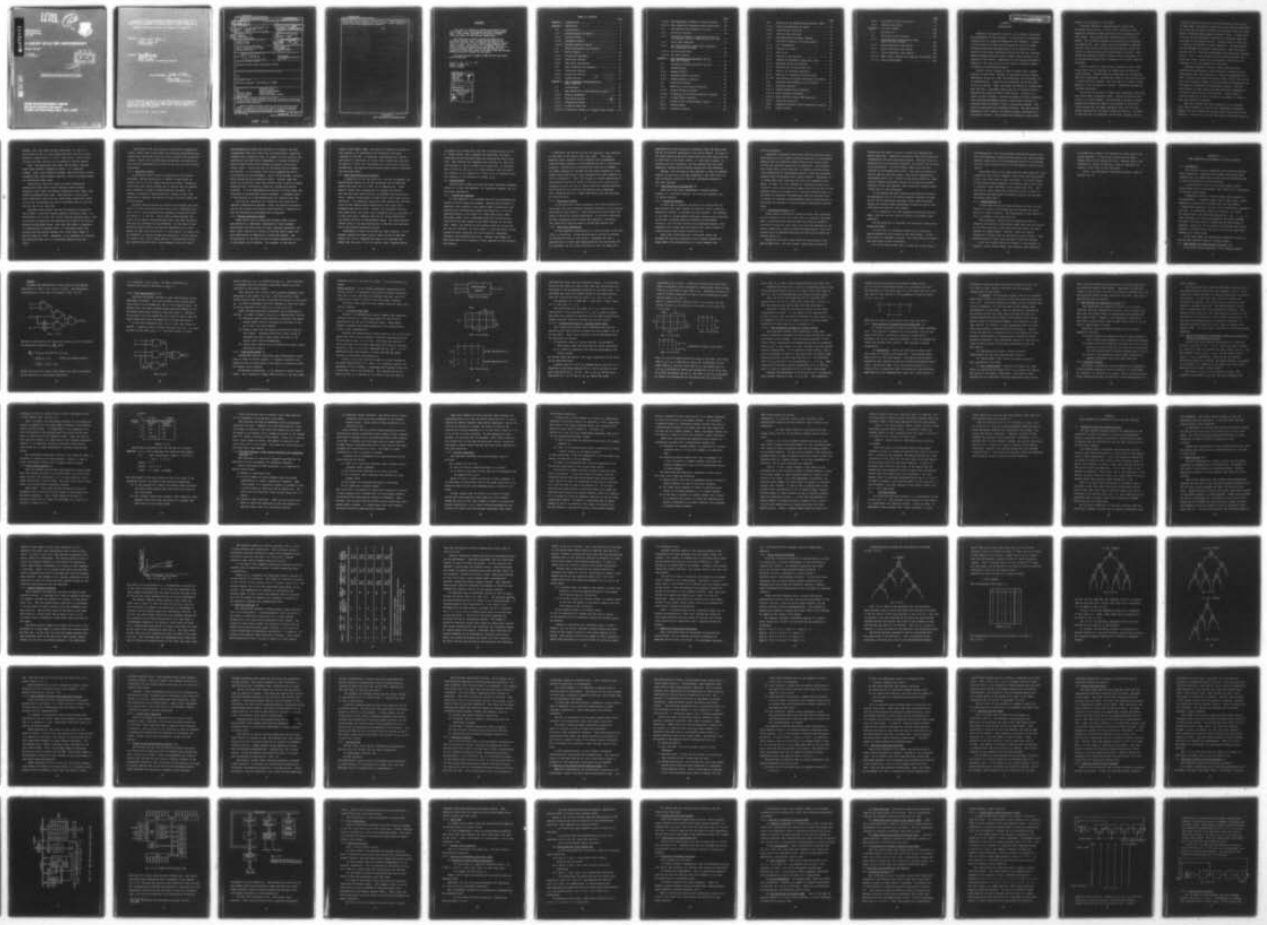
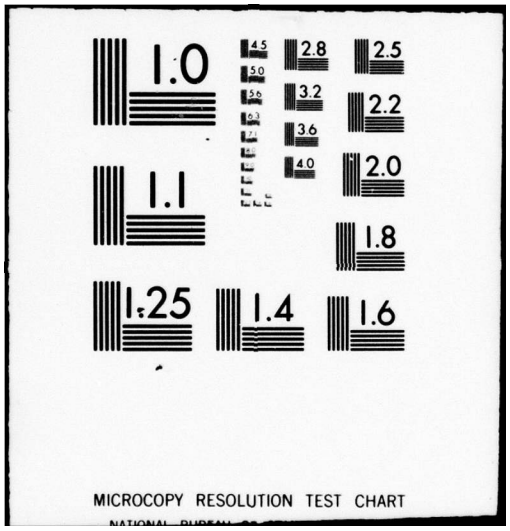
AD-A070 733 CLARKSON COLL OF TECHNOLOGY POTSDAM N Y F/G 14/2
A SURVEY OF LSI TEST METHODOLOGY.(U)
MAY 79 U V GUMASTE, R M MATTHEYSES F30602-75-C-0082
UNCLASSIFIED RADC -TR-79-85 NL

1 OF 2
AD
A070733




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART

LEVEL II

AD A070733

RADC-TR-79-85
Final Technical Report
May 1979

A SURVEY OF LSI TEST METHODOLOGY

Clarkson College

U.V Gumaste
R.M. Mattheyses

A SURVEY OF LSI TEST METHODOLOGY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-85 has been reviewed and is approved for publication.

APPROVED: Warren H. DeBany, Jr.

WARREN H. DEBANY, JR.
Project Engineer

APPROVED: Joseph J. Maresky

JOSEPH J. MARESKY
Chief, Reliability & Compatibility Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBEM), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(18) 1. REPORT NUMBER: RADC-TR-79-85

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): A SURVEY OF LSI TEST METHODOLOGY

(9) 5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, July 1977 - September 1978

6. PERFORMING ORG. REPORT NUMBER: N/A

(10) 7. AUTHOR(s): U.V. Gumaste, R.M. Mattheyses

(15) 8. CONTRACT OR GRANT NUMBER(s): F30602-75-C-0082

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Clardson College, Potsdam NY 13676 085 000

(16) 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 233801P0 (17) 01

11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (RBRM), Griffiss AFB NY 13441

(11) 12. REPORT DATE: May 1979

13. NUMBER OF PAGES: 112

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12) 122 p.

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Warren Debany, Jr. (RBRM)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

LSI	primitive d-cubes
VLSI	binary decision diagrams
functional testing	signature analysis
structural testing	pseudo random binary sequence
random testing	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discusses different test methods available for testing Logic Circuits and Large Scale Integrated Circuits (LSI) from Structural, Functional and Random testing viewpoints.

Many test schemes have been proposed in the literature and this report attempts to classify the different test schemes into two basic approaches, namely, test generation and test application. An attempt is also made to indicate the philosophy underlying different test equipment.

(over)

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

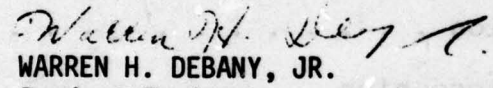
The report also incorporates a brief introduction to digital testing which should enable engineers unfamiliar with the concepts of digital testing to read this report with minimum external references.

EVALUATION

This report is intended to introduce the digital LSI/VLSI test designer to the various methods available for generating tests. These methods, described in the current literature under many names, are described and compared here.

Most terms used in today's testing environment are explained and examples given. Methods outlined here are not peculiar to one logic family or another, but pertain to digital logic as a whole. Thus, this is not a testing cookbook, but rather a guide to the derivation of test methods. As such, it will be of greatest benefit to those who must generate test programs for unique device types or who need to know the rationale behind existing test concepts for any reason.

This work was done in support of RADC TPO 5B, Solid State Device Reliability.



WARREN H. DEBANY, JR.
Project Engineer

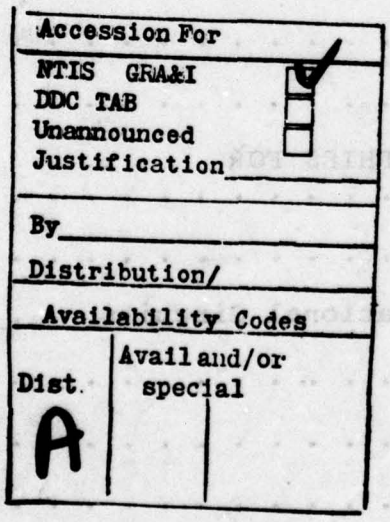


TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Testing is an important activity in the design, acceptance, and maintenance of large systems. In many areas of engineering testing is an art requiring experience and judgement. Digital circuit testing has been elevated beyond the artistic stage. Techniques have been developed which can be applied to any digital circuit given sufficient storage and time for analysis. These techniques generate sets of inputs (test vectors) to be applied to the circuit. Observation of the outputs produced in response to the test vectors can be used to determine whether the device is functioning properly.

Digital circuit testing is used to ensure that a system operates as specified. Testing in the design phase of system development serves as a tool for determining the correctness of an implementation. Errors may be introduced during the fabrication of a device. Thus, even after the design of a device has been verified, a purchaser should test the circuits he buys to establish that they are free of such errors. Finally, as a result of age or misuse, a device may cease to function properly. Thus, periodic maintenance should include device testing to establish that the device is still functioning properly. This report is concerned with acceptance testing. The test methods considered could, however, be used for design verification or maintenance testing. The criteria for selecting a test method

depends on the purpose of the tests.

Classical methods of testing digital systems were developed during the era of small scale integration (SSI). A system was composed of many packages, each containing a few logic gates. The packages were interconnected on circuit boards. The connection between the packages were accessible for observation. Thus, tests could be performed by applying inputs to the circuit and observing the response within the circuit at various test points using scopes or meters. Even though the structure within a package was not accessible, it was sufficiently simple that a complete set of tests could be developed.

With the advent of medium scale integration (MSI), functional units such as registers or adders were included in a single package. Methods which were developed for SSI devices were still applicable although the number of test points within a circuit had been reduced. More care had to be taken in the design of a set of tests for a device. Exhaustive testing was becoming infeasible.

We are now in the age of large scale integration (LSI) and very large scale integration (VLSI). Entire systems are contained in a single package. The problems of testing a single package have become the same as the problems of testing an entire system with the added constraint that the only test points available are the system inputs and outputs. In order to test one part of a package, the stimulus (inputs) and the

responses (results) may have to pass through many other parts. In some cases, the desired outputs may not be directly observable as in the case of testing whether an arithmetic logic unit (ALU) within a microprocessor properly handles a flag. In such cases tests can only be performed indirectly by observing other outputs such as the program counter. Another cause of difficulty in testing VLSI packages is their generality. Microprocessors are not designed for specific applications. They are general purpose devices. They are programmable and contain many internal states. Thus, a test consists of getting the processor into a particular state, giving it the appropriate instruction and supplying it with the appropriate input data.

