].$$

Then f(s) becomes,

$$f(s) = 2X - 2e^{-(A/2)s}\, \mathrm{Re}\,[\, (Y + jW)\ \ e^{jVs} \,]$$

$$= 2X - 2e^{-(A/2)s}\, \mathrm{Re}\,[\, (Y + jW)\ \ \{\cos(Vs) + j\sin(Vs)\}]$$

$$= 2X - 2e^{-(A/2)s}\, \mathrm{Re}\,[\, (Y + jW)\ \ \{\cos(Vs) + j\sin(Vs)\}]$$

$$= 2X - 2e^{-(A/2)s}\, \{\, Y\cos(Vs) - W\sin(Vs) \,\}$$

We can solve this equation numerically using the bisection method described in the case of $(B - \frac{A^2}{4}) < 0$. In the C program, we have to write a subroutine specifying the expressions of V, W and f(s).

4.7 Chapter Summary

In this chapter, a continuous Markov model for faulty sequential machines was defined, and differential equations of the model were derived. Markov parameters and state probabilities were derived. Testing time was calculated, for the cases, $(B - \frac{A^2}{4}) < 0$ and $(B - \frac{A^2}{4}) > 0$, using a numerical method as well as the closed form solution method.

CHAPTER 5
FAULT DETECTION OF DECOMPOSABLE SEQUENTIAL CIRCUITS THROUGH COMPUTER SIMULATION

5.1 Introduction

Test generation for high fault coverage for sequential circuits has been a difficult task [3, 4] and an expensive process. This is due to the large search space to be considered in test pattern generation [50]. Different approaches have been taken in the past to solve the problem of fault detection and test generation in sequential circuits [29, 33, 41, 50, 57, 70]. A popular approach called the scan design [7], is often used where the problem of test generation in sequential circuits is transformed into that of combinational circuits. Unfortunately, this approach is primarily restricted to synchronous sequential circuits free of critical races. And also the testing time associated with scan design is longer than that of non-scanned design, because values have to be sequentially scanned into and out of memory elements one clock cycle at a time. Moreover, when a circuit is very large and complex, the test generation can be quite complex, making the ad hoc approaches ineffective. Therefore, alternative methods should be considered.

In this chapter, a random testing approach utilizing the decomposition theory (Chapter 3) and the continuous Markov model (Chapter 4) is presented for the detection of permanent faults in sequential circuits. The proposed approach has the potential to provide good testability and modularity of a circuit design in addition to the benefits of random testing. Through the computer simulation, the maximum testing times for a specified degree of confidence to detect different stuck faults are calculated for some example circuits with and without decomposition principles. The maximum testing times in the two cases are compared to observe the benefits of decomposition.

This chapter is organized as follows. Section 5.2 briefly outlines the existing approaches including the basic circuit partitioning approach and the deterministic approach in decomposable sequential machines. This then leads to the discussion on the random testing using a continuous parameter Markov model in Section 5.3, where we discuss fault

detection procedures, data structures for sequential machine, and algorithm development. In Section 5.4, the benefits from the proposed approach against its counterparts are described. Experimentation on various sequential circuits is given in Section 5.5 followed by a summary in Section 5.6.

5.2 Existing Approach: Deterministic Approach on Parallel-Decomposable Sequential Machines

Das *et al* [18] have presented a deterministic approach for the reduction of test sequence length of parallel-decomposable sequential machines. In this approach, the decomposable sequential machine (composite) is realized by a parallel connection of n component machines and fault detection experiments for each component are designed, assuming other components are fault-free. If the experiments for each component machine are successful, this verifies that the composite machine is fault-free. The proposed procedure, called the machine transition identification experiments, measures the transition of a component machine in response to the restricted (distinguishing) input sequences with other component in a fixed reference. The authors claimed that the designed experiments are generally shorter than the composite machine. A saving in the length of the experiment was achieved due to the fact that the composite machine has a number of states equal to the product of the number of its component states, while checking of component machines requires identifying and verifying transitions for the sum of the numbers of states. For instance, consider a sequential machine with 8 states realized as two parallel connected component machines, one with 4 states and the other with 2 states. In order to check the composite machine, we have to identify and verify transitions for 8 states; on the other hand, checking of the two component machines requires identifying and verifying transitions for the sum of the numbers of states of the component machines, that is, 6.

The authors pointed out that if the output of a component machine is not directly observable, it may be difficult to identify the component machine. Chandrasekharan *et al* [14] extended the above work more systematically and quantitatively, and further claimed that their method was suited for testing large-state sequential machines.

5.3 A New Approach: A Random Testing Utilizing Markov Model and Parallel Decomposition

5.3.1 Basic Concept

The circuit partitioning concept or decomposition theory is quite well known in the literature [39, 44] and has been applied in circuit synthesis in many circumstances [33]. We can use this principle to reduce the complexity of test generation in sequential circuits. By the partitioning of a sequential circuit, we mean the determination of a set of smaller circuits which can be combined together to form a composite circuit identical in behavior to the given circuit. These smaller circuit will each have, in general, fewer states than the original circuit.

Given a large sequential machine M, we decompose it into several smaller partitions using the state substitution property discussed Chapter 3. Each partition becomes a sequential circuit with a smaller number of states than the original circuit, thus making it easy to analyze each partition separately. Once all the partitions of the original circuit are obtained, we then develop a Markov model and analyze the problem of detecting permanent faults in the partitions by random testing, using a three-state continuous parameter Markov model discussed in Chapter 4. For a specified degree of confidence, it is fairly straightforward to derive the model, and to calculate the lengths of the test-input patterns required for fault detection in a partition. Using the model, it is also possible to determine the maximum testing time needed so that the probability of a wrong conclusion is smaller than or equal to some prespecifed value α.

5.3.2 Fault Detection Procedure

The fault detection procedure for a large sequential machine is divided into three parts such as: testable design phase, testing and simulation phase, and testing time calculatiuon and evaluation phase. The three phases are connected together in order to complete the procedures. A flow chart of the test procedure is shown in Fig. 5.1.

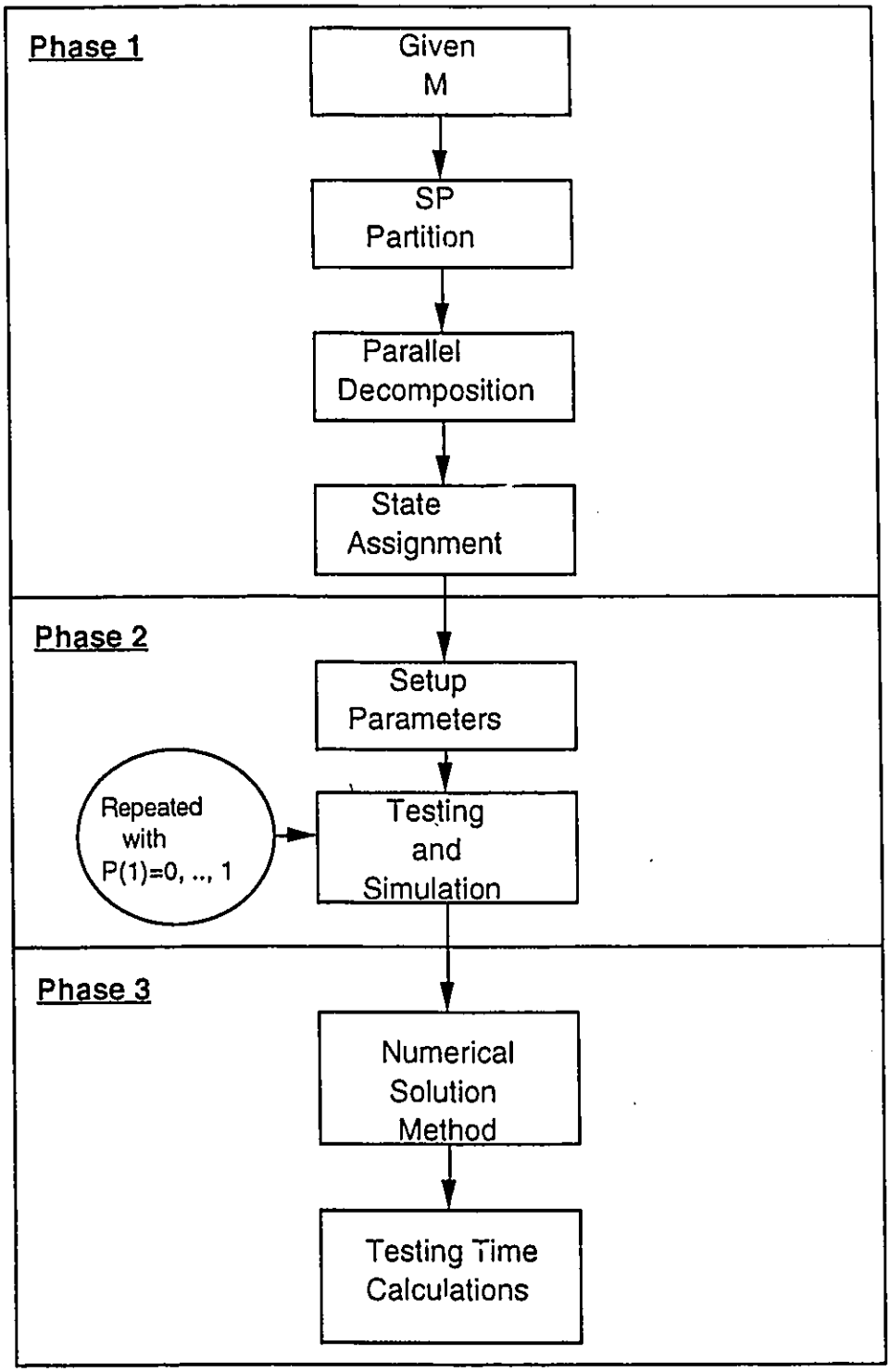


Fig. 5.1 A flow chart of the test procedure.

Under each phase, there are high level tasks, which can later be used for the development of algorithms. The detailed explanations are as follows:

Phase 1: Testable Design

Perform parallel decomposition of the large sequential machine M (into two or more submachines) using SP partition theory. Realize a sequential circuit by a state assignment on the decomposed machine.

Phase 2: Testing and Simulation of Each Component Machine

Given a sequential circuit in a partition with certain faults specified, derive the original state table and its error version from an analysis of the partition under fault-free and faulty conditions, respectively. By simulating these two tables, one can calculate the parameters of the desired Markov model. We set up a number of simulation runs each with a different probability of producing a 1, P(1), from random number generator by defining the parameters in the simulation program. For example, we can run the simulation 1000 times each with a different value of P(1) such as, P(1)=0.000, 0.001, ..,0.999, 1.000.