Classical methods of digital circuit testing may be theoretically applicable to VLSI testing but storage and time requirements make them impractical. Since microprocessors are used in applications where malfunction would involve the loss of life and/or property damage, new and accurate techniques must be developed for acceptance and maintenance testing. The non-observability of some outputs and the structural complexity of the devices make this goal seem distant. In some cases a reduced confidence in device integrity is accepted in order to make test generation feasible.

Many test schemes have been proposed in the literature. This report attempts to identity the fundamental differences between the test schemes and consider their various merits and demerits. In doing so, we limit our study to two basic areas of

basic test philosophy; test generation, and test application. Under test generation we will consider three basic approaches to generating test inputs. We consider the difficulty of generating the inputs in terms of hardware and software requirements as well as the limitations on test confidence imposed by each method. Under test application we will consider the hardware and software requirements for conducting a test. The essential features of various test methods are abstracted in an attempt to develop a system for classifying test application methods.

Chapter two defines the basic terminology that will be used in the report and also discusses fault models and device models that are currently in use. Chapter three presents a discussion of test generation techniques for logic circuits, which is centered around combinational and sequential circuits. Chapter four presents test generation philosophy for LSI and VLSI circuits. Also, it presents a taxonomy for test application techniques. This discussion is based on structural, functional and random testing. Chapter five considers some of the currently available systems for testing VLSI devices.

CHAPTER II

DEFINITIONS

2.1 Introduction

In this chapter we will discuss different concepts relevant to faults and tests and describe some types of faults which occur in different technologies such as LSI and VLSI circuits.

A fault can be considered to be a defect in design or an aberration introduced in the manufacturing process which causes a device behavior to deviate from what is specified.

In a general sense, testing consists of applying a sequence of inputs to a circuit, observing the output sequence and comparing it with a precomputed expected output sequence. Any discrepancy is said to constitute an error and the cause of this error is said to be a physical fault.

2.2 Classification of Faults [1]

Faults can be broadly classified as logical, parametric or pattern sensitive.

2.2.1 Logical Faults

A logical fault is one which causes a device to appear to implement a different logic function than the one specified. A typical logical fault which is frequently considered is stuck at (SA) fault. In this class of faults circuit signals become fixed at some constant value, say logical one, in which case the signal is said to be stuck-at-one (s-a-1), or logical zero, in which case the signal is said to be stuck-at-zero (s-a-0). For

example, for a two input OR gate with inputs "a" and "b", a s-a-0 fault on the "a" input causes the logic function of the OR gate to change from a + b to b. Many faults such as short circuits and open circuits can be modeled as logical faults.

Here one should note that under the SA model, failures cause fixed signals to appear at leads - i.e. leads get clamped. Thus, tests based on SA model deal with static faults. Parameters that affect dynamic behavior, such as switching speed, are verified by other tests.

The stuck at (SA) fault model was first proposed for dealing with early logic circuit families [such as Diode Transistor Logic (DTL) and Resistor Transistor Logic (RTL)] when discrete components were used. Just how well it fits large scale integration (LSI) and very large scale integration (VLSI) is not clear and perhaps the use of a SA model only to test LSI and VLSI may not be justified.

Prominent among faults that are not adequately covered by the SA model is shorting between adjacent conducting lines. In technologies such as RTL, DTL and ECL (Emitter Coupled Logic), this failure can be modeled as the insertion of an AND or OR function between the shortened leads. Even when this model is adequate - in consideration of shorted lines - it introduces great complexity into the testing process, due to the large number of pair of lines on a chip. Therefore, it is necessary to eliminate all lead pairs that are at a sufficient distance from each other.

The SA model also falls down in dealing with intermittent faults. Tests under the SA model can be easily described by the conventional analytical tools for logic circuits such as Boolean algebra. Hence, the SA model offers analytical convenience as well as good representation of most of the (but not all) failure mechanisms.

2.2.2 Parametric Faults

A parametric fault alters the magnitude of the circuit parameter, thereby resulting in a change in some factor or factors such as circuit speed, current or voltage. Parametric faults may occur during storage due to factors such as temperature, humidity, leakage of sealed elements and aging. An example of parametric faults would be a change in the clock speed due to improper functioning of clock circuitry inside the microprocessor.

Periodic testing should be carried out throughout the lifetime of a device since faults may occur or get introduced into a logic circuit during manufacturing assembly, storage and while the device is in service. During each of these periods, the nature of the faults introduced and, hence, the type of testing that must be performed will be different. At the time of manufacturing typical faults that may be introduced are (1) open bonds, (2) open interconnections, (3) bulk shorts, (4) shorts due to scratches, (5) shorts through the dielectric, (6) pin shorts, (7) cracks, etc. Due to these factors, a manufactured circuit may contain multiple faults, some permanent [this is a fault

that permanently changes the character of a device] and some intermittent [this is a fault that is discontinuous, appearing randomly over a period of time, an example of this would be the shorting of two leads due to mechanical or voltage stressing]. Some of these faults can be modeled as logical faults while others cannot. Faults may also be introduced during assembly and testing. It is possible that faulty elements may not be discovered until after assembly. Also, during storage, circuits might develop certain parametric faults. Finally, when the device is in service, these same factors occur as well as others caused by heat, dissipation, vibration, voltage and current stresses. It is an established fact that as a circuit ages, the occurrence of intermittent faults increases. A probable cause of this problem is in the deterioration of contacts with time.

Failures are frequently not random but the result of an imperfect manufacturing process. Hence, the accurate determination of the location and cause of such failures is important so that the manufacturing processes can be improved.

2.2.3 Pattern Sensitive Faults

These faults occur in situations brought about by worst case switching of addresses, topologically "next neighbor" metalization runs which are too close, bad poly insulators, etc. Such a fault results in a failure of the LSI device under certain combinations of addressing, writing and reading. The failure occurs in the form of loss of stored information in one or more memory cell locations. For example, in the case of

random access memory (RAM), the result of reading or writing in some register R is affected by the contents of the other registers of the RAM. A frequently used approach for testing such faults is functional in nature and is based upon treating the RAM as a "black box" [this makes very little use of circuit information], and applying test patterns which tend to "exercise" the functional model.

2.3 Fault Distribution and Location

Irrespective of how the effect of a failure is modeled, one should decide whether or not the assumption can be made that faults only occur one at a time, or in combinations. Tests based on single fault assumptions are simpler and shorter than the ones that would consider all possible failure combinations. However, sufficiency of single fault tests needs careful evaluation. In production testing for high density LSI and VLSI [even for MSI (Medium Scale Integration)] the single fault assumption may not be valid. However, for equipment that has been operational, the single fault assumption may be justified. It is also true that a complete test for all single faults will also detect the bulk of simultaneous faults. But this requires careful study of the test results due to the fact that multiple faults can produce misleading results.

Redundancy in tests can be used for fault location. As an example, assume that there exists a fault in one of the four units A, B, C or D. Suppose that test 1 in a sequence can reveal that only unit A or B is faulty, test 2 detects faults

in either C or D units only, and test 3 detects faults in A or C. Hence, tests 1 and 2 together are sufficient for fault detection but all the three tests are required for fault location. The condition for complete fault location is that if a fault exists, it can be traced to a single unit, if for every pair of units U_i and U_j ($i \neq j$) there exists a test for which the response in the presence of a fault in U_i is different from the response in the presence of a fault in U_j.

2.4 Device Models

Various device models are used for fault detection purposes, namely (1) a Black Box Approach, (2) Functional Diagram, and (3) Gate Level Description.

2.4.1 A Black Box Approach

In this approach, it is assumed that the test equipment has access to device inputs and outputs. Testing would consist of applying a test sequence to the device under consideration and observing the corresponding outputs and comparing it with the expected outputs or the output of a known good device (KGD). This model seems to be a reasonable model for microprocessors since we do not have access to the nodes internal to the VLSI chip. Hence, the testing of the internal nodes can be carried out only by inference. For this reason, a black box model is well suited for fault detection and can be used for acceptance testing at the manufacturing level. However, this model for VLSI will serve little purpose if fault location is the criterion for testing.

Typically, the testing consists of applying a test sequence to the input of the device under test (DUT). The corresponding output sequence can be compared with a precomputed expected output sequence to check whether the DUT is functioning properly. Another way of verifying this would be to compare the output sequence of the DUT with the output sequence of a so-called "known good device" (also referred to as gold standard device in literature). Here, the concept of a known good device is questionable since for a device to be fault free it has to be completely tested and this is rather difficult because prevalent test methods do not guarantee 100 percent test confidence. Also, synchronization of a known good device with the DUT might be very difficult.

2.4.2 Functional Diagram

In the functional diagram approach, one tries to partition a given VLSI chip into different subblocks according to the function performed by each subblock. After functional division each subblock is tested for proper functioning. In doing so, there may be some overlapping tests between the subblocks. Currently, this approach of testing seems to be the most popular one.

2.4.3 Gate Level Description

In this approach, subdivision of a VLSI chip down to the gate level is considered. Proper functioning of the DUT is ensured by testing each gate in the circuit. Problems with this approach are: (1) Gate level description of the circuit may not be available; (2) It is very difficult to test all gates and

some gates can only be tested by inference since the gate output may not be directly observable at the device output; (3) Some of the gates cannot be directly stimulated since gate inputs may not be available at the device inputs. In this type of model, to achieve high test confidence, one may require very long test sequences, thereby making testing lengthy and time consuming.

However, the advantage of this approach is that if complete testing is carried out one may accomplish a very high level of test confidence relative to the test confidence achieved by any other approach.

2.5 Test Generation Philosophies [5]

There are three basic approaches to testing digital systems: (1) Functional Testing, (2) Structural Testing, and (3) Random Testing.

2.5.1 Functional Testing

In this type of testing, the goal is to verify that the device under test (DUT) behaves as required. This is done by determining whether it performs its task properly. For example, in the case of microprocessors functional testing would typically verify that a program counter increments and decrements correctly, that all ALU functions are performed properly, that registers can be read from and written into, that transfer of control occurs under proper conditions, etc.

Tests that verify functions (functional tests) are suitable for central processing units or memory systems and these tests find wide application in final assembly and

field maintenance.