Phase 3: Testing Time Calculation and Evaluation

We can determine the maximum testing time for each partition so that the probability of a wrong conclusion is smaller than or equal to some predetermined value α.To minimize the probability of such a wrong conclusion we have to apply a large number of test patterns to the circuit until either the fault is detected, or the confidence about the circuit being error free is larger than or equal to some prespecified value. Chapter 4 described the calculation of the maximum testing time (T_{max}) in a sequential circuit based on a three-state continuous Markov model and give an upper bound on the testing time. If we apply test-input patterns to a partition with testing time at least equal to the maximum testing time, we will have the confidence degree 1 - α that no detection error occurs. We calculate n different maximum testing times (T_{max}) using the parameters passed from the Phase 2, where n is the number of simulation with different probability of 1 in the random number generator.

5.3.3 Data Structure for Sequential Machine

Throughout our discussion, the only information required is regarding the state table of a sequential machine M. Therefore, prior to the development of the algorithm for the fault detection procedures, we have to define a data structure to represent the state table of the machine in the computer and develop an algorithm for specifying the state table entries to the computer.

5.3.3.1 Representation of Sequential Machine in Computer

(a) Inputs to the Computer for State Table Configuration:

Since the number of variables in the sequential machine, such as, input variables, state variables, and output variables determine the size of state table, we need the number of these variables, as inputs to the computer, to configure the size of the state table of machine M.

(b) State Transition Array and Output Array:

In the computer, the state table can be expressed as two separate but the same sized arrays: one is state transition array for state transition table and the other is output table array for output table. Each array is a two-dimensional array with its column size dependent on the number of states and its row size dependent on the number of inputs entered. For Moore model machine, only one array is enough to express the state table because the output is fixed according to the present state.

5.3.3.2 Algorithm for State Table Entry to Computer

Step 1: Get the number of states, inputs, and outputs.
Step 2: For (0 to the number of states)
For (0 to the number of inputs)
Enter next state followed by output
If (next state > the number of states)
Enter next state again
If (output > the number of outputs)
Enter output again
Increment element of array
Increment master array of array

Example 5.1: Consider the state table of the machine M as shown below.

Present State	Next State,output	
	x=0	x=1
S_0	S_3,0	S_2,0
S_1	S_5,0	S_2,1
S_2	S_4,1	S_1,1
S_3	S_1,1	S_4,0
S_4	S_0,1	S_3,0
S_5	S_2,0	S_3,1

Table 5.1 State table of machine M.

Next state array = [the number of states] [the number of inputs]
= [6] [2],
and the data format in the computer is:

$$\begin{bmatrix} 3 & 2 \\ 5 & 2 \\ 4 & 1 \\ 1 & 4 \\ 0 & 3 \\ 2 & 3 \end{bmatrix}$$

Output array = [the number of states] [the number of inputs]
= [6] [2],
and the data format in the computer is:

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Output array =[the number of states] [the number of inputs]
=[6][2],
and the data format in the computer is:

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

5.3.4 Algorithm Development

Based on the data structure and the statements under each of the three phases of the procedures, we can develop algorithms for each phase. Each phase consists of a sequence of tasks, for which an algorithm is to be developed.

Phase 1: Machine Decomposition

Step 1: State Table Entry.
The state table of machine M is entered according to the data structure discussed in 5.3.3.

Step 2: Computation of SP partitions

i) Find smallest partitions.

ii) Compute all the possible sums of SP partition pairs. These sums generate additional SP partitions.

Step 3: Check the necessary conditions of parallel decomposition $\pi_j . \pi_k = \pi_0$, where π_j and π_k are SP partitions and π_0 is zero partition.

If yes, the SP partition π_j and π_k are chosen for component machines generation and go to next step.

If not, apply the state splitting technique [44] starting from Step 1.

Step 4: Build two component state tables as follows:

i) For each block of SP partition π_j, assign a new state.

ii) Apply the new states on the state machine M and build a state table of the component machine M_j.

iii) For SP partition π_k, build a state table of the component machine M_k.

Phase 2: Testing and Simulation for Each Component Machine

In this phase, the machine M is tested using the Markov model. We then analyze the problem of detecting permanent faults by random testing, using a three-state continuous parameter Markov model discussed in Chapter 4.

Step 1: Simulation parameters setup:

Seed for random generator

Upper and lower limits of testing time for numerical solution

The resolution of probability of 1 of random generator and the total number of testing time calculations n

Confidence level parameter α

Total number of input patterns

Scaling factor=total number of input patterns per unit time

A priori probability of the stuck fault

Bring the machine M_j or M_k to the initial state.

Step 2: The original state table and its error version.

For the machine M, we have two two-dimensional arrays, the next state array and output array. We call these arrays the Fault-Free Transition array (FFT) and the Fault-Free Output Array (FFO) in the program. Column and row of both arrays are dependent on the number of states and external inputs of the machine or the circuit.

Determine the size of arrays with reference to the state table of machine or the circuit under test (CUT) given.

With a stuck-at fault specified on the machine M, we will have another two two- dimensional arrays. These arrays are called the Faulty Transition array (FT) and the Faulty Output Array (FO) in the program.

Step 3: Simulation and testing.

Simulate the machine n times, each with different value of probability of 1 P(1) from random number generator as follows:

Collect the nine different transition counts.

Find the parameters of Markov model from those counts.

Calculate 9 parameters (A,B, ..., C_2", D_2") and the convergence condition B - $\frac{A^2}{4}$, which are used for the calculation of testing time calculation.

Phase 3: Testing Time Calculation and Evaluation

Step 1: Testing time calculation.

Using the passed 9 parameters (A,B, ..., C_2", D_2") and the convergence condition B - $\frac{A^2}{4}$, calculate testing time using the numerical method developed in Chapter 4. Two different functions for (B - $\frac{A^2}{4}$) > 0 and (B - $\frac{A^2}{4}$) < 0 should be designed in the simulation program.

Step 2: Result evaluation.

Find reliable results (testing time) by observing the output of simulation runs.

5.4 Benefits from the Proposed Approach

Our approach is based on random testing which utilizes parallel decomposition concept and Markov models. Therefore we can consider benefits of our approach by comparing each counterpart, such as, our approach (random testing approach) vs. deterministic approach [18], our approach (parallel-decomposition approach) vs. composite testing without decomposition, and our approach (improved method to calculate testing time) vs. existing Markov model simulation. A summary of significant benefits obtained from the proposed approach are as follows:

5.4.1 Random Testing Approach vs. Deterministic Approach [18]

Das *at al* [18] presented a deterministic approach for the reduction of test sequence length of parallel-decomposable sequential machines. This approach requires a complex test generation procedure for the large-state machines even though they are decomposed. Moreover, the following assumptions were made:

1. Each component machine possesses a distinguishing sequence.
2. Each component machine can be in a reference (fixed) state.
3. Machine M is strongly connected.

The random testing approach will provide us with the following benefits:

1. The random testing approach eliminates the assumptions 1 and 2.
2. Simplicity (No concern about the reference state of the component machine is required).
3. No calculation of complex test sequences to identify machines is required.

5.4.2 Parallel-Decomposition Approach vs. No Decomposition.

If we implement parallel decomposition technique in testing of a large-state sequential circuit, the following two benefits can be gained by this approach:

1. Testability and modularity are improved. i.e. it may be much easier to detect or to analyze faults.

2. Shorter testing time (reduction of test sequence length) is obtained. This results in keeping the cost of the test generation and the real testing time whithin reasonable limits.

5.4.3 Improved Method to Calculate Testing Time vs. Existing Markov Model Simulation [29].

In case the condition $(B - \frac{A^2}{4}) > 0$ is not met, Das *et al* [29] suggest that simulation should be repeated until finding $(B - \frac{A^2}{4}) > 0$, by applying a new value of probability of 1 for each simulation, from the random number generator. This is a somewhat time consuming process during simulation of sequential circuit. This algorithm is shown to work properly in the simulation of some circuits (e.g.74 LS 164 and 74 LS191); however, through intensive experimentations, we discovered that sometimes, the desired condition could not be reached in spite of a numerous values of P(1) applied.

The authors also solved the equation for testing time using an approximate estimation in the closed form solution. That is, to estimate the value of " A sin α ", an inequality form " $- A \leq A \sin \alpha \leq A$ " was used. However, in some cases, this method may not provide us an accurate testing time.

These critical problems have motivated us to improve the random testing method by solving the nonlinear equations, so that the improved one works properly in any sequential circuit. We also designed an algorithm to find an optimum P(1) so that we can discover the shortest testing time for a given fault in sequential circuits. In the next section, we will consider the problem we found during simulations and the benefits from the developed method.

5.5 Experimentations

Based on the algorithm developed in the last section, a complete program to simulate and test sequential circuits and to calculate the maximum testing time was written. The computer simulation system environment was the following:

Hardware: DEC Station 3100
Operating system: Ultrix, UNIX Implementation
Language: C

As a source of input patterns, we used a random number generator called 'rand' available in ULTRIX C library. This rand subroutine uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudo-random numbers in the range of 2^{31}- 1. The generator is reinitialized by calling 'srand' with 1 as argument. It can be set to a random starting point by calling srand with whatever user like as argument. The determinations of '1' or '0' from the random number generator is programmed as:

For a given P(1), probability of 1 of the random number generator, a threshold value called LIMIT is calculated as LIMIT = $(2^{31}- 1).P(1)$ .Then we call up a random number by the 'rand()' subroutine call. If it is in the range of 0 to LIMIT, it is treated as a '1'; otherwise, it is treated as a '0'.

The simulation and testing parameter setup is:

Seed for random generator: an arbitrary number 2345
The total number of test time calculations: 11
Resolution of the probability of 1, P(1), from random number generator: 0.1
Confidence level parameter: α=0.05 i.e. a 95% confidence
Total number of input patterns: 100,000
Scaling factor=total number of input patterns per unit time: 1,000/mS
A priori probability of the stuck fault: 0.1

The organization of the experimentation is as follows:

First, a commonly used sequential circuit from a logic data book, to which our circuit partitioning or decomposition approach is not applied, is chosen to examine the model. In other words, only Phase 2 and Phase 3 of the algorithm are being used.

Then, we take a sequential circuit which requires numerical method solutions to calculate the maximum testing time since the condition is not met throughout the simulation runs by changing the P(1) values. We deal with the same circuit used in the last section for continuing the investigation.

Next, we take a sequential machine and realize the circuit using the parallel decomposition technique and test the circuit. We also realize a circuit for the same sequential machine with an arbitrary state assignment and test it. Finally, we compare the results between circuits.

In the last example, we simulate a practical circuit, LS191, to prove the practicality of our approach. We partition LS191 into two components, test the several signal lines on each component separately, and obtain the testing times. We also test the same signal lines on the original circuit and obtain the testing times. Then we compare the testing times obtained from the two cases.

5.5.1 Example 1: DM54LS164/DM74LS164

Considered is the DM74LS164, an 8 bit serial in/parallel out shift register, from National Semiconductor logic data book. Its logic diagram are shown as Fig. 5.2.

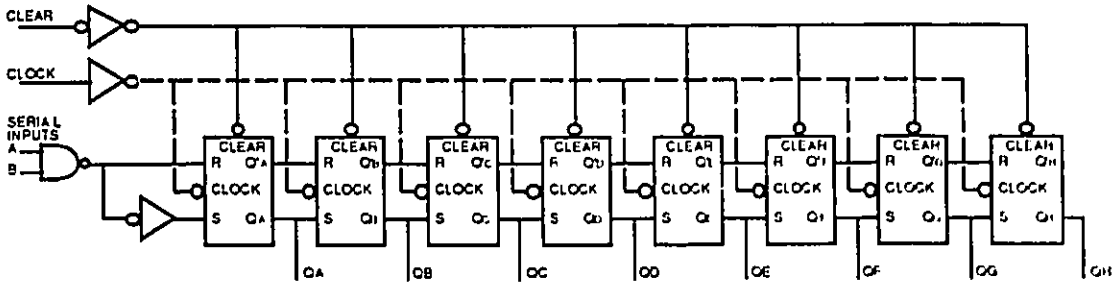


Fig 5.2 Logic representation of DM74LS164.

There are two synchronous gated serial inputs A and B, and an asynchronous input CLEAR. During simulation of the circuit, we pulled the CLEAR input to the HIGH logic level in order not to disable the shift function of the circuit, and tied the serial inputs A and B together so that we could connect the random number generator to a single line. There are eight R-S flip-flops connected together in an iterating form; therefore, in the simulation program, we need arrays with a size of 256 (2^8 different states) columns and 2 rows such as, FFT[256][2],FFO[256][2].

Since each individual RS flip-flop of the circuit is in an ambiguous state in the case of R=S=1, we only deal with stuck at zero (s-a-0) faults when testing each individual RS flip-flop of the circuit.

Results of the simulation are presented in Table 5.2, where the first column gives stuck-at-0 fault at the signal line of a flip-flop, and P(1) is the probability of having a logic 1 at the input.

s-a-0 at	P(1)	Markov Model Parameters	Testing Time
Q'_A	0.915	λ_0=0.0, λ_1=3.266, λ_2=0.0, μ_0=5.537, μ_1=7.803, μ_2=4.538	0.30592 ms
Q'_B	0.886	λ_0=0.0, λ_1=4.66, λ_2=0.0, μ_0=5.502, μ_1=10.114, μ_2=5.502.	0.30539 ms
Q'_C	0.886	λ_0=0.0, λ_1=3.842, λ_2=0.0, μ_0=6.286, μ_1=10.128, μ_2=6.286.	0.21160 ms
Q'_D	0.885	λ_0=0.0, λ_1=57.4, λ_2=0.0, μ_0=6.34,μ_1=12.094, μ_2=6,98.	0.16998 ms
Q'_E	0.814	λ_0=0.0, λ_1=7.23, λ_2=0.0, μ_0=8.58, μ_1=15.78, μ_2=8.56.	0.12995 ms
Q'_F	0.731	λ_0=0.0, λ_1=5.351, λ_2=0.0, μ_0=14.27, μ_1=19.636, μ_2=14.285.	0.09055 ms
Q'_G	0.535	λ_0=0.0, λ_1=161.16 λ_2=0.0, μ_0=13.3.9, μ_1=23.21 μ_2=13.91	Condition not met

Table 5.2: Results of simulation of DM74LS164.

5.5.2 Example 2:

Consider a circuit [67] shown in Fig.5.3, which is the same circuit as in Chapter 4 but additional signal lines are labeled such as, L2, L3, and L4.

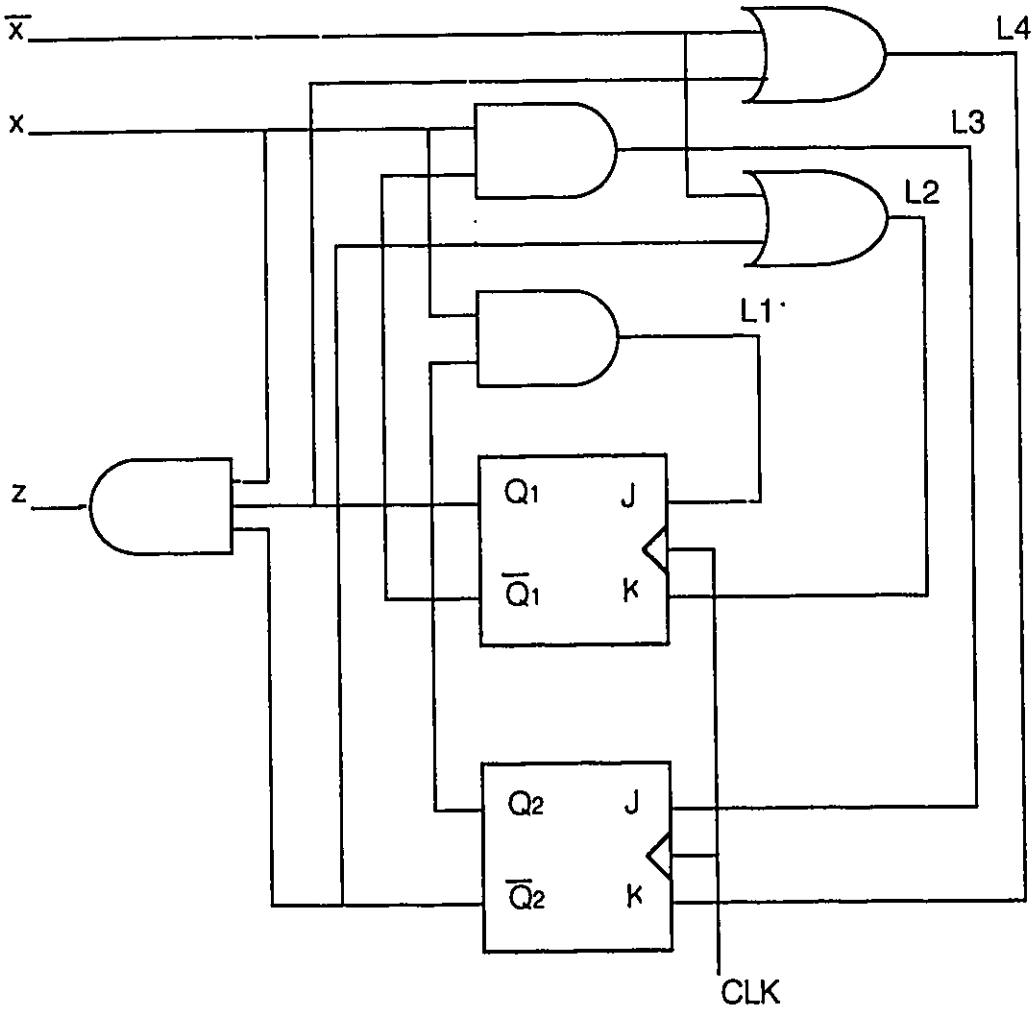


Fig. 5.3 A circuit to be simulated.

For a specified stuck fault at a specific signal line, we ran the simulation program to calculate the convergence criterion $(B - \frac{A^2}{4}) > 0$ for the different value of probability of 1 for the random number generator. The result is shown in Table 5.3, where Li: s-a-0 stands for stuck at zero on the signal line Li, and P(1) stands for the probability of 1 for the

random number generator. A check mark (√) implies that the convergence criterion $(B - \frac{A^2}{4}) > 0$ is met.

P(1)	L1:s-a-0	L1:s-a-1	L2:s-a-0	L2:s-a-1	L3:s-a-0	L3:s-a-1	L4:s-a-0	L4:s-a-1
0.1	-0.8463	-1838.3	-80.236	-0.0070	-80.344	-1801.5	-0.8566	-0.6897
0.2	-9.7402	-1316.1	-252.82	-0.2841	-248.44	-1225.0	-10.182	-0.4221
0.3	-32.448	-913.15	-442.19	-2.0267	-415.42	-789.23	-36.204	-16.424
0.4	-65.064	-607.93	-600.07	-6.2809	-514.29	-481.55	-81.443	-24.337
0.5	-89.862	-358.34	-705.88	-13.643	-504.29	-260.42	-141.31	-21.239
o.6	-84.350	-189.41	-761.67	-24.275	-402.22	-120.95	-207.42	-10.389
0.7	-31.183	-90.980	-767.78	-40.473	-251.24	-41.966	-279.77	-26.414
0.8	√	-78.935	-734.89	-66.304	-146.57	-26.850	-362.49	-112.47
0.9	√	-139.39	687.01	-120.85	-219.83	-90.177	-467.81	-284.73
1.0	√	-225.66	624.92·	-225.68	-624.95	-225.65	-624.97	-486.08

Table 5.3 Calculations for $(B - \frac{A^2}{4})$.

As shown in Table 5.3, except a stuck-at 0 fault on the signal line L1, the condition $(B - \frac{A^2}{4}) > 0$ is not met at all for any P(1). Even though we applied higher resolution values of probability of 1 of random number generator, such as P(1)=0.81, 0.82,.., 0.89 for both cases of stuck faults on all the signal lines L1 to L4, the condition was still not met.

If the condition $(B - \frac{A^2}{4}) > 0$ is not met, further calculations in the closed form solution, discussed in Chapter 4, cannot proceed because the value of $\sqrt{B - \frac{A^2}{4}}$ becomes a complex number. This ends up the calculation of testing time being infinite. In other words, for the case of $(B - \frac{A^2}{4}) < 0$, the testing time calculation is not possible using the closed form method.

For this reason, we have solved the nonlinear equations by using a numerical solution method and developed functions in the main program to calculate the testing time. In this example, for the case of $(B - \frac{A^2}{4}) > 0$, we calculate testing time with the closed form

solution or the numerical method and, for the case of $(B - \frac{A^2}{4}) < 0$, we calculate the testing time with the help of the numerical method using the C subroutine functions developed. The result is shown in Table 5.4.

P(1)	L1:s-a-0	L1:s-a-1	L2:s-a-0	L2:s-a-1	L3:s-a-0	L3:s-a-1	L4:s-a-0	L4:s-a-1
0.1	50.000	0.20876	N/R	50.000	50.000	23.0994	50.000	9.48041
0.2	6.98336	0.11598	N/R	6.83242	7.04592	3.12654	7.12216	1.32037
0.3	1.61293	0.08581	N/R	1.60485	1.64621	1.05448	1.68721	0.43572
0.4	0.58640	0.07116	N/R	0.59814	0.60881	0.53279	0.63513	0.21092
0.5	0.26315	0.06116	N/R	0.28104	0.28113	0.28676	0.29905	0.12146
0.6	0.14248	0.5445	N/R	0.16093	0.15847	0.17641	0.17089	0.08014
0.7	0.08681	0.04876	N/R	0.10480	0.10255	0.11332	0.11009	0.05968
0.8	0.08104	0.04463	N/R	0.07392	0.07473	0.07757	0.07733	0.05008
0.9	0.06384	0.04201	N/R	0.05451	0.06196	0.05541	0.05763	0.04803
1.0	0.05545	0.04122	N/R	0.04122	N/R	0.04122	0.04530	0.05406

Table 5.4 Testing time calculations for $(B - \frac{A^2}{4})$.