Typically, functional tests would start by verifying that a small portion of the system [called the hardcore] is functioning properly and then progress outward through the circuits that perform the remaining repertoire of operations. The drawback with these tests is that they are, in general, written without close examination of the hardware details, since the programmer usually cannot be burdened with circuit details. Therefore, it is likely that these tests may turn out to be somewhat incomplete. In fact, it is not unusual for functional tests to be limited in their fault detecting capabilities to only half of the possible faults in a digital unit. But tests derived solely on the basis of functional considerations need not necessarily be so limited in coverage; microprogrammed machine organizations in particular are amenable to fairly complete functionally derived tests.

2.5.2 Structural Testing [5,6]

Tests based on structural criterion assure that individual circuit elements of the unit under test are operating correctly. Thus, a structural test assures that every gate on the DUT is functioning correctly, that every flip flop goes through all its states [namely set, reset or toggled as appropriate], that clock signals occur at designed frequency and that parity checks are correct and so on.

Designing functional tests calls for familiarity of the device under test. For this reason, these tests can best be

designed by the person or group of people who designed the device under test. Typically, functional tests do not take into consideration the fine structural details. When the tests are designed from the structural viewpoint, one starts with the gate level description. For this reason, these tests are more suitable at the manufacturing level. This type of testing can also be used for off-line repair. When strict dependability requirements exist [i.e. failure of the device results in catastrophic damage] structural testing can be used for periodic checking of real time systems, since structural testing yields a high level of test confidence.

If the assumptions made regarding the effects of failure are appropriate and if the correct network structure is known at the time of developing structural tests, then structural testing can furnish proper diagnostics through the use of the algorithms developed for this purpose.

If no assumptions are made regarding the device's failure modes, then the test will have to contain all possible input sequences.

The key features of structural and functional testing are summarized below.

(1) The level of confidence offered by the structural testing depends on the accuracy of the fault model. If the fault model is reasonably accurate, then these tests can yield a very high level of test confidence.

(2) If the system under test is large [as in the case of

microprocessors], then the structural testing may require very long test sequences, thereby making testing very time consuming. However, these tests can be developed by means of an algorithm and they can be optimized as far as the sequence length is concerned.

Functional tests on the other hand are very useful for large systems but they are not complete in the sense that they cannot yield a very high level of test confidence. The effectiveness of functional testing depends on who designs these tests since it requires a good knowledge of the system under test. [For this reason, these tests will yield best results if they are designed by the same person (or group of people) who designed the device under test.]

2.5.3 Random Testing [8]

In this type of testing, input patterns are fed to a prototype of a circuit to be tested and are analyzed for their ability to detect failures. The disadvantage of this method is that it produces a very large test set.

Very frequently, testing can be economically done by comparing the outputs of the circuit under test with the outputs of the known good unit, while both units are fed by the same sequence of random inputs. However, a good unit may be impractical to obtain and its reliability is not ensured. Also, synchronization of both units may pose a difficult problem.

A new test method consists of feeding a long sequence of random or pseudo random inputs, and computing some statistics

of outputs [for example, the frequency of logic ones in the output sequence]. If the output statistics are satisfactory then the unit passes the test. However, due to the probabilistic nature of testing, it may be impossible to achieve a high level of test confidence.

[Note: For discussion of Signature Analysis, refer to Section 4.8.1.]

CHAPTER III

TEST GENERATION PHILOSOPHIES FOR LOGIC CIRCUITS

3.1 Introduction

This chapter discusses different test generation philosophies [structural, functional and random] for digital [both combinational and sequential] circuits and test methods based on these philosophies are also discussed.

A combinational circuit is one whose present output value depends only on the present input value and is independent of the past input values.

A sequential circuit is the one whose output depends on two parameters: 1) present inputs, and 2) past inputs. Again, sequential circuits can be divided into two sub-classes, a) synchronous sequential circuits and b) asynchronous sequential circuits. In synchronous sequential circuits, the inputs are synchronized with some timing sequence t_1, t_2, ..., t_n. At every occurrence of the timing sequence, the circuit samples the input, the next state is entered and the next output is produced. Typically, the circuitry that generates this timing sequence is referred to as the clock circuitry and the timing sequence as a clock signal.

On the other hand, the asynchronous sequential circuit is the one that operates without a clocking signal.

3.2 Test Methods for Combinational Circuits [1,5]

A structural testing approach is very commonly used to

test this type of circuit. It means that typically, testing would consist of verifying whether individual hardware components of the circuit under test are functioning properly or not. Since the structural approach is used, most of the test methods can be developed with the aid of algorithms.

Prominent test methods for testing combinational circuits are a) Boolean Difference, b) Path Sensitization and c) the D-Algorithm.

3.2.1 Boolean Difference

Let $+$, $\cdot$, $\oplus$ denote logical OR, logical AND and Exclusive OR functions respectively. If x is a boolean variable, then $\bar{x}$ represents its complement. Consider a combinational circuit whose output z is the function of a set of inputs, say, x_1, x_2, ..., x_{i-1}, x_i, ..., x_n. Mathematically, this can be represented as

$$z = f(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) = f(\underline{x})$$

where $\underline{x}$ represents a vector $(x_1, x_2, \ldots, x_{i-1}, x_i, \ldots, x_n)$. If one of the inputs x_i is, say, stuck-at-1 (s-a-1) then this can be represented as

$$z_i^1 = f_i^1(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) = f_i^1(\underline{x}) \qquad \text{(I)}$$

Here, the notation f_i^1 means that the ith input is s-a-1. Similarly, if one of the inputs x_i is stuck-at-0 (s-a-0), it can be represented by

$$z_i^0 = f_i^0(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) = f_i^0(\underline{x}) \qquad \text{(II)}$$

In general notation conditions I and II can be written as $z_i^J = f_i^J(x_1, x_2, \ldots, x_{i-1}, J, x_{i+1}, \ldots, x_n) = f_i^J(\underline{x})$ where $J = 1$ for s-a-1 type of fault and $J = 0$ for s-a-0 type of fault.

The set of tests that will detect the fault J [i.e. one of the inputs i either stuck-at-1 (s-a-1) or stuck-at-0 (s-a-0)] corresponds to the function

$$Y(\underline{x}_i)^* = f \cdot \bar{f}_i^J + \bar{f} \cdot f_i^J$$

$$= f \oplus f_i^J$$

consider only the stuck-at-0 type of fault (i.e. J=0). Applying Shannon's expansion formula: $f(x_1, x_2, \ldots, x_n) = x_1 f(1, x_2, \ldots, x_n) + \bar{x}_1 f(0, x_2, \ldots, x_n)$ to Y yields: [Refer to Section 3.2.2]

For s-a-0 test, $x_1 = 1$

$$Y(\underline{x}_i) = (x_i \cdot f_i^1 + \bar{x}_i \cdot f_i^0) \oplus f_i^0$$

$$= x_i \cdot f_i^1 \cdot \bar{f}_i^0 + x_i \cdot \bar{f}_i^1 \cdot f_i^0 \qquad \text{(after due expansion and simplification)}$$

$$= x_i \cdot (f_i^1 \oplus f_i^0)$$

The term $f_i^1 \oplus f_i^0$ is referred to as the "Boolean difference" of the combinational function f with respect to the input x_i. If we denote $f_i^1 \oplus f_i^0$ by $\frac{dz}{dx_i}$, it represents all the conditions for which the value of f is sensitive to the input x_i alone. $x_i \frac{dz}{dx_i} = 1$ would represent the set of tests for the input x_i to be s-a-0 while a similar $\overline{x_i} \frac{dz}{dx_i} = 1$ would represent the set of tests for the input x_i to be stuck-at-1.

[Note: Here $\underline{x}_i$ represents a specific binary value of the vector x.]

Example

Consider the combinational circuit given by the Boolean expression $z = f(\underline{x}) = (x_1 + x_2) \cdot x_3 + \bar{x}_3 \cdot \bar{x}_4$. The diagramatic representation of the same is as shown in Fig. 3.2.1-1.

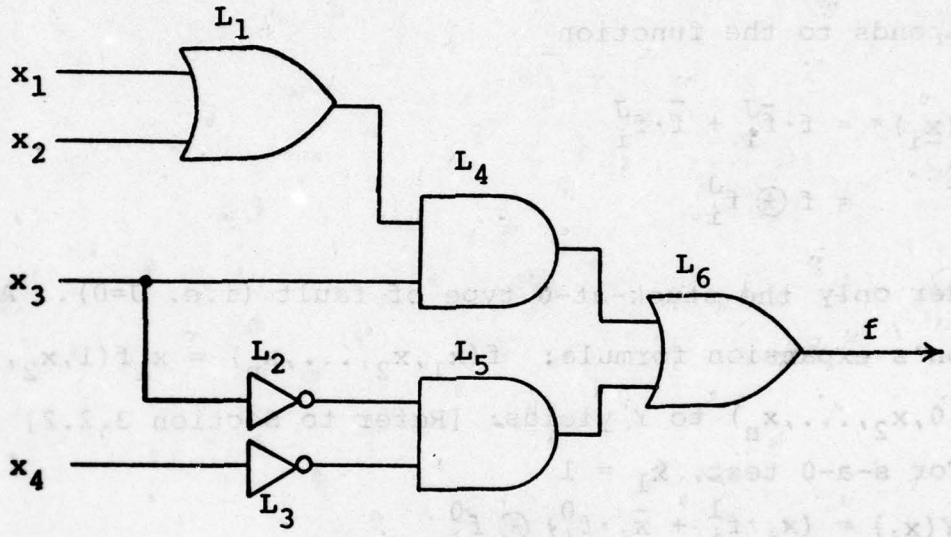


Fig. 3.2.1-1

The set of tests which will detect the fault x_3 s-a-0 is defined by the Boolean expression $x_3 \frac{dz}{dx_3}$ where

$$\frac{dz}{dx_3} = f(x_1, x_2, 0, x_4) \oplus f(x_1, x_2, 1, x_4)$$

$$= \bar{x}_4 \oplus (x_1 + x_2) \qquad \text{[After due simplification]}$$

$$= \bar{x}_1 \bar{x}_2 \bar{x}_4 + x_1 x_4 + x_2 x_4$$

Hence, the set of all tests which detect this fault is defined by the solutions to the Boolean expression

$Y = x_3(\bar{x}_1\bar{x}_2\bar{x}_4 + x_1x_4 + x_2x_4)$. An input combination $\underline{x}_i$ detects this fault if and only if $Y(\underline{x}_i) = 1$.

3.2.2 Path Sensitization [1,5]

The basic principle involved in path sensitization can be described as follows. Let k be the node in the circuit under test which is to be tested for, say, a s-a-0 type of fault. To achieve this, the input signals $\underline{x}$ must cause the signal at k in the normal fault-free circuit to take the value 1. Similarly, if the node k is to be tested for s-a-1 type of fault, then the input signal must cause the signal at k to take the value 0.

In the logic diagram of Fig. 3.2.2-1, each gate output is labeled. Suppose we want to detect the fault L_2 s-a-0. An input x_i must be such that $L_2(x_i) = 1$. For this circuit it would

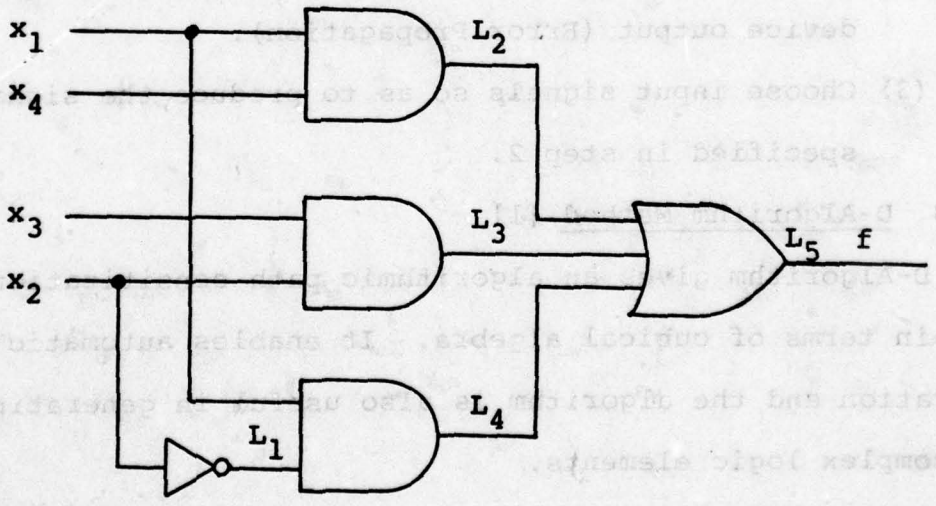


Fig. 3.2.2-1

require that $x_1 = x_4 = 1$, denoted by $x_1x_4 = 1$. This condition is necessary but not sufficient since gate output L_2 is not available at the device output.

This requires that a signal from L_2 should propagate along some path to the output (i.e. L_5). For this circuit, there exists only one path and that is through the OR gate to the output L_5. In order to propagate the fault through L_5, both the inputs x_3 and x_2 should be 0 so that L_5 is 1 only if L_2 is 1.

Thus, the path sensitizing procedure can be described as follows. In order to detect a fault in a combinational circuit:

(1) Determine input values required to generate the appropriate signal value at the point of the fault (0 for s-a-1 and 1 for s-a-0 faults).

(2) Choose a path from the point of the fault to the circuit output. Determine additional signal values to propagate the fault signal along this path to the device output (Error Propagation).

(3) Choose input signals so as to produce the signal values specified in step 2.

3.2.3 D-Algorithm Method [1]

D-Algorithm gives an algorithmic path sensitization procedure in terms of cubical algebra. It enables automatic test generation and the algorithm is also useful in generating tests for complex logic elements.

To describe D-Algorithm, it is required to define certain terms. Let U represent a signal whose value is 1 for the normal

operation and 0 in the case of a fault. $\bar{U}$ can be defined vice versa.

Prime Implicant: If the Boolean expression representing the digital circuit is expressed in the sum of products (minterms) form then a prime implicant is a product term that cannot be combined with others to yield a term with fewer literals.

We define three types of cubes [these terms are taken from cubical algebra].

3.2.3.1 Primitive Cube (PC)

Consider the combinational circuit element that generates a Boolean function f. Prime implicants of f and $\bar{f}$ can be represented with the aid of primitive cubes. These cubes properly represent the logical behavior of the combinational circuit under consideration.

Consider the combinational circuit shown in Fig. 3.2.3.1-1(a). Assume that this circuit realizes a function represented by the Karnaugh map in Fig. 3.2.3.1-1(b). The prime implicants of f are $\bar{x}_1$ and $\bar{x}_3\bar{x}_2$ and the prime implicants of $\bar{f}$ are x_3x_1 and x_2x_1. Labeling in Fig. 3.2.3.1-1(a) indicates that x_i (i=1,...,3) is associated with each position i of the cube and the output function is associated with position 4.

In Fig. 3.2.3.1-1(c) the first cube of α_1 0XX1 represents the prime implicant $\bar{x}_1$ of f. Here, 0 in a position implies the complement of the variable, 1 represents the variable itself and X indicates a don't care condition. The intersection of two cubes (α_i and μ_j) is defined as the value of the two cubes in

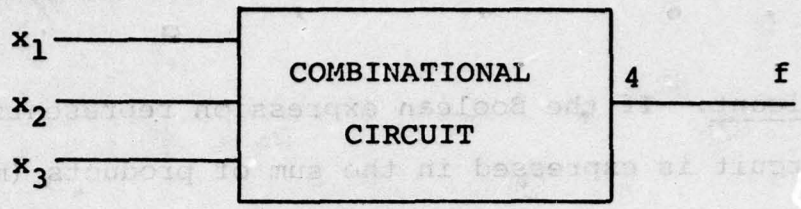


Fig. 3.2.3.1-1(a)

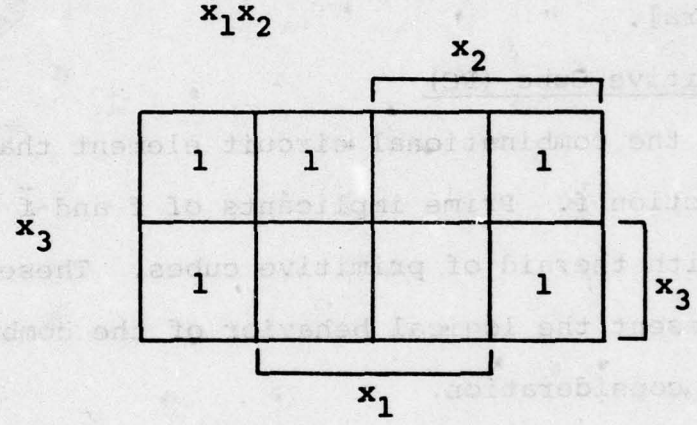


Fig. 3.2.3.1-1(b)

	1	2	3	4	
α_1	0	x	x	1	Prime Implicants of f
	x	1	0	1	
α_0	1	x	1	0	Prime Implicants of $\bar{f}$
	1	1	x	0	

Fig. 3.2.3.1-1(c)

each position where they have identical values. If a position in one of the cubes is a don't care, then the intersection has the value of the corresponding position in the other cube. If two cubes have specified unequal values in the same position then the intersection does not exist, i.e. ($\alpha_i \cap \mu_j = \phi$).

For example, if α = 0XX1 and δ = X101 and γ = 1X10, then $\alpha \cap \delta$ = 0101 and $\alpha \cap \gamma = \phi$.

The intersection of two cubes ($\alpha_i \cap \mu_j$) is said to be inconsistent if the intersection of α_i and β_j does not exist; this means two conditions (belonging to two cubes respectively) assign different values to the same line on the circuit.

3.2.3.2 A Primitive D-Cube of a Logical Fault (pdcf)

Let μ denote the minimal input conditions that must be applied to the logic device to produce an error signal (U or $\bar{U}$) at the output. These input conditions can be determined from the following two factors:

(1) primitive cubes of logical function f performed by the circuit under the fault-free or normal conditions;

(2) f_j, where f_j represents the function performed by the faulty circuit.

By knowing these two factors, the input conditions can be found out as described below.

A faulty output U (or $\bar{U}$) is produced by an input if it is contained in the prime implicants of f (or $\bar{f}$) and also in the prime implicants of f_j (or $\bar{f}_j$). Let α_1 (or α_0) denote prime implicants of f (or $\bar{f}$) and μ_1 (or μ_0) denote the prime

implicants of f_J (or $\bar{f}_j$). Primitive D cubes of the fault that result in output $\bar{U}$ can be obtained by intersecting inputs of each cube in μ_1 with that of each cube in α_0. Similarly, primitive D-cubes of a fault resulting in output U can be obtained by intersecting cubes in μ_0 and α_1.

EX for a combinational circuit of Fig. 3.2.3.1-1(a), let the faulty function f_j be defined as in the Fig. 3.2.3.2-1(a) and 3.2.3.2-1(b).

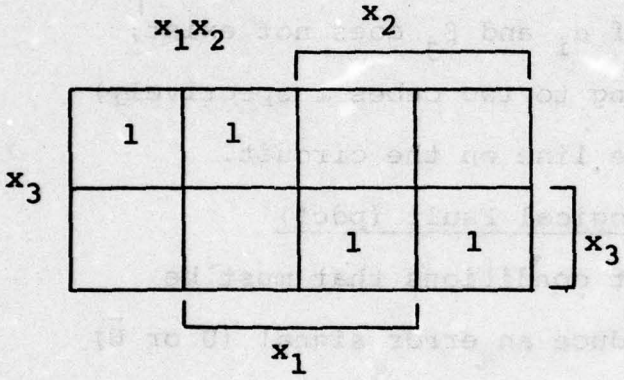


Fig. 3.2.3.2-1(a)

1	2	3	4	
X	1	1	1	μ_1
X	0	0	1	
X	0	1	0	μ_0
X	1	0	0	

Fig. 3.2.3.2-1(b)

1	2	3	4	
0	0	1	U	$\alpha_1 \cap \mu_0$
0	1	0	U	
1	1	1	$\bar{U}$	$\alpha_0 \cap \mu_1$

[Primitive D cubes of the fault]

Fig. 3.2.3.2-1(c)

Fig. 3.2.3.2-1(c) shows primitive D-cubes of the fault. The first cube in $\alpha_1 \cap \mu_0$ is obtained by taking the intersection of the first cube of α_1 and the first cube of μ_0; the second cube in $\alpha_1 \cap \mu_0$ is obtained by taking the intersection of the first cube of α_0 with the second cube of μ_0. The cube in $\alpha_0 \cap \mu_1$ is obtained by taking the intersection of the second cube in α_0 with the

first cube in μ_1 [here, other intersections do not exist (ϕ)].

Consider the first row in $\alpha_1 \cap \mu_0$; it specifies that if the first and second inputs are set to 0 and if the third input is set to 1, then the combinational gate output has the value U.

Here, the first cube specifies that if the first and second inputs are set to 1 and if the third input is set to 0, then the combinational gate output will have the value U.

Primitive D cubes for the stuck-at faults can be easily constructed as follows. If the output of the combinational circuit is s-a-1 then the output coordinate of every cube in α_0 is changed to $\bar{U}$ and, if the output is s-a-0, then the output coordinate of every cube in α_1 is changed to U.