As shown in Table 5.3 and Table 5.4 , a great improvement on testing time calculation has been made by using the numerical solution for the case of negative value of $(B - \frac{A^2}{4})$.

Note that no result (N/R marked in Table 5.4) was generated for the stuck-at 0 fault at L2. This may mean that testing time is beyond the upper bound set up (i.e. 50ms) of the numerical method function program.

The results 50 ms in Table 5.4 is an approximation . We may have a more accurate approximation by setting up a higher number than the present one (10,000) for 'Number of Iterations' in the numerical method function program. However, this will make the numerical method function program slower.

5.5.3 Example 3:

Consider a machine M whose state table is given as Table 5.5.

Present State	Next State	
	x=0	x=1
S_0	S_2,0	S_7,1
S_1	S_3,1	S_6,0
S_2	S_4,0	S_1,3
S_3	S_5,1	S_0,2
S_4	S_6,0	S_3,1
S_5	S_7,1	S_2,0
S_6	S_0,2	S_5,1
S_7	S_1,3	S_4,0

Table 5.5 State table of machine M.

From the table, we can easily find two SP partitions π_a and π_b satisfying the necessary condition of the parallel decomposition for the sequential machines, such as $\pi_a \cdot \pi_b = \pi_0$. These partitions are:

$$\pi_a = (\overline{S_0, S_1}, \overline{S_2, S_3}, \overline{S_4, S_5}, \overline{S_6, S_7})$$
$$\pi_b = (\overline{S_0, S_2, S_4, S_6}, \overline{S_1, S_3, S_5, S_7})$$

If we assign a new state for each block of two SP partitions π_a and π_b, two component machines for the parallel decomposition of machine M can be built as specified in Table 5.6 and Table 5.7.

Present State	Next State, z1	
	x=0	x=1
A	B,0	D,0
B	C,0	A,1
C	D,0	B,0
D	A1,	C,0

Table 5.6 State table of machine M_1.

Present State	Next State, z2	
	x=0	x=1
E	E,0	F,1
F	F,1	E,0

Table 5.7 State table of machine M_2.

A state assignment based on the numerical order is now shown in Table 5.9., where Q_A, Q_B, and Q_C are the states of flip-flops.

Present State	$Q_AQ_BQ_C$	Next State, z= z1+z2 x=0 $Q_AQ_BQ_C$,z		x=1 $Q_AQ_BQ_C$,z	
S_0	0 0 0	S_2	0 1 0, 0	S_7	1 1 1, 1
S_1	0 0 1	S_3	0 1 1, 1	S_6	1 1 0, 0
S_2	0 1 0	S_4	1 0 0, 0	S_1	0 0 1, 3
S_3	0 1 1	S_5	1 0 1, 1	S_0	0 0 0, 2
S_4	1 0 0	S_6	1 1 0, 0	S_3	0 1 1, 1
S_5	1 0 1	S_7	1 1 1, 1	S_2	0 1 0, 0
S_6	1 1 0	S_0	0 0 0, 2	S_5	1 0 1, 1
S_7	1 1 1	S_1	0 0 1, 3	S_4	1 0 0, 0

Table 5.8 State assignment of machine M.

Using J-K flip-flops, we can design a sequential circuit as shown in Fig. 5.4, to which the parallel decomposition technique has been implemented.

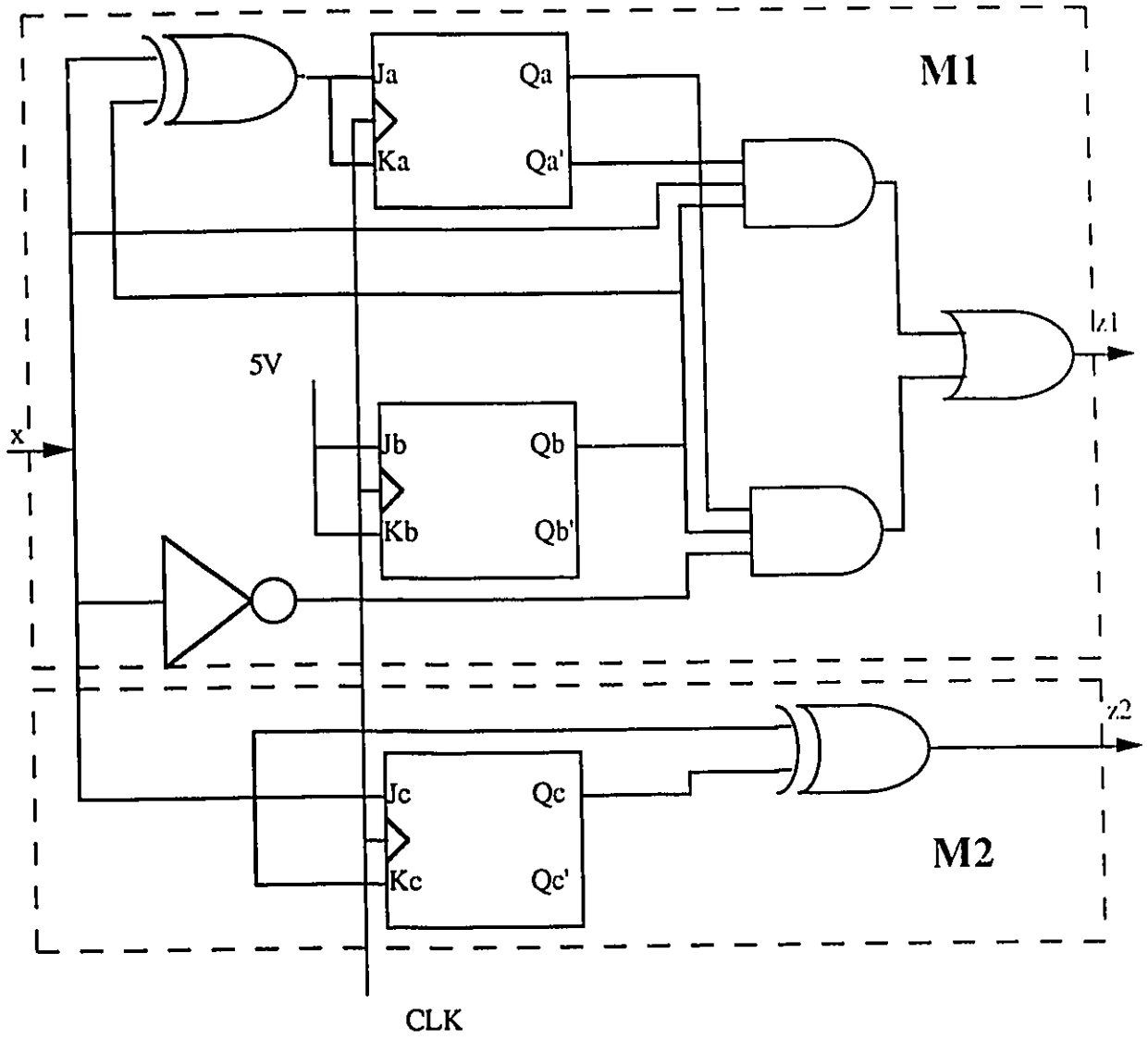


Fig. 5.4 A circuit realization of parallel decomposition of machine M.

We then simulate the circuit with certain faults specified. In this example, we calculate testing time using numerical approach for both cases, i.e. $B - \frac{\Lambda^2}{4} > 0$ and $B - \frac{\Lambda^2}{4} < 0$, and derive accurate testing time for a large number of cases which were not possible before. The results of the testing time calculations will be specified later in this section for comparison with other results which will be generated from a different circuit realization of machine M.

Next, we performed an arbitrary state assignment, as shown in Table 5.10, without the parallel decomposition concept .

Present State		Next State, z= z1+z2			
		x=0		x=1	
	$Q_AQ_BQ_C$		$Q_AQ_BQ_C$,z		$Q_AQ_BQ_C$,z
S_0	0 0 0	S_2	0 1 0, 0	S_7	1 1 0, 1
S_1	0 0 1	S_3	0 1 1, 1	S_6	1 0 1, 0
S_2	0 1 0	S_4	1 0 0, 0	S_1	0 0 1, 3
S_3	0 1 1	S_5	1 1 1, 1	S_0	0 0 0, 2
S_4	1 0 0	S_6	1 0 1, 0	S_3	0 1 1, 1
S_5	1 1 1	S_7	1 1 0, 1	S_2	0 1 0, 0
S_6	1 0 0	S_0	0 0 0, 2	S_5	1 1 1, 1
S_7	1 1 0	S_1	0 0 1, 3	S_4	1 0 0, 0

Table 5.9 An arbitrary state assignment of machine M.

Again, using J-K flip-flops, we designed a sequential circuit to realize the machine M. The schematic drawing is not included in this thesis since it has too many gates and interconnections. However, we can simulate the circuit with certain faults specified at the signal lines corresponding to the ones in Fig. 5.4. The results of testing time calculations for both circuit realization are shown in Table 5.10, where the third column shows testing time calculations of the first circuit realization (Parallel Decomposition) and the fourth one for the second realization (Arbitrary Assignment). For P(1) in second column, we selected a P(1), a reasonably stable result from the 10 simulation runs, i.e. no severe result fluctuations between neighboring P(1)s.

Faults	P(1)	Testing Time (ms) Parallel Decomposition	Testing Time (ms) Arbitrary Assignment	Testing Time Ratio
J_a:s-a-0	0.5	0.03688	0.06471	0.56993
J_a:s-a-1	0.4	0.03121	0.09032	0.34555
K_a:s-a-0	0.5	0.03688	0.07423	0.49683
K_a:s-a-1	0.4	0.03121	0.08410	0.37111
J_b:s-a-0	0.3	0.05027	0.07529	0.66768
J_b:s-a-1	0.6	N/A	0.06616	N/A
K_b:s-a-0	0.3	0.05027	0.07736	0.64982
K_b:s-a-1	0.6	N/A	0.09156	N/A
J_c:s-a-0	0.5	0.03690*	0.06052	0.60972
K_c:s-a-0	0.5	0.03690*	0.06068	0.60811
K_c:s-a-1	0.3	0.03809*	0.08425	0.45211
Q_a:s-a-0	0.4	0.03876	0.08453	0.45854
Q_a:s-a-1	0.6	0.03729	0.06202	0.60126
Q_b:s-a-0	0.4	0.03699	0.06871	0.53835
Q_b:s-a-1	0.8	0.03701	0.08651	0.42732
Q_c:s-a-0	0.7	0.03027	0.06117	0.49485
Q_c:s-a-1	0.6	0.03282	0.06013	0.54582
Q_a':s-a-0	0.4	0.03147	N/R	N/A
Q_a':s-a-1	0.5	0.03753	0.06883	0.54526
x:s-a-0	0.5	0.03690*	N/R	N/A
x:s-a-1	N/A	N/R	N/R	N/A
Total		*0.59928*	*1.16734*	*0.51337*

Table 5.10 Testing time comparison of two circuit realizations.