3.2.3.3 The Propagation D-Cubes of a Logic Element

This specifies minimal input conditions to the logic element required to propagate an error signal between the input and output of that element. Let the logical behavior of the circuit under consideration be denoted by two sets of primitive cubes α_0 and α_1 that result in 0 and 1 outputs (from the logical circuit) respectively. If "e" denotes an error signal, then in order to propagate an error on the input line, the other inputs should be defined in such a way that for e = 1 the resulting cube is in β_1 and for e = 0 the resulting cube is in β_0. These cubes can be derived by taking intersection of e = 0 with β_0 and e = 1 with β_1 resulting in $\bar{u}$ or u or both e and output.

Consider the propagation of an error on line 1 through the logic element defined in Fig. 3.2.3.1-1(a). The propagation

D-cubes can be derived from the primitive cubes of Fig. 3.2.3.1-1(c) by intersecting cubes in α_0 for which the line 1 has the value 1; with those cubes in α_1 for which the line 1 has the value 0, this results in the propagation D cubes as shown in Fig. 3.2.3.3-1.

1	2	3	4
U	1	X	$\bar{U}$
$\bar{U}$	1	X	U

Fig. 3.2.3.3-1

There are no cubes in α_0 for which line 1 has the value 0.

3.2.3 Test Generation Procedure Using D-Algorithm [1]

(1) Choose a primitive D-cube of the fault under consideration. This results in producing an error signal u or $\bar{u}$ at the site of the fault. Initially a choice of D-cube will be arbitrary, but as the algorithm proceeds, it may be necessary to return and consider other choices. This process is called backtracking.

(2) Implication - During the execution of Step 1 some gate inputs or outputs may be specified. This requires specifying values of the other signals. The implication procedure determines these signals in both the forward and backward directions. During this step, if an inconsistency occurs [i.e. a value of 0(1) is implied on some line which has been previously specified to be the complementary value 1(0)], backtracking is

effected to the point where the choice existed, all the lines are reset to their value at this point and the procedure resumes with the new choice.

(3) D-Drive - Let D-frontier denote a set of all elements whose output values are unspecified but whose input has some signal u or $\bar{u}$. The D-drive selects an element in the D-frontier and tries to propagate u or $\bar{u}$ at the input of the element under consideration to the output. This is achieved by intersecting the current circuit test cube [this cube specifies all previously determined signal values of the circuit] with a propagation D-cube of the selected element. This results in a new test cube for the element. If this intersection does not exist then a new element is selected from the D-frontier. However, if the intersection is undefined for all the elements of the D-frontier, backtracking to the last point at which the choice existed is required. This backtracing results in resetting all lines to their values at that point and beginning with the next choice.

(4) Implication of D-drive - Carry out Step 2 (Implication) for the test cube derived in Step 3.

(5) Repeat Steps 3 and 4 until the faulty signal has been propagated to an output.

(6) Line Justification - Execution of Steps 1 to 5 may result in specifying the output value of an element but leaving the inputs to the element unspecified. The inputs to such elements are now specified to produce specified output values.

This is done by taking the intersection of the test cube with the primitive cubes of the element. Implication is then performed on the new test cube and the process is repeated until all the specified element outputs have been justified. Here again, backtracking may be required.

3.3 Test Methods for Sequential Circuits [1]

Test generation for sequential circuits is a lot more difficult than for combinational circuits. This is due to the fact that the present output is determined by the present and past input values. Usually, a test sequence is required to test these circuits rather than a single input vector as in the case of combinational circuits.

Primarily, there are three different ways of testing sequential circuits:

(1) verify whether the sequential circuit under consideration is functioning properly or not [Functional Testing];

(2) convert the given sequential circuits into a set of combinational circuits and test the resulting circuits with the test methods used for testing combinational circuits;

(3) verify if the sequential circuit under consideration behaves according to the truth table specified.

3.3.1 Functional Testing

In general, a sequential circuit consists of a combination of the following sequential circuit elements: (1) flip flops, (2) counters, (3) registers, (4) timing (clock) circuitry. Functional testing would mean verification of the functionality of

these elements.

For flip flops such a testing would mean verifying that it can change states as specified; for registers, that it can be loaded and cleared and that all shifting operations are carried out properly. It would test that counters increment and decrement as required, that clock circuitry generates clock signals at desired rates, etc. The effectiveness of such a test is judged with the aid of simulation techniques. This method can be effective if the test engineer is familiar with the details of the sequential circuit under consideration. At the stage of designing tests for sequential circuits, it requires repeated simulation runs in order to improve test effectiveness. For this reason, the method is time and resource consuming.

3.3.2 Testing by Treating a Sequential Circuit as an Iterative Array of Combinational Circuits

Consider the block diagram representation of sequential circuits, as shown in Fig. 3.3.2-1. Here, the vector notation $\underline{X}$, $\underline{Y}$ and $\underline{Z}$ are used to represent the input vector $(x_0,x_1,\ldots,x_n)$, the feedback vector $(y_0,y_1,\ldots,y_n)$, and the output vector $(z_0,z_1,\ldots,z_n)$, respectively. Since we are considering synchronous sequential circuits, the memory function is assumed to be clocked, i.e. updating of the memory only occurs when specified by the update (clock) signal.

The circuit in Fig. 3.3.2-1 works as follows. The contents of the memory along with the input determine the

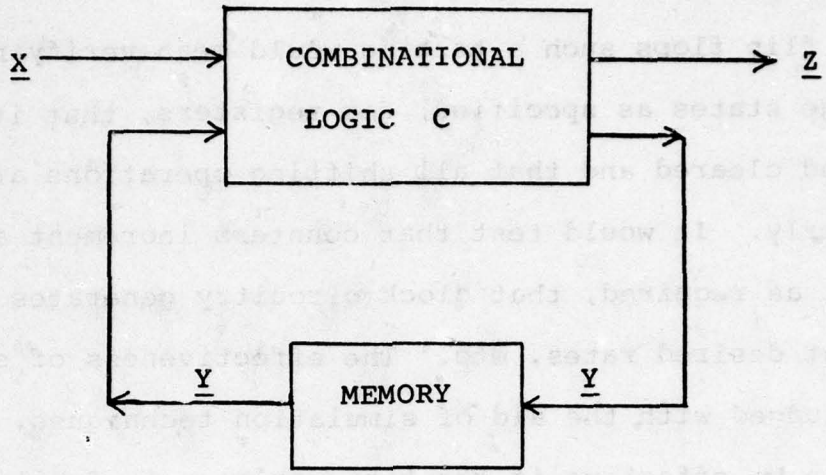


Fig. 3.3.2-1

current output. When the update signal [initiated by the clock circuitry] occurs, the contents of the memory are revised on the basis of the inputs at that time and the contents of the memory just prior to the update signal from the clock memory. The state of the sequential circuit is left unchanged until the next update signal from the clock.

Fig. 3.3.2-2 shows an equivalent combinational iterative array for the synchronous sequential circuit of Fig. 3.3.2-1 This iterative array would generate the output z_i from cell i in response to the input x_i; $1 \leq i \leq n$. Here, y_0 (i.e. the feedback at t = 0) is assumed to be known and its effect is treated as an input to the iterative array. In this transformation the clocked memory is modeled as a set of combinational elements referred to as pseudo memory functions.

Since the memory is modeled as a combinational element, the circuit in Fig. 3.3.2-2 can be tested by most of the

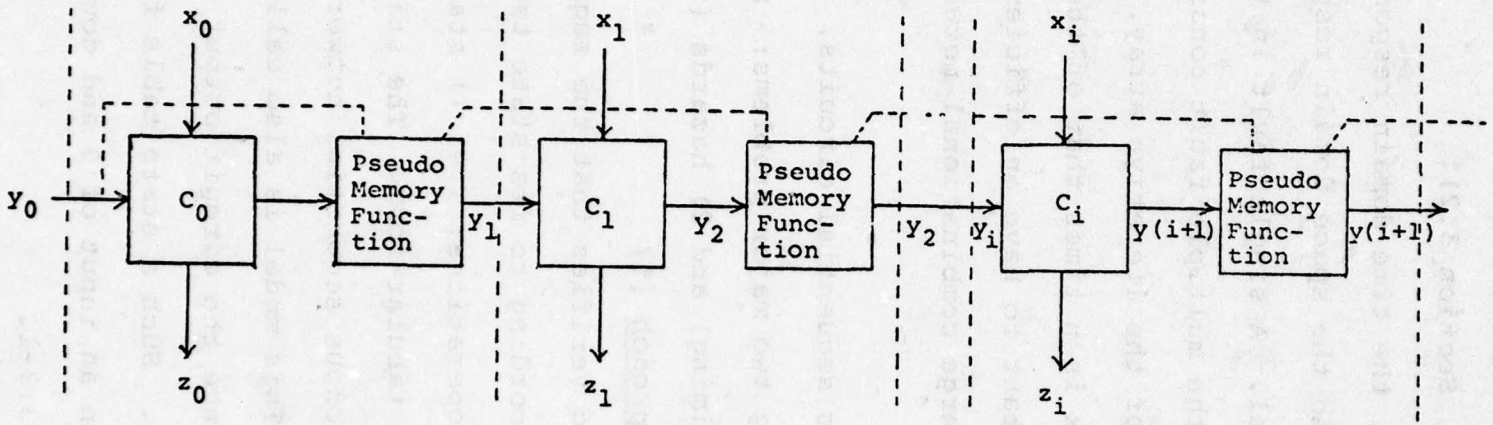


Fig. 3.3.2-2 - Equivalent combinational iterative array for the synchronous sequential circuit of Fig. 3.3.2-1. [Ref. 1]

techniques related to combinational circuits discussed earlier in this chapter (Ref. Section 3.2).

Here, in effect, the time domain response of the sequential circuit is mapped into the space domain response of the combinational iterative model. A single fault in the sequential circuit corresponds to the multiple fault consisting of the same fault in every cell of the iterative array. Since the size of the iterative network is n times that of the original sequential network, it is important to have an efficient algorithm for fault detection in large combinational networks with multiple faults.

For asynchronous sequential circuits, the iterative model is incapable of handling two major problems: namely, 1) that of races (coincidence timing) and 2) hazards (false outputs).

3.3.3 State Table Approach [5]

This test method verifies that the sequential network under test does operate according to its state table. The state table helps represent two operations: i.e. 1) state updating, and 2) output generation in tabular form. The state table uses a mealy model for the synchronous sequential network.

Mealy model. - This model is also called the transition triggered circuit since the circuit output is associated with the state transition. Such a state table for the mod (4) counter [counts up on an input of 0 and down on an input of 1] is given in table 3.3.3-1.