In Table 5.10, an * (asterisk) indicates the result obtained when one of the component machines is simulated separately from the composite machine. In some cases, if testing times are not calculated (N/R, which means no result, in Table 5.10) by simulating the composite machine,we may obtain testing times by simulating the component machines separately.

As shown in Table 5.10, by using the parallel decomposition technique in a sequential realization, we will have a testing time saving of amount approximately 48% as compared to the circuit realized with an arbitrary state assignment.

The testing time saving will become even greater when realizing a circuit for large-state sequential machine. For example, if it takes total 25 ms to test a set of signal lines on a circuit realizing a 50 state sequential machine, testing time of the parallel decomposition circuit will just be 13 ms.

Generally, the parallel decomposable circuit has fewer interconnections than other circuits. Therefore, the total testing time for interconnections will also be smaller.

5.5.4 Example 4: DM74LS191

Consider a 4-bit up/down counter, 74LS191. Its logic diagram is shown in Fig.5.5. The functional description of this device is well explained in any IC manufacturer's data book.

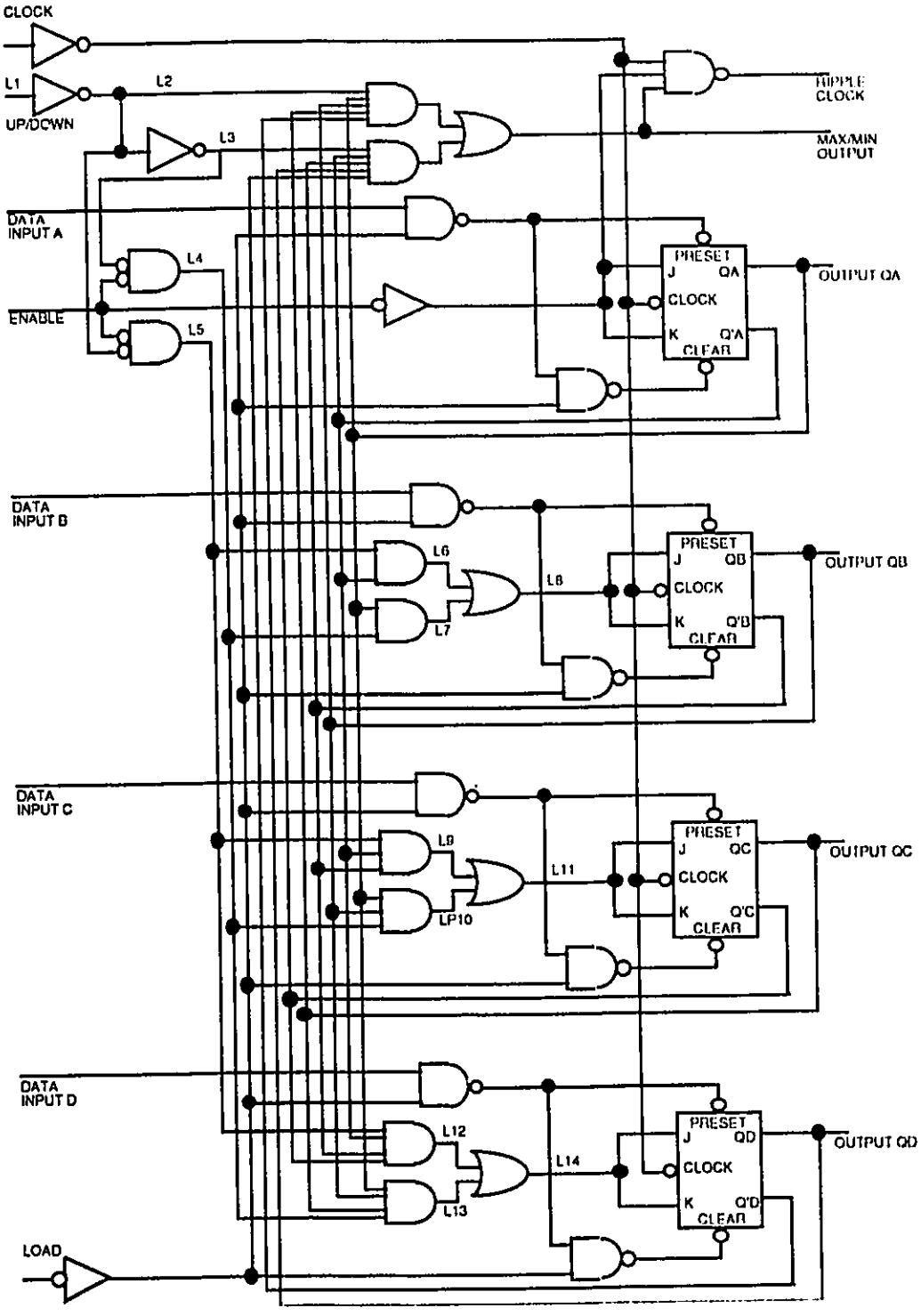


Fig.5.5 Logic diagram of LS191.

For the simulation, the ENABLE input line was set LOW so that this device could function as a counter, and the LOAD input line was set HIGH so that the device could count continuously. The random generator was connected to the UP/DOWN input line.

We simulated the device with the assignment of stuck-at faults on lines L1 through L14 as shown in Fig.5.5.

Then we partitioned the device into two component circuits: Partition A-B and Partition C-D. Partition A-B consisted of flip-flops Q_A and Q_B together with their related logic gates. Similarly, Partition C-D consisted of flip-flops Q_C and Q_D together with their related logic gates. We simulated each partition with the same stuck-at faults at corresponding lines in the original device. Finally we compared the maximum testing times obtained by the simulations.

The fault-free state table of LS 191 obtained from our simulation program is shown in Table 5.11.

Present State	Next State	
	x=0	x=1
S_0	$S_1,0$	$S_{15},0$
S_1	$S_2,1$	$S_0,1$
S_2	$S_3,2$	$S_1,2$
S_3	$S_4,3$	$S_2,3$
S_4	$S_5,4$	$S_3,4$
S_5	$S_6,5$	$S_4,5$
S_6	$S_7,6$	$S_5,6$
S_7	$S_8,7$	$S_6,7$
S_8	$S_9,8$	$S_7,8$
S_9	$S_{10},9$	$S_8,9$
S_{10}	$S_{11},10$	$S_9,10$
S_{11}	$S_{12},11$	$S_{10},11$
S_{12}	$S_{13},12$	$S_{11},12$
S_{13}	$S_{14},13$	$S_{12},13$
S_{14}	$S_{15},14$	$S_{13},14$
S_{15}	$S_0,15$	$S_{14},15$

Table 5.11 Fault free table of LS 191 simulation.

The fault-free state table of the two partitions of LS 191, Partition A-B and Partition C-D, are shown in Table 5.12 and Table 5.13, respectively.

Present State	Next State	
	x=0	x=1
S_0	S_1,0	S_3,0
S_1	S_2,1	S_0,1
S_2	S_3,2	S_1,2
S_3	S_0,3	S_2,3

Table 5.12 Fault-free table of partition A-B of LS 191 simulation.

Present State	Next State	
	x=0	x=1
S_0	S_0,0	S_3,0
S_1	S_1,1	S_0,1
S_2	S_2,2	S_1,2
S_3	S_3,3	S_2,3

Table 5.13 Fault-free table of partition C-D of LS 191 simulation.

Results of simulations are presented in Table 5.14. As shown in the table, testing times in the partitioned case are approximately 10 times faster than those in the non-partitioned case.

Fault at Line L_j j=1 ..14	Equiv. fault to Line L_j j=1 ..14	Best P(1) P(1)=0.1 ..1.0	Testing Time (ms) Non-partitioned circuit	Testing Time (ms) Partitioned circuit
L_1:s-a-0	L_2:s-a-1	0.3	0.19189	0.01890
L_1:s-a-1	L_2:s-a-0	0.8	0.18575	0.08180
L_2:s-a-0	L_1:s-a-1	0.8	0.18575	0.08180
L_2:s-a-1	L_1:s-a-0	0.3	0.19189	0.01890
L_3:s-a-0	L_4:s-a-1	0.9	0.04970	0.00339
L_3:s-a-1	L_4:s-a-0	0.1	0.04937	0.00338
L_4:s-a-0	L_3:s-a-1	0.1	0.04937	0.00338
L_4:s-a-1	L_3:s-a-0	0.9	0.04970	0.00339
L_5:s-a-0	L_6:s-a-0	0.9	0.05016	0.00340
L_5:s-a-1	N/A	0.1	0.04986	0.00340
L_6:s-a-0	L_5:s-a-0	0.9	0.05016	0.00340
L_6:s-a-1	L_7:s-a-1	0.1	0.06019	0.00428
L_7:s-a-0	L_3:s-a-1	0.1	0.04937	0.00338
L_7:s-a-1	L_6:s-a-1	0.1	0.06019	0.00428
L_8:s-a-0	N/A	0.2	0.09248	0.00788
L_8:s-a-1	L_7:s-a-1	0.1	0.06019	0.00428
L_9:s-a-0	N/A	0.8	0.06236	0.05894
L_9:s-a-1	L_{10}:s-a-1	0.2	0.06087	0.05939
L_{10}:s-a-0	N/A	0.2	0.06216	0.05881
L_{10}:s-a-1	L_9:s-a-1	0.2	0.06087	0.05939
L_{11}:s-a-0	N/A	0.8	0.09534	0.05894
L_{11}:s-a-1	L_{10}:s-a-1	0.2	0.06087	0.05939
L_{12}:s-a-0	N/A	0.9	0.04943	0.00339
L_{12}:s-a-1	L_{13}:s-a-1	0.9	0.03097	0.00357
L_{13}:s-a-0	N/A	0.1	0.04978	0.00339
L_{13}:s-a-1	L_{12}:s-a-1	0.9	0.03097	0.00357
L_{14}:s-a-0	N/A	0.9	0.05996	0.00339
L_{14}:s-a-1	L_{13}:s-a-1	0.9	0.03097	0.00357

Table 5.14 Testing Time Comparisons between Partitions and Original Circuit simulations for LS191.

5.6 Chapter Summary

A random testing method based on the circuit decomposition approach utilizing the continuous Markov model is presented in this chapter. Benefits gained from the proposed approach against its counterparts are described. Through the experimentation on four different sequential circuits, we have shown that i) the proposed approach is feasible, ii) the new testing time calculation method is an improvement over the existing method, iii) there can be great savings in testing time in the case of decomposed circuits as compared to the nondecomposed circuits.