Present State \ Input x	0 (Up)	1 (Down)
	NEXT STATE/ OUTPUT	NEXT STATE/ OUTPUT
A	B/0	D/Borrow
B	C/0	A/0
C	D/0	B/0
D	A/Carry	C/0

Table 3.3.3-1

Mealy Machine Representation for the Mod of 4 Counter

NOTATION: A, B, C and D represent the states of the counter.

I/J : Next state/output where I = A or B or C or D J = 0 or 1

Inputs : 0, 1

States : A, B, C, D

Output : 0, CARRY, or BORROW.

The tests based on the verification of the truth table of the sequential network are based on the following assumptions:

(1) In the presence of a fault, the network under consideration has no more states than those which are listed in its truth table.

(2) For a normal (fault-free) network, there exists an input sequence which can cause the machine to transfer from each state to every other state.

Since this method uses an abstract state table approach, it is independent of the hardware fault model.

This approach can also be extended to test asynchronous sequential circuits. However, it is not very practical since the number of entries in the state table depend on the number of states in the network [if N = number of states in the network then 2^N = number of entries in the table]. Hence, for a network with a large number of states, the number of entries in the state table will be very large. This makes the construction of the state table very tedious.

3.3.4 The ATVG Program: A Test Vector Generator For Sequential Networks [7]

This technique is a practical computer algorithm for the generation of test procedures for sequential networks.

Before describing the ATVG algorithm, it is required to define certain terms.

A network is constructed from

(1) A finite number of logic elements which are either combinatorial or memory (sequential) elements. Each logic element in a sequential network is assumed to have a finite number of input pins and an output pin. The input pins are electrical loads and the output pin is a source.

(2) Connector input terminals: These are the circuit elements to which the components of the test vector are applied; hence, each is an electrical source.

(3) Connector output terminals: The binary value at these terminals constitutes the components of the network output vector. Hence, each of these terminals is an electrical load.

Logic elements and the entities mentioned above are connected by electrical connections called leads to form a network. Each lead is assumed to be a directed connection from a source to a load. A network will have many paths of leads starting at some source, traversing from lead to lead through the logic elements and terminating on a load. The number of leads traversed in a path is called its length.

A network can be considered as an interconnection of the above entities as follows:

(1) There is at least one connector input terminal and one connector output terminal.

(2) All sources are connected to at least one lead.

(3) All connector input terminals are connected to logic element leads.

(4) All loads are connected from exactly one source.

(5) All paths are of finite length.

This algorithm is developed for synchronous sequential circuits; it also requires that a network does not have pseudo elements constructed from combinatorial elements as feedback. This, however, does not mean that the actual circuit cannot have such pseudo memory elements. It simply means that such elements should be treated as combinational elements.

Each logic element and each connector input terminal are associated with one of the two binary values [0 or 1], or else [except for memory elements] they may be unassigned. During the test vector generation process, the values of each of the entities will become assigned [i.e. it will have a value of 0 or 1] if it had not been initially. At the completion of this process, all entities will be assigned. This (final) state of the network will be the same as the actual circuit, given the same memory element values and the same test vector applied to the input terminals.

3.3.4.1 Failure Conditions

The algorithm considers only the following types of failures:

(1) a lead s-a-0 or s-a-1

(2) only one failure condition exists in a network

(3) if there is any open circuit then the corresponding load is assumed to be s-a-0.

Hence, to test a network consisting of logic elements, it is sufficient to verify that none of the leads connected to its loads are open and each lead connected from its source is not fixed.

A logic element test (or tests) is a vector of binary values which when applied to the loads of a logic element causes the value of the logic element to react to a particular failure condition(s), i.e. a logic function performed by the circuit will differ from the function performed under normal

(fault-free) conditions.

A test T for a logic element L is said to be immediately completed if the test vector T is applied to L and a sensitive path from L to an output terminal exists [for definition of sensitive path, refer to page 21]. In case T is applied to L, but it is not immediately completed, then one of the two possibilities exists, either:

(1) there is at least one sensitive path from L to a memory element; or,

(2) there is no sensitive path from L terminating on either a memory element or an output terminal.

In the second case, T must be reapplied to L by some other test vector in order for it to be complete.

In case 1 it is possible that T can be complete but not during the application of the current test vector.

Let M denote a memory element on which a sensitive path from L terminates. If the next state of M under a failure condition differs from that which exists under normal conditions, then M is said to be sensitive to the failure condition of L, and the test is said to be stored in M. If in such a case the next state vector is such that there is at least one sensitive path from M to an output terminal, then T is complete. If T is stored in M but not complete, then one of the two cases listed above exists. If there is a sensitive path to memory element M_i and it is sensitive to M, then T is stored in M_i. By this process a test may be stored in many memory elements

during a sequence of test vectors until it is either completed or fails to be stored in any memory element. In conclusion, a sensitive path in a synchronous network differs from that in a combinational network in the sense that the last element of the path may be a memory element. Hence, the memory element output pins are a special form of the network's output terminals.

Additional requirements for generating test vectors for synchronous networks over a combinational network are:

(1) Determine if a memory element is sensitive to a failure condition when it is the last element in a sensitive path.

(2) Creating sensitive paths from a set of elements (i.e. a test is stored in a memory element) such that the terminal element will react to the simultaneous choice of these elements.

The ATVG algorithm generates a sequence of test vectors. Each test vector does the following:

(1) applies tests [that are not completed and not stored in memory elements] to some of the logic elements

(2) for these logic elements at least one sensitive path exists to an output terminal or to a memory element

(3) if the test is stored in a memory element [as a result of a previous test vector] at least one sensitive path exists, preferably to an output terminal, but otherwise to another memory element.

Test vector generation process.

Assumptions: (1) Initially binary values for each of the combinational elements and the connector input terminals are unassigned.

(2) Each memory element is assigned one of the binary values and some of the memory elements may be storing tests.

(Step 1) For a set of memory elements storing the same test vector, assign the minimum number of input terminals which will create at least a sensitive path from the memory element to an output terminal. If this is not possible, then assign the minimum number of input terminals which will create at least one sensitive path to a memory element. If no such assignment is possible, then no input terminals are assigned.

(Step 2) If for a logic element a test remains to be completed and is not currently stored in memory elements, assign the minimum number of input terminals which will apply a test vector to its loads. If no such assignment is possible, use the same logic element for other tests if possible or consider another logic element. If such a test is applied and if the element under consideration is not a memory element, assign a minimum number of input terminals so that a sensitive path(s) would be created as in Step 1. If the test was applied to a memory element, no sensitive path is created [this is due to the fact that memory does not change state until the next update signal]. Hence, a memory element does not react to a

failure condition until the next test vector is applied. For the same reason, sensitive paths for memory elements which are created in Step 1 have tests applied by the last test vector.

(Step 3) For every memory element, decide the next (after the update signal) source value. If possible, assign the minimum number of input terminals which will apply a vector to its loads and will produce the above value.

(Step 4) Assign values to unassigned input terminals at random.

The third step in the procedure above drives memory elements in order to produce the desired next state of the network. In most of the networks, the memory elements are interconnected in such a way that the next state of one may depend on the present state of many other elements. If Step 3 is executed after the first two steps, it may happen that the network has been assigned to such an extent that the desired control over the network state is not possible. For this reason, Step 3 is sometimes performed before Step 2 or Step 1. In order to produce a near minimum number of test vectors in a test procedure, the state changes of each memory element must be carefully controlled.

3.3.4.2 The ATVG Program

The input data for this program is 1) a description of the circuit set, i.e., the allowable logic elements of the network, 2) a description of a specific network. Using this data, it generates a test procedure and a fault dictionary. A test

vector generation algorithm uses the procedure described above [for details refer to reference 7].

The most difficult part of the procedure is Step 3 since it is not at all clear as to what strategies should be used to drive the memory elements into "desirable" next states. One cannot even define "desirable." However, in spite of this problem, the program is very successful in practice. It is observed that for the "average" network, the initial test procedure can test a network for about 85 percent of the failure conditions. However, this program allows the logic designer to modify Step 3. This provision typically allows the completed test procedure to be at least 95 percent effective.

CHAPTER 4

TEST GENERATION PHILOSOPHIES FOR LSI AND VLSI CIRCUITS

4.1 Problems of Testing LSI/VLSI Circuits

The test methods discussed earlier for combinational and sequential circuits are based on structural considerations. However, techniques based on structural considerations cannot be applied with the same ease for LSI and VLSI circuits due to the following reasons.

Any method based on structural considerations requires either a gate level description or a state table of the digital circuit under consideration. Due to the secrecy in design and the number of gates on VLSI circuits being extremely large, it may be very difficult to obtain a gate level description of the VLSI circuit under consideration. On the other hand, if one decides to construct a state table, due to the complexity of the device, the state table will have a very large number of entries thereby making this approach almost impossible to use.

Even if the gate level description is available, some of the gates on the chip cannot be directly stimulated from the device input pins; also, some of the gate outputs may not be available at the device output pins, i.e. some gates may not be observable from the device input and/or output pins thereby making testing very complicated.

Due to device complexity, to test VLSI circuits from the structural viewpoint, one may have to generate excessively long

test sequences. This would require storage of long test patterns by the test equipment. It may also mean that the testing would be time consuming.

In conclusion, we can say that rigorous structural testing for VLSI circuits can yield a very high level of test confidence. But the difficulties mentioned above make it almost impossible to apply structural techniques to VLSI circuits.

Random test and functional test philosophies are also used for testing VLSI circuits and a brief description of the same is given below.

4.2 Random Testing [8,9]

The test techniques for logical circuits can be broadly classified as probabilistic or deterministic. An example of the former is the random test generation method. Methods described in section 2 of this chapter fall under the latter category.

Test pattern generators (TPG) based on the deterministic approach are not able to cope with the increasing complexity of LSI and VLSI circuit packages and their low pin-to-circuit ratio. Random testing is one way of getting around this problem. In the methods based on a random approach, random input patterns are fed to a prototype [or a simulator] of the circuit to be tested and are analyzed for their ability to detect failures. The drawback of this test method is that it produces a very large test set. To avoid this problem, the following method is

suggested. In this method a set of randomly generated patterns are applied to the primary inputs (PI's) [a primary input is one that can be applied to the device's input terminals] of the device under test. Since all PI's do not have the same functional importance [some being more important than others], the method exercises certain PI's more often than others [for definition refer to section 4.2.2].

During its development, the approach has undergone a number of changes. In the beginning, the effectiveness of purely random patterns was measured. At later stages weights were assigned to the PI's in proportion to their relative importance. Finally, the dynamic adaptive technique was developed by analyzing the rate of change of switching activity inside the logic as a result of exercising a PI.

The last modification achieved higher test coverages. A pattern reduction technique can be used to compress the patterns generated to a manageable size without any loss of test coverage.