CHAPTER 6

CONCLUSIONS

In this thesis, we proposed a simple random testing approach, based on the well-known "divide and conquer" principle, for sequential circuits. Given a large sequential circuit, we decomposed it into several smaller partitions using state substitution property. Each partition became a sequential circuit with smaller number of states than the original circuit, thus making it easy to analyze each partition separately. We then analyzed the problem of detecting permanent faults in the partitions by random testing, using a three-state continuous parameter Markov model. Given a sequential circuit in a partition with certain faults specified, the original state table and its error version were derived from an analysis of the partition under fault-free and faulty conditions, respectively. By simulation of these two tables on a computer, the parameters of the desired Markov model were obtained. For a specified degree of confidence, it was fairly straightforward to derive the parameters of the model, and to calculate the lengths of the test-input patterns required for fault detection in a partition. Using the model, it was also possible to determine the maximum testing time needed so that the probability of a wrong conclusion is smaller than or equal to some prespecified confidence value α.

The developed model and the related mathematical analysis provide some useful insights into the nature of faults in relation to random testing in a sequential digital circuit and the associated confidence measure.

In Chapter 2, various fault detection strategies based on deterministic and probabilistic methods for sequential circuits were surveyed. Furthermore, automatic test pattern generation approach was also discussed.

In Chapter 3, the principles and types of decompositions of sequential circuits were discussed. Important properties for decomposition theory were discussed in detail. Based on the decomposition theory, two types of decompositions, such as serial and parallel decompositions were discussed with examples.

In Chapter 4, a continuous Markov model of faulty sequential machines was defined and the differential equations of the model were derived. Markov parameters and the state probabilities were also derived. Testing time calculation, using both the closed form solution and the numerical approximation, was discussed in two separate cases: (i) $(B - \frac{A^2}{4}) > 0$ and (ii) $(B - \frac{A^2}{4}) < 0$. A problem in simulation was solved using a numerical method for solving the nonlinear equations.

In Chapter 5, a random testing method based on the circuit decomposition approach utilizing the continuous Markov model was presented. Benefits gained from the proposed approach against its counterparts were described. Through the experimentation on three different sequential circuits, we had shown that i) the proposed approach is feasible, ii) the new testing time calculation method is an improvement over the existing method, and iii) there can be great savings in testing time in the case of decomposed circuits as compared to the nondecomposed circuits.

BIBLIOGRAPHY

[1] M.S. Abadir and M.A. Breuer, "A Knowledge-Based System for Designing Testable VLSI Chips", *IEEE Design and Test* , pp. 56-68, August 1985.

[2] P. Agrawal and V. D. Agrawal, "Probabilistic Analysis of Random Test Generation Method for Irredundant Combinational Logic Networks", *IEEE Transactions on Computers*, pp. 691-694, July 1975.

[3] P. Agrawal and V. D. Agrawal, "On Improving the Efficiency of Monte Carlo Test Generation", *1975 Int. Symp. on Fault-Tolerant Computing Digest (FTCS-5)*, pp. 205-209, 1975.

[4] P. Agrawal and V.D. Agrawal, "On Monte Carlo Testing of Logic Tree Networks", *IEEE Transactions on Computers*, pp. 664-667, June 1976.

[5] V. D. Agrawal, "When to Use Random Testing", *IEEE Transactions on Computers*, pp. 1054-1055, Nov. 1978.

[6] V. D. Agrawal, S. K. Jain and D. M. Singer, "Automation in Design for Testability", *Proc. Custom Integrated Circuit Conference*, May 1984.

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

[8] V.K. Agrawal, "Multiple Fault Detection in Programmable Logic Arrays", *IEEE Transactions on Computers*, pp. 518-522, June 1980.

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

[10] D. Bastin, E. Girard, J. C. Rault and R. Tullioue, "Probabilistic Test Generation Methods : Statistical Estimation of Sequence Length and Efficiency", *1973 Int. Symp. on Fault-Tolerant Computing Digest (FTCS-3)*, p. 171, 1973.

[11] M. A. Breuer, "A Random and an Algorithmic Technique for Fault Detection Test Generation for Sequential Circuits", *IEEE Transactions on Computers*, pp. 1364-1370, Nov. 1971.

[12] M. A. Breuer, "Testing for Intermittent Faults in Digital Circuits", *IEEE Transactions on Computer*, pp. 241-246, March 1973.

[13] I. Berger and Z. Kohavi, "Fault-Detection in Fan-out-free Combinational Networks", *IEEE Transactions on Computers*, pp. 908-914, Oct. 1973.

[14] P.C. Chandrasekharon and B. Thomas, "Test Sequence Reduction for Large-State Sequential Machines through Parallel Decomposition", *Zeitschrift für electrische Informations. und Energietechnik*, Jahrg. 1-13; 1971-1983, Leipzig, Geest & Portig, pp. 349-358, 1981.

[15] W.T. Cheng and T.J. Chakraborty, "Gentest: An Automatic Test-Generation System for Sequential Circuits", *IEEE Computer*, pp. 43-49, April 1989.

[16] K.T. Chung and V.D. Agrawal, "Unified Methods for VLSI Simulation and Test Generation", Kluwar Academic Publishers, 1989.

[17] F. W, Clegg, "Use of SPOOF's in the Analysis of Faulty Logic Networks", *IEEE Transactions on Computers*, pp. 229-234, Oct. 1973.

[18] P. Das and D.E. Farmer, "Fault-Detection Experiments for Parallel-Decomposable Sequential Machines", *IEEE Transactions on Computers*, pp. 1104-1109, Nov. 1975.

[19] S. R. Das, P. K. Srimani and C. R. Datta, "On Multiple Fault Analysis in Combinational Circuits by Means of Boolean Difference", *Proceedings of IEEE*, pp. 1447-1449, Sept. 1976.

[20] S. R. Das and A. Bhattacharyya, "Fault Detection in Sequential Machines with Increased Fault Coverage", *Electronics Letters*, pp. 28-29, Jan. 1978.

[21] S. R. Das, C. L. Sheng and Z. Chen, "An Algorithm for Finding all Maximal Complete Subgraphs and an Estimate of the Order of Computational Complexity", *Computers and Electrical Engineering*, pp. 365-368, Dec. 1978.

[22] S. R. Das, Z. Chen, S. M. Wu, S. Y. Lee and A. Bhattacharyya, "On the Design of Improved Failure Detection Experiments in Synchronous Sequential Machines Based on Terminal Measurements", *Computers and Electrical Engineering*, pp. 293-297, Dec. 1979.

[23] S. R. Das and W. B. Jone, "Modified Transition Matrix and Fault Testing in Sequential Logic Circuits under Random Stimuli with a Specified Measure of Confidence", *Cybernetics and Systems : An International Journal*, pp. 1-12, 1986.

[24] S. R. Das, "Random Test Generation for LSI/VLSI Circuits - A Survey", *IEEE VLSI Technical Bulletin*, pp. 4-11, Mar. 1987.

[25] S. R. Das, K. W. Chiang and W. B. Jone, "A First-Order Optimal Algorithm for State Identification in Sequential Logic using the Concept of Entropy", *Cybernetics and Systems : An International Journal*, pp. 251-270, Sept. 1987.

[26] S. R. Das, "Adaptive Scheduled Experimentation and Fault Location in Large Combinational Logic Networks", *Cybernetics and Systems : An International Journal*, pp. 1-12, Jan. 1988.

[27] S. R. Das, "Nontransient Fault Testing in Sequential Logic Circuits using Stochastic Modeling and Simulation", *Cybernetics and Systems : An International Journal*, pp. 15-27, Jan. 1989.

[28] S. R. Das, W. B. Jone, G. E. Fares and A. R. Nayak, "Probabilistic Fault Location in Combinational Logic Network using Concepts of Fault Distance and Input Feature", *Cybernetics and Systems : An International Journal*, pp. 385-399, Dec. 1989.

[29] S. R. Das, W. B. Jone and K. L. Wong, "Probabilisitc Modeling and Fault Analysis in Sequential Logic using Computer Simulation", *IEEE Transactions on Systems, Man and Cybernetics*, pp. 490-498, Mar./Apr. 1990.

[30] R. David and G. Blanchet, "About Random Fault Detection of Combinational Network", *IEEE Transactions on Computers*, pp. 659-664, June 1976.

[31] R. David and R. Tellez-Giron, "Comments on 'The Error Latency of a Fault in a Sequential Digital Circuit'", *IEEE Transactions on Computers*, pp. 85-86, Jan. 1979.

[32] R. David and P. Thevenod-Fosse, "Random Testing of Integrated Circuits", *IEEE Transactions on Computers*, pp. 20-25, March 1981.

[33] R. David and P. Thevenod-Fosse, "Minimal Detecting Transition Sequences : Application to Random Testing", *IEEE Transactions on Computers*, pp. 514-518, June 1980.

[34] S. Devadas, H. K. Ma, A. R. Newton, and Sangiovanni-Vincenttelli, "Irredundent Sequential Machines via Optimal Logic Synthesis", *IEEE Transactions on Computer-aided Design*, pp. 8-18, Jan. 1990.

[35] J. L. Doob, "Stochastic Processes", John Wiley & Sons Inc., NJ., 1958.

[36] E. B. Eichelberger and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test", *IBM J. Res. Develop.*, pp. 265-272, May 1983.

[37] R. R. Fritzemeier, H. T. Nagle, and C. F. Hawkins, " Fundamental of Testability : A Tutorial", *IEEE Transactions on Industrial Electronics*, Vol. 36, No. 2, pp. 117-127, May 1989.

[38] S. W. Golomb, "Shift Register Sequences", revised ed., Laguna Hills, CA: Agegean Park, 1982.

[39] J. Hartmanis and R.E. Stearns, "Algebraic Structure Theory of Sequential Machines", Prentice-Hall Inc., NJ, 1966.

[40] C. F. Hawkins, H. T. Nagle, R. R. Fritzemeier, and J. R. Guth, " The VLSI Circuit Test Problem-A Tutorial", *IEEE Transactions on Industrial Electronics*, Vol. 36, pp. 111-116, May 1989.

[41] S.K. Jain and V.D. Agrawal, "Statistical Fault Analysis", *IEEE Design and Test of Computers*, Vol. 2, pp. 38-44, Feb. 1985.

[42] S. Kamal and C. V. Page, "Intermittent Faults : A Model and a Detection Procedure", *IEEE Transactions on Computers*, pp. 713-719, July 1974.

[43] S. Kamal, "An Approach to the Diagnosis of Intermittent Faults", *IEEE Transactions on Computers*, pp. 461-467, May 1975.

[44] Z. Kohavi, "Switching and Finite Automata Theory", McGraw-Hill, New York, NY, 1978 (Second Edition).

[45] M.C. Kohn, "Practical Numerical Methods: Algorithms and Programs", Macmillan Publishing Company, NY, 1986.

[46] I.Koren and Z. Kohavi, "Diagnosis of Intermittent Faults in Combinational Networks", *IEEE Transactions on Computers*, pp. 1154-1158, Nov. 1977.

[47] P. K. Lala, "Fault Tolerant and Fault Testable Hardware Design", Prentice-Hall Inc., London, 1985.

[48] S. C. Lee, "Digital Circuits and Logic Design", Prentice-Hall Inc., Englewood Cliffs, N.J., 1976.

[49] J. Losq, "Efficiency of Random Compact Testing", *IEEE Transactions on Computers*, pp. 516-525, June 1978.

[50] H.T. Ma, S. Devadas, A.R. Newton and A. Sangiovanni-Vincetelli, "Test generation for sequential circuits", *IEEE Trans. Computer-Aided Design*, Vol. 7, pp. 1081-1093, Oct. 1988.

[51] R. Marlett, "An Effective Test Generation System for Sequential Circuits", *23rd Design Automation Conference*, Las Vegas, Nevada, pp. 250-256, June 1986.

[52] J. F. Meyer and K. Yeh, "Diagnosable Machine Realizations of Sequential Behavior", *1971 Int. Symp. on Fault-Tolerant Computing Digest (FTCS-1)*, pp. 22-25, 1971.

[53] G.D. Micheli, R.K. Brayton and A. Sangiovanni-Vincentelli, "Optimum State Assignment for Finite State Machines" *IEEE Transactions on Computer-Aided Design*, pp 269-285, July 1985.

[54] E. I. Muehldorf and A. D. Savkar, "LSI Logic Testing - An Overview", *IEEE Transactions on Computers*, pp. 1-17, Jan. 1981.

[55] P. Muth, "A Nine-Valued Circuit Model for Test Generation", *IEEE Transactions on Computers*, pp. 630-636, June 1976.

[56] K. P. Parker, "Probabilistic Test Generation", *1973 Int. Symp. on Fault-Tolerant Computing Digest (FTCS-3)*, p. 179, 1973.

[57] K. P. Parker and E. J. McCluskey, "Analysis of Logic Circuits with Faults using Input Signal Probabilities", *IEEE Transactions on Computers*, pp. 573-578, May 1975.

[58] K. P. Parker and E. J. McCluskey, "Probabilistic Treatment of General Combinational Networks", *IEEE Transactions on Computers*, pp. 668-670, June 1975.

[59] J. C. Rault, "A Graph Theoretical and Probabilistic Approach to the Fault Detection of Digital Circuits", *1971 Int. Symp. on Fault-Tolerant Computing Digest (FTCS-1)*, pp. 26-29, 1971.

[60] J. P. Roth, W.G. Bouricius and P.R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits", *IEEE Transactions on Electon. Comput.*, vol. EC-16, pp. 567-580, Oct. 1967.

[61] J. Savir, "Optimal Random Testing of Single Intermittent Failures in Combinational Circuits", *1977 Int. Symp. on Fault-Tolerant Computing Digest (FTCS-7)*, pp. 180-185, June 1977.

[62] J. Savir, G. S. Ditlow, and P. H. Bardell, "Random Pattern Testability", *IEEE Transactions on Computers*, pp. 79-90, Jan. 1984.

[63] J. Savir and P. H. Bardell, "On Random Pattern Test Length", *IEEE Transactions on Computers*, pp. 467-474, June 1984.

[64] P. B. Schneck, "Comment on 'When to Use Random Testing'", *IEEE Transactions on Computers*, pp. 580-581, Aug. 1979.

[65] H. D. Schnurmann, E. Lindbloom and R. G. Carpenter, "The Weighted Random Test-Pattern Generator", *IEEE Transactions on Computers*, pp. 695-700, July 1975.

[66] J. J. Shedletsky and E. J. McCluskey, "The Error Latency of a Fault in a Combinational Digital Circuit", *1975 Int. Symp. on Fault-Tolerant Computing Digest (FTCS-5)*, pp. 210-214, June 1975.

[67] J. J. Shedletsky and E. J. McCluskey, "The Error Latency of a Fault in a Sequential Digital Circuit", *IEEE Transactions on Computers*, pp.655-659, June 1976.

[68] K. G. Shin and Y. H. Lee, "Measurement and Application of Fault Latency", *IEEE Transactions on Computers*, pp. 370-374, April 1986.

[69] D. P. Siewiorek and L. K. W. Lai, "Testing of Digital Systems", *IEEE Transactions on Computers*, pp. 1321-1333, Oct. 1981.

[70] S. Y. H. Su, I. Koren, and Y. K. Malaiya, "A Continuous-Parameter Markov Model and Detection Procedures for Intermittent Faults", *IEEE Transactions on Computers*, pp. 567-570, June 1978.

[71] L. Takacs, "Stochastic Processes, Problems and Solutions", John Wiley & Sons Inc., N.J., 1960.

[72] P. Thevenod-Fosse, "Asynchronous Random Testing of Sequential Circuits", *1978 Int. Symp. on Fault-Tolerant Computing Digest (FTCS-8)*, p. 213, 1978.