4.2.1 Random Patterns

In this method a purely random technique assigns the same numerical weight to every primary input (PI) of the chip to be tested. Thus, each PI is exercised approximately the same number of times and the resulting patterns have the characteristic randomness. This approach is more suited to chips which contain only combinational logic because they do not require sequences of patterns; often, a set of equally weighted or random patterns are all that is required. However, this method

is inadequate for sequential circuits when functional packages such as counters and shift registers are included on the chip.

4.2.2 Weighted Adaptive Patterns

In this method weights are assigned to the PI's in proportion to their relative importance. Off-line good machine simulation is performed by software using a set of random patterns as input. Each pattern from the generator activates only one PI at a time. It is possible to count how many gates inside the chip change for the first time from logic 1 to 0 and vice versa, as the result of switching one of the PI's. The switching activity count is then accumulated over the complete set of patterns. By comparing the activity created by all PI's one can determine the relative importance of each PI through the use of this weight vector in generating new pattern sets of equal length; iteration continues until the activity count no longer increases.

Activity can be measured by observing the state of circuit inputs by themselves or in combination with their output state. Test results indicate that there is no appreciable difference in fault coverage between the above two measurements. This method proved to be a definite improvement over the equally weighted patterns. However, the method does not give very satisfactory results because of the following two problems: 1) improper handling of reset lines, and 2) the difficulty created by not having a set of weights with the simulation process.

The problem with the reset lines is that they directly

switch a large number of gates after applying a set of patterns, the reset lines consistently show a high activity count. In terms of the final fault detection patterns, over-active resets are undesirable. The second problem is that of having the adaptive weights one step behind the simulation. Hence, these weights correspond to a history of what happened in the past. Hence, it is a weight which would be more efficient in changing the state of the same gate that has already been switched. This method is not, however, predictive in the sense that it does not assign the best possible weight to the remaining gates which are not yet affected.

4.2.3 Dynamic Adaptive Patterns

This technique also measures activity in terms of gate switching states. It searches for a rate of change of activity. For example, consider two inputs to a circuit 1) clock and 2) reset line. When the random pattern simulation is performed, it is the reset line that initially shows the steepest increment activity. The activity count of the clock on the other hand rises at a slower rate, and as shown in Fig. 4.2.3-1 its final value is reached at a much later stage than that of the reset.

The faster initial slope of the reset line is due to the switching of a large number of gates which are directly driven by that line. By the time the first few random patterns have been exercised, most of the gates will have already changed states. For the next few hundred patterns, a few additional

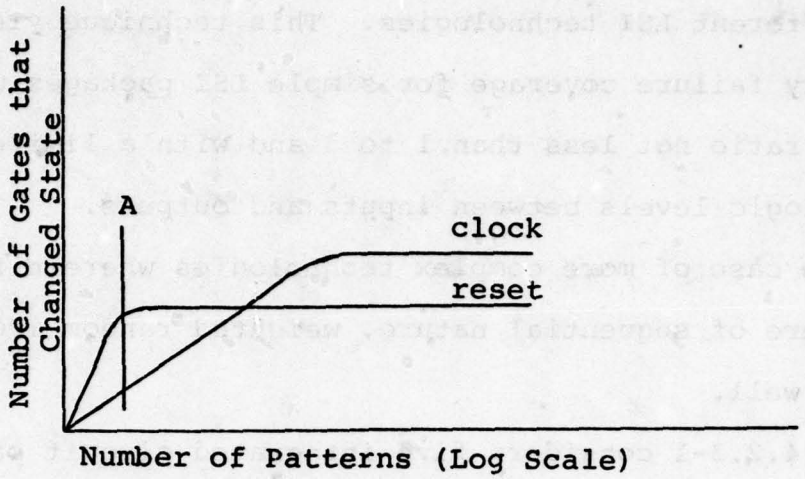


Fig. 4.2.3-1 [Ref. 9]

new gates will be switched by it. Consequently, the activity of the reset line decreases. In the previous techniques, the final weight associated with the reset line was the high value of the activity count obtained during its initial performance.

The dynamic adaptive scheme locates a point where the reset line is about to leave the knee of the curve (point A). Once this point is located, the simulation is interrupted to continue from the pattern where the interruption occurred. At the end, the clock weight [which was only slightly affected by the resetting to 0 of the activity count] will have shown a considerable increase in its activity, while reset can reach only a steady-state value which is slightly more than its value at the time of the intersection with line A. The final weight of the clock line will therefore be high, and that of the reset line low. This technique does not presuppose prior identification of functional characteristics of the clock and reset lines.

The weighted random test pattern generator (TPG) is used to test different LSI technologies. This technique yields a satisfactory failure coverage for simple LSI packages with pin to circuit ratio not less than 1 to 3 and with a limited number of logic levels between inputs and outputs.

In the case of more complex technologies wherein functional elements were of sequential nature, weighted random TPG seems to perform well.

Table 4.2.3-1 considers five integrated circuit packages of varying degrees of complexity. The number of functional elements in the circuit and the combined total of primary inputs (PI's) and primary outputs (PO's) give the topological description of the circuit. The complexity of a package is indicated by the maximum number of logic levels that separate the PI's from the PO's and the number of feedback cuts necessary to convert all the sequential elements into combinational ones.

4.3 Functional Testing [5]

The conventional methods of testing cannot handle complexities of microprocessors. A microprocessor's logic structure is not simply a collection of gates nor is it a well ordered assembly as in the large scale integrated circuit memory. The classic dc tests used to check the integrated circuits such as measuring one and zero state output voltages, can do little to ensure satisfactory microprocessor performance. In addition, the commonly used computer aided simulation technique, which tests the microprocessor with a string of inputs in a burst proves

CIRCUIT	NUMBER OF BLOCKS	NUMBER OF I/O's	NUMBER OF LOGIC LEVELS BETWEEN PI's AND PO's	NUMBER OF FEEDBACK CUTS	DETERMINISTIC TPG's		DYNAMIC WEIGHTED ADAPTIVE TPG
					PATH ORIENTED	FAULT ORIENTED	
A	60	21	17	16	93.6* 96** 0:11***	92.7 18 0:08	94.8 134 0:37
B	155	21	16	90	76.7 493 1:12	13.1 21 1:47	82.6 1116 2:37
C	236	35	25	62	38.8 185 5:37	25.8 45 2:28	86.5 1131 4:11
D	317	46	17	124	66.7 1087 4:38	81.0 834 4:19	88.4 3092 7:25
E	519	38	37	45	81.5 463 10:02	91.6 433 6:32	92.0 796 7:20

* - Testability Coverage in Percentage

** - Number of patterns after applying pattern reduction

*** - CPU time on System 360/Model 85 in minutes

TABLE 4.2.3-1 [Ref. 9]

only that the device is free of steady-state faults such as s-a-1 or s-a-0.

However, functional testing can provide comprehensive tests for microprocessors. The basic philosophy used for functional testing is that of divide and conquer. In this approach the problem is partitioned into smaller blocks. These sub-blocks are tested for proper functioning and then combined. Hence, in this approach, instead of solving a large, complicated problem at once, one tries to solve several small nonhomogeneous problems, generating several simpler results. This approach often yields efficient solutions to problems in which the sub-problems are smaller versions of the original problem.

An important step in functional testing is that of partitioning the device under test (DUT) [32]. One possible approach for partitioning the DUT is to model it as a directed graph. [A directed graph is the ordered pair D = <A,R> where A is a non-empty set of nodes (points, vertices) and R is a relation in A, i.e., R is a set of ordered pairs which are called arcs (lines, pointers).] The arcs of this graph represent the direction of flow of information signals. The system is tested by inserting various test signals at selectively located test points and monitoring and evaluating the resulting outputs at various output test points. Efficient fault detection and location methods can be developed by inserting break points within the system and priming the latter with test signals at the entry vertices and monitoring and evaluating the resulting

signal at the exit vertices. Here, the break point on an edge of the system graph simply blocks or unblocks the flow of a signal. However, it should be noted that the break points help diagnose part of the system by reducing or severing its interaction with others; those severed edges must, in turn, be tested later as a whole to verify their proper functioning.

Another way of partitioning the system would be to represent the system as a set of interconnected elements, each element having its own input and output pins. All interconnections within the system are assumed to be of one of the three types:

(1) a primary input [an input to the element is said to be primary if it can be applied directly to the element from external pins on the device];

(2) an element output to a primary output [a primary output is one that can appear on the device output pins from the elements output directly];

(3) an element output to an element input.

It will be appropriate to assume that the entire testing of the system must be accomplished through the primary inputs and outputs.

Instead of considering the functional nature of each element, one can assign a measure t which reflects the testability of the element relative to the other elements in the system. This measure could be the number of tests required to detect some percentage of stuck-at-faults or may be as simple

as a component count.

Another important aspect of the testing problem is the accessibility of these elements from the primary inputs and outputs. For example, an element with a relatively small testability measure may turn out to be quite difficult to test because it is buried deep in the system. Likewise, some primary inputs and outputs may bear a far greater share of the test load than others.

One way to do this is to picture the testing of an element [with a measure t] as a flow process. Initially, a set of t input tests or excitations are introduced onto the primary inputs of the system. These excitations then propagate through the system to the inputs of the elements involved. Here, they pass through the element and emerge on an output as a set of t detections. These detections then continue on to the primary outputs of the system, where finally they can be examined for indications of possible malfunction.

Hence, a testing process for an individual element can be modeled by postulating a "test flow" through the system but with added provision that the nature of the flow will change (from excitations to detections) when it passes through the element.

4.4 Modeling of Functional Sub-Blocks

After considering some of the methods of partitioning large systems, one can consider different techniques for modeling the functional sub-blocks. Two techniques available

are: 1) Binary Decision Diagrams, and 2) A Graph Theory Approach.

4.4.1 Binary Decision Diagrams

Once the functional division of a microprocessor (or VLSI) is carried out, a function description of the sub-blocks is required for testing. One method of achieving this would be to use different sophisticated design languages. A draw-back of such a description is that it makes a detailed logical investigation difficult. On the other hand, techniques such as truth tables, Boolean expressions and Karnaugh maps can be used for extensive analysis. However, these techniques grow exponentially with the number of variables involved.

Binary Decision Diagrams give a concise description regarding the logical structure and testing requirements of the function involved. With the aid of these diagrams, one can define a digital function diagramatically. This diagram is essentially a means to compute the output value of the function by examining the values of the inputs.

For example, consider a switching function $f = A \oplus B \cdot C$. Given values of A, B and C one can find out the values of f as follows:

Case 1: If A = 1, B = 1 and C = 1, then f = 0

Case 2: If A = 1, B or C = 0, then f = 1

Case 3: If A = 0, B = C = 1, then f = 1

Case 4: If A = 0, B or C = 0, then f = 0.