APPENDIX

/* This is a 'C' program designed for the simulations of the circuit of the example 3 in Chapter 5, experimentation part. */

```
#include <math.h>
#define MAX_INT 2147483646
        /* 2^31 - 1 = 2147483646: THE MAXIMUM INTEGER FROM THE RANDOM
        NUMBER GENERATOR.*/
#define MAX_LENGTH 100000
        /* The NUMBER OF INPUT PATTERNS TO BE APPLIED.*/
#define NUMTEST 11
        /*  Number of Test time calculation, optional */
#define resolution 0.1
        /*  resolution of p(1) depends on NUMTEST value  */
#define mask 1
        /* To mask all the bits except LSB in the calculation for the construction of the stable
        of circuit to be simulated. */
int FFT[8][2],FFO[8][2],FT[8][2],FO[8][2];
        /* FFT = STATE TRANSITIONS FOR FAULT FREE CIRCUIT    */
        /* FFO = OUTPUTS  FOR FAULT FREE CIRCUIT */
        /* FT  = STATE TRANSITIONS FOR FAULTY CIRCUIT */
        /* FO  = OUTPUTS   FOR FAULTY CIRCUIT */
        /* EACH ARRAY HAS 8 STATE ARAY AND 2 input ELEMENTS.  */
int c[3][3];
        /*COUNTERS FOR MARKOV STATE TRANSITION (C_ij). IT HAS 3 PRESENT
        CIRCUIT  STATE ARRAY AND 3 NEXT STATES ELEMENTS*/
double T_array[NUMTEST];
        /* Keeps NUMTEST testing times corresponding to each P(1)'s*/
**************************************************************************
main()
{
        extern int FFT[8][2],FFO[8][2],FT[8][2],FO[8][2];
        int a,b,c,x,aN,bN,cN,ja,jb,jc,ka,kb,kc;
        int fa,fb,faN,fbN,fja,fjb,fka,fkb,fc,fcN,fjc,fkc;
        int i,j;
```

```
        /*   a,b,c: the presernt state of J-K flip-flop A,B,C */
        /*   x: the input of circuit */
        /*   aN,bN,cN: the next state of J-K flip-flop A ,B,C */
        /*   ja,jb,jc: J input of J-K flip-flop A,B,C*/
        /*   ka,kb,kc: K input of J-K flip-flop A,B,C*/
        /*   The prefix 'f' in the variables represents for the variable of faulty circuit  */
        /*   e.g 'fa' means present state of J-K flip-flop A of  fault circuit.            */

double p1;        /*  PROBABILITY OF 1 FOR RANDOM INPUT PATTERNS  */
int seed;         /*  seed for the random number generator  */
int ip1;          /*  index of P(1)  */

        /* This part of the program constructs state tables for fault free circuit  and faulty
        circuit. */
for (a=0,fa=0;a<2,fa<2;a++,fa++)
 {
  for (b=0,fb=0;b<2,fb<2;b++,fb++)
   {
     for (c=0,fc=0;c<2,fc<2;c++,fc++)
      {
        for (x=0;x<2;x++){
         ja=c;
         fja=1;
         jb=x & (mask&~a);
         fjb=x & (mask&~fa);
         ka=(mask&~x) | (mask&~b);
         fka=(mask&~x) | (mask&~fb);
         kb=(mask&~x) | a;
         fkb=(mask&~x) | fa;
         jc=x&b;
         fjc=x&fb;
         kc=0;
         fkc=0;
         aN=ja&(mask&~a) | (mask&~ka)&a;
         faN=fja&(mask&~fa) | (mask&~fka)&fa;
```

```
            bN=jb&(mask&~b) | (mask&~kb)&b;
            fbN=fjb&(mask&~fb) | (mask&~fkb)&fb;
            cN=jc&(mask&~c) | (mask&~kc)&c;
            fcN=fjc&(mask&~fc) | (mask&~fkc)&fc;
            FFT[a*4+b*2+c][x]=aN*4 + bN*2 +cN;
            FT[fa*4+fb*2+fc][x]=faN*4 +fbN*2+fcN;
            FFO[a*4+b*2+c][x]=x & a & (~b);
            FFO[a*4+b*2+c][x]=x & fa & (~fb);
        }
       }
    }
}


/*Print out fault-free and output transition table*/
printf("< Fault Free Transition & output Table >\n"); printf("\nPresent");
printf("\n State     X=0     X=1");
for (i=0;i<8;i++)
{
printf("\n");
printf("S%d",i);
for (j=0;j<2;j++)
printf("\t   S%d,%d",FFT[i][j],FFO[i][j]);
}

/*print out Faulty transition output table*/
printf("\n\n\n\n< Faulty  Transition and Output Table >\n"); printf("\nPresent");
printf("\n State     X=0     X=1");
for (i=0;i<8;i++)
{
printf("\n");
printf("S%d",i);
for (j=0;j<2;j++)
printf("\t   S%d,%d",FT[i][j],FO[i][j]);
}
printf("\n\n\n\n\n\n");
```

```
seed=2345;
printf("seed = %d\n",seed);

/*For each different value of PROBABILITY OF 1 OF RANDOM INPUT PATTERNS*/
for(ip1=0;ip1<NUMTEST;ip1++)
{
p1=ip1*resolution;
printf("P(1) = %lf\n",p1);
simu(seed,ip1);        /* function call */
Markov(ip1);
Testing_time(ip1);
}
Optimum(ip1);
}              /* End of program*/

*************************************************************************

/*THE FUNCTION 'SIMU' CALCULATE NINE DIFFERENT COUNTERS.*/ /*THIS
COUNTS WILL BE USING FOR CALCULATING THE MARKOV  */ /*CHAIN
PARAMETERS LATER.                                  */

simu(seed,ip1)
int seed;
int ip1;   /*PROBABILITY OF 1 FOR RANDOM INPUT PATTERNS X 10 */
{
extern int FFT[8][2],FFO[8][2],FT[8][2],FO[8][2],c[3][3];
 int P_STATE,N_STATE;    /*MARKOV CHAIN STATES, P=PRESENT AND
N=NEXT*/
int PFF_STATE,NFF_STATE,PF_STATE,NF_STATE;
/*PRESENT AND NEXT STATE FOR FAULTY AND FAULT FREE CRCUITS.*/
int I,J;                    /*  INDEX FOR COUNTERS  */
int num,K;                  /*  FOR RANDOM GENERATOR */
int RAND_BIT;               /*  0 OR 1 FROM RANDOM GENERATOR */
int F_OUT,FF_OUT;           /*  FAULTY & FAULT FREE OUTPUT */
int LIMIT;                  /*  THRESHOLD FOR 1 OR 0 */
double p1=ip1*resolution;   /*  Probability of 1   */
```

```
/*THIS SECTION INITIALIZES COUNTERS FOR MARKOV STATE
TRANSITIONS*/ for(I=0;I<3;I++)
{
for(J=0;J<3;J++)
c[I][J]=0;
}

/*INITIALIZING MARKOV STATES*/
P_STATE=0;
PFF_STATE=0;
PF_STATE=0;

/*Determination of 1 or 0 from random number generator */

LIMIT=p1*MAX_INT;

srand(seed);
for(K=0;K<MAX_LENGTH;K++)
{
num=rand();          /* Get a random number. */
if(num<LIMIT)        /* Decide 1 or 0        */
   RAND_BIT=1;
else
   RAND_BIT=0;

NFF_STATE=FFT[PFF_STATE][RAND_BIT];      /*  STATE MAPPING   */
NF_STATE=FT[PF_STATE][RAND_BIT];

FF_OUT=FFO[PFF_STATE][RAND_BIT];         /*  OUTPUT MAPPING  */
F_OUT=FO[PF_STATE][RAND_BIT];

/*DETERMINING MARKOV NEXT STATE*/
if(F_OUT != FF_OUT)
{
N_STATE=1;          /*  CCT STATE #1   */
```

```
c[P_STATE][1]++;     /*   INCREMENTING COUNTER  */
}
else if(NF_STATE!=NFF_STATE)
{
N_STATE=2;           /*   CCT STATE #2  */
c[P_STATE][2]++;
}
else
{
N_STATE=0;           /*  CCT STATE #0  */
c[P_STATE][0]++;
}
/*UPDATING PFF_STATE,PF_STATE AND P_STATE*/
PFF_STATE=NFF_STATE;
PF_STATE=NF_STATE;
P_STATE=N_STATE;
}      /*END OF FOR LOOP*/

/*OUTPUT*/
for(I=0;I<3;I++)
{
for(J=0;J<3;J++)
printf("C%1D%1D=%-15d",I,J,c[I][J]);
printf("\n");
}
}  /*END OF function simu()*/
*************************************************************************
/*This function infers the Markov parameters using transition frequency counts and finds
the semi-optimum P(1) for testing time calculation.*/

double A,B;
double C0,D0,C1,D1,C2,D2;
double C0_,D0_,C1_,D1_,C2_,D2_;          /* _ means prime eg. C0_:C0 prime */
double C0__,D0__,C1__,D1__,C2__,D2__;    /* __ means double prime. */
double condition;                        /* eg. C0__:C0 double prime */
```

```
#define scale 1000                    /* test patterns per unit time */

Markov(ip1)
int ip1;
{
extern int c[3][3];
int i;
double l[3],mu[3];      /*  l means lambda  */

for(i=0;i<3;i++)
{
l[i]=c[i][(i+1)%3];     /*  scaling factor is 1000  */
printf("lambda%d=%-15.3lf",i,l[i]/scale);
}

for(i=0;i<3;i++)
{
mu[i]=c[(i+1)%3][i];
printf("mu%d=%-15.3lf",i,mu[i]/scale);
}
printf("\n");

/*The following parameters are being used for calculating of Testing Time for both cases.
i.e. B - A*A/4 > 0 and B - A*A/4 < 0  */

C0=(l[1]+l[2]+mu[0]+mu[1])/scale;
D0=(l[1]*l[2]+l[2]*mu[0]+mu[0]*mu[1])/(scale*scale);

C1=l[0]/scale;
D1=(l[0]*l[2]+l[0]*mu[1]+mu[1]*mu[2])/(scale*scale);

C2=mu[2]/scale;
D2=(l[0]*l[1]+l[1]*mu[2]+mu[0]*mu[2])/(scale*scale);

C0_=(l[0]+l[2]+mu[2]+mu[1])/scale;
D0_=(l[0]*l[2]+l[0]*mu[1]+mu[2]*mu[1])/(scale*scale);
```

```
C1_=l[1]/scale;
D1_=(l[0]*l[1]+l[1]*mu[2]+mu[0]*mu[2])/(scale*scale);

C2_=mu[0]/scale;
D2_=(l[1]*l[2]+l[2]*mu[0]+mu[0]*mu[1])/(scale*scale);

C0__=(l[0]+l[1]+mu[2]+mu[0])/scale;
D0__=(l[0]*l[1]+l[1]*mu[2]+mu[2]*mu[0])/(scale*scale);

C1__=l[2]/scale;
D1__=(l[2]*l[1]+l[2]*mu[0]+mu[0]*mu[1])/(scale*scale);

C2__=mu[1]/scale;
D2__=(l[0]*l[2]+l[0]*mu[1]+mu[2]*mu[1])/(scale*scale);

A=(l[0]+l[1]+l[2]+mu[0]+mu[1]+mu[2])/scale;
B=(l[0]*l[1]+l[1]*l[2]+l[2]*l[0]+mu[0]*mu[1]+mu[1]*mu[2]+mu[2]*mu[0]
+l[0]*mu[1]+l[1]*mu[2]+l[2]*mu[0])/(scale*scale);

condition=B-(A*A)/4;
printf("B-(A*A)/4=%-15.7lf\n",condition);

/* This section fills each element of Array[index][9] with values calculated under a given
P(1).*/
}       /*end of function of Markov()*/
*****************************************************************

/*This function calculate the testing time.*/
#include <math.h>
#define PR 0.1          /*PR is priori probabilty*/
#define ALPHA 0.05   /*predefined value for confidence= 10 to the power -6*/

Testing_time(ip1)
int ip1;
{
extern double A,B;
```

```
extern double D0,C1,D1,D2;
extern double C2__,D2__;
extern double condition;

double g1,g2,h1,h2;                 /*  For B - A*A/4 > 0  */
double m,n,omega1,omega2,W;   /* For closed form solution */
double log();
double test;
double sneg, spos, snew, f();      /* Numerical solution for B-A*A/4 < 0  and B-A*A/4  */
                                   /* > 0 */
int   iter, imax;
float testing_time;
int index=ip1;
extern double T_array[NUMTEST];

*  In case of B - A*A/4 > 0 */
/*Since the passed value of (B - A*A)/4 is greater than zero, either the numerical or the */
/*closed form solution is enough for calculating testing time.*/

/* (A) Numerical Solution */
 if(condition>0)
{
sneg=0;
spos=50;
imax=10000;

for(iter=0; iter<imax; iter++){
snew = (sneg + spos)/2.0;
if (abs(g(snew)) < 0.001)
break;
if (g(snew) < 0)
sneg = snew;
else
spos = snew;
}
printf("no of iterations, s (Testing Time), g(s) = ");
```

```
printf("%10d %10.5f %10.5f\n", iter, snew, f(snew));
printf("Test time = %7.7f",snew);
printf("\n");
printf("\n");
printf("\n");
T_array[index]=snew;
}  /* end of the case of B - A*A/4 < 0 */
}
*************************************************************************
double g(s) /* Function for numerical analyis when (B - A*A)/4 is greater than zero*/
double s;
{
double  V, X, Y, W;
double  fcn;

V = B - A*A/4.0;
X = ALPHA*B*B/PR - D0*(B-D1) - D2*(B-D2);
Y = D0*D1 + D2*D2__;
W = ((2*B*C1-A*D1)*D0 +(2*B*C2__-A*D2__)*D2)/(2*U);
fcn = 2.0 * X - (Y-Z)*exp ((U-A/2.0)*2)  * (Y*cos (V-s) - W*sin (V*s));
return(fcn);
}  /* end of function g(s) */
*************************************************************************
/* (B) Closed form Solution */
 if(condition>0)
 {
 g1=(-D1)/B;
 g2=(-D2__)/B;

 m=sqrt(condition);
 h1=(2*B*C1-A*D1)/(2*B*m);
 h2=(2*B*C2__-A*D2__)/(2*B*m);

 omega1= sqrt(g1*g1+h1*h1);
 omega2= sqrt(g2*g2+h2*h2);
```

```
W=(D0*omega1+D2*omega2)*(PR/B)/(ALPHA-(PR/(B*B))*(D0*(B-D1)+D2*(B-
D2__))); test=PR/(B*B)*(D0*(B-D1)+D2*(B-D2__));
printf("W=%-15.3lf, and ",W);
printf("test=%-15.3lf\n",test);
n=log(W);
testing_time=(2/A)* n;
printf("Test time = %7.7f",testing_time);
printf("\n");
printf("\n");
printf("\n");
T_array[index]=testir g_time;
}   /* end of the case of B - A*A/4 > 0 */
*****************************************************************
/*  In case of B - A*A/4 < 0, only use a numerical method.*/
else {
sneg=0;
spos=50;
imax=10000;

for(iter=0; iter<imax; iter++){
snew = (sneg + spos)/2.0;
if (abs(f(snew)) < 0.001)
break;
if (f(snew) < 0)
sneg = snew;
else
spos = snew;
}
printf("no of iterations, s, f(s) = ");
printf("%10d %10.5f %10.5f\n", iter, snew, f(snew));
printf("Test time = %7.7f",snew);
printf("\n");
T_array[index]=snew;
}   /* end of the case of B - A*A/4 < 0 */
}
*****************************************************************
```

```
double f(s) /* Function for numerical analyis */
double s;
{
double  U, X, Y, Z;
double  fcn;

U = sqrt(-(B - A*A/4.0));
X = ALPHA*B*B/PR - D0*(B-D1) - D2*(B-D2);
Y = D0*D1 + D2*D2__;
Z = ((2*B*C1-A*D1)*D0 +(2*B*C2__-A*D2__)*D2)/(2*U);
fcn = 2.0 * X - (Y-Z)*exp((U-A/2.0)*s) - (Y+Z)*exp(-(U+A/2.0)*s);
return(fcn);
}  /* end of function f(s) */
**************************************************************************
Optimum(ip1)  /* Fins optimum P(1) for the best testing time */ int ip1;
{
extern double T_array[NUMTEST]; /* keeps testing times corresponding to P(1)'s*/
int op_index=50000;             /* Set a dummy index with a high enough value */
double best_time=1.0e20;        /*  Set a dummy maximum testing time to check program  */
int index=ip1;
for(index=0;index<NUMTEST;index++)
printf("%d.  testing time=%5.5f\n",index,T_array[index]);

for(index=0;index<NUMTEST;index++)
{
if (T_array[index]<best_time )
{
op_index=index;
best_time=T_array[index];
}

if (op_index==50000)
/* if no Positive value was found for the 11 different*/ /* P(1)   */
printf("You have a big trouble, big guy!");
}
printf("\n");
```

```
printf("The optimum prob of 1 for the best testing time is:\n");
printf("P(1)=%2.5f for best T(max) of %7.5f mSec",op_index*resolution,best_time);
printf("\n");
} /* end of function Optimum */
```