A Binary Decision Diagram for this function is as shown in Fig. 4.4.1-1.

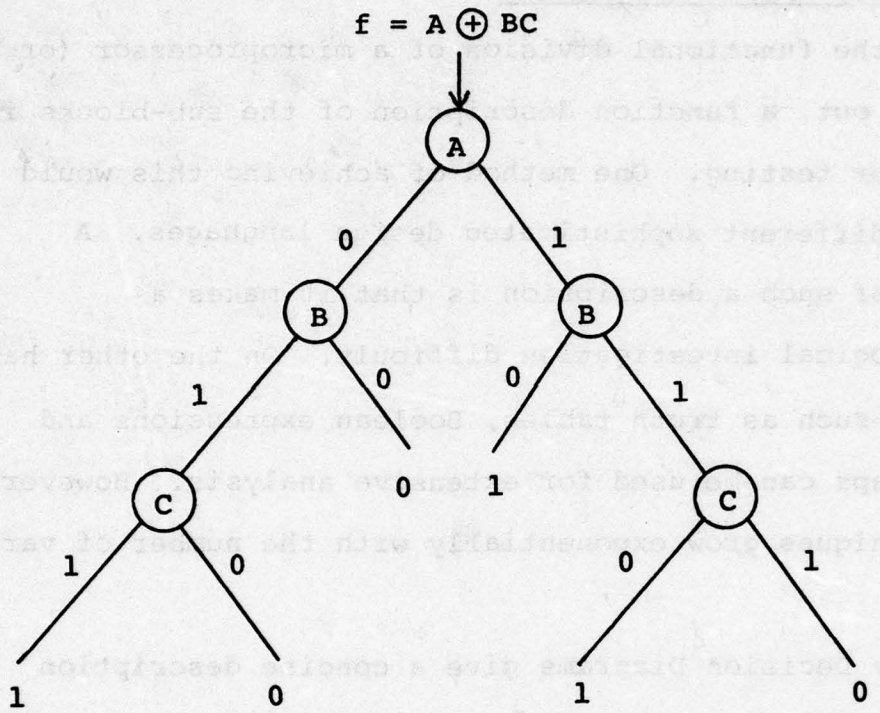


Fig. 4.4.1-1

Fig. 4.4.1-1 shows a tree structure for this procedure. [The procedure used to reduce the above tree structure is similar to the one used to reduce the structure in Fig. 4.4.1-2.] The tree is entered at the node indicated by the arrow and then proceeds downward through the diagram. The value of the variable at each node decides the branch to be followed. When a 0 or 1 value is reached, it gives the value of the function f and the process ends.

The Binary Decision Diagram can also be constructed from the truth table of the function f. This is achieved by constructing a tree that has a one to one correspondence between

the 2^n rows of the table [the truth table for the Boolean function f with n variables will have 2^n rows] and the 2^n paths to the outputs of the diagram. These outputs can be labeled with the corresponding values of f and the required diagram results. With n variables there will be initially 2^{n-1} nodes but this number can be reduced by carrying out certain reduction techniques as shown in the example below.

Consider the logic function f of three variables

$$f = \bar{B} \cdot C + (A \oplus B)$$

The corresponding truth table is:

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Table 4.4.1-1

The corresponding binary decision diagram is as shown in Fig. 4.4.1-2.

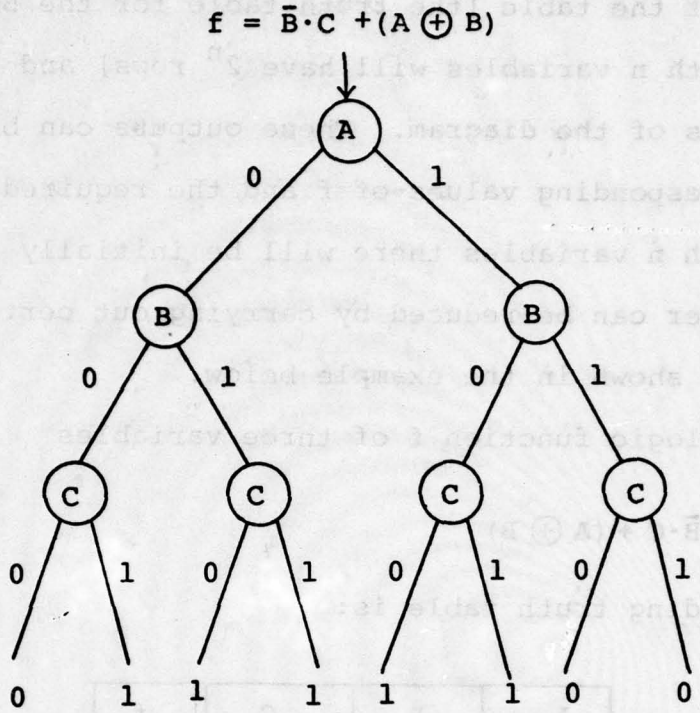


Fig. 4.4.1-2

In Fig. 4.4.1-2, note that the rightmost C-node is 0 regardless of the value of C. Hence, this node can be replaced by a 0, as shown in Fig. 4.4.1-3.

In Fig. 4.4.1-3 two rightmost C-nodes are 1 regardless of the value of C. Hence, these nodes can be replaced by 1, as shown in Fig. 4.4.1-4.

Fig. 4.4.1-4 is actually a simplified Binary Decision Diagram for the function $f = \bar{B}C + (A \oplus B)$.

If, however, the switching function is specified by a Boolean expression, a top down procedure can be used to derive the diagram by repeated application of Shannon's expansion formula:

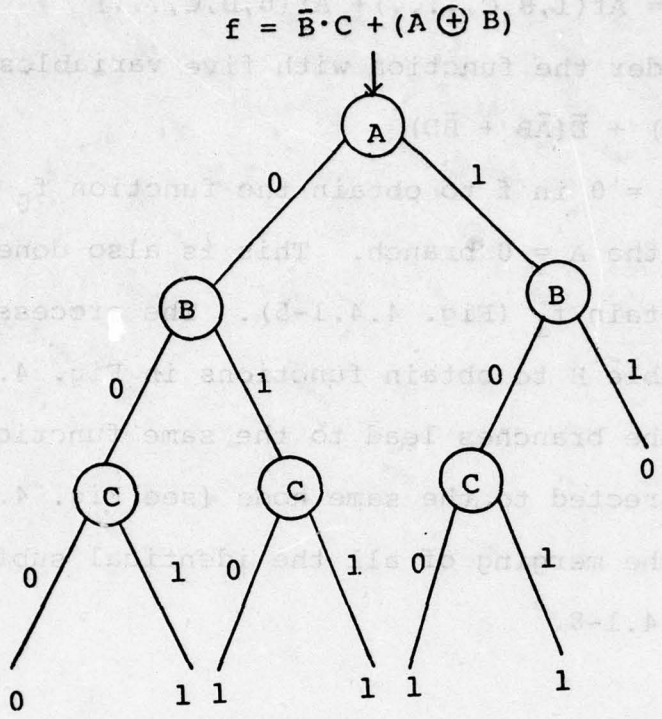


Fig. 4.4.1-3

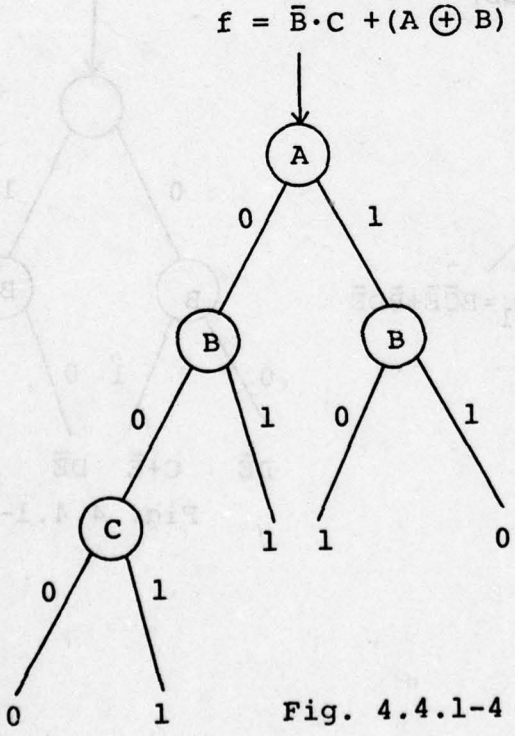


Fig. 4.4.1-4

$$f(A,B,C,\ldots) = Af(1,B,C,\ldots,) + \bar{A}f(0,B,C,\ldots)$$

For example, consider the function with five variables:

$$f = B(\bar{A}C + \bar{C}\bar{E}) + \bar{E}(\bar{A}B + \bar{B}D)$$

Begin by setting A = 0 in f to obtain the function f_0 which must be realized below the A = 0 branch. This is also done for the A = 1 branch to obtain f_1 (Fig. 4.4.1-5). The process is repeated for variable B to obtain functions in Fig. 4.4.1-6. Note that two of the branches lead to the same function ($D\bar{E}$) so these may be directed to the same node [see Fig. 4.4.1-7]. In this fashion the merging of all the identical subfunctions results in Fig. 4.4.1-8.

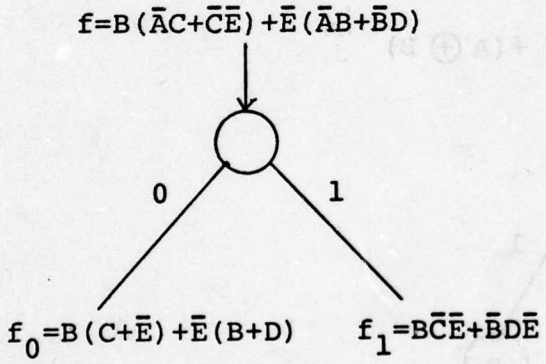


Fig. 4.4.1-5

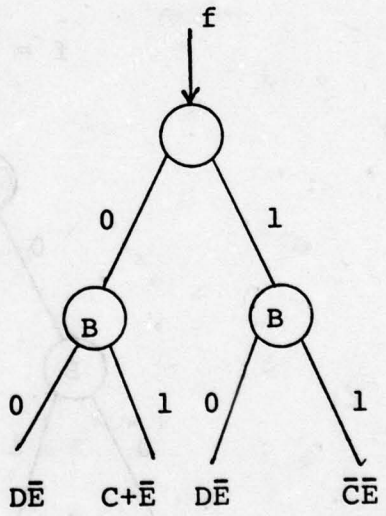


Fig. 4.4.1-6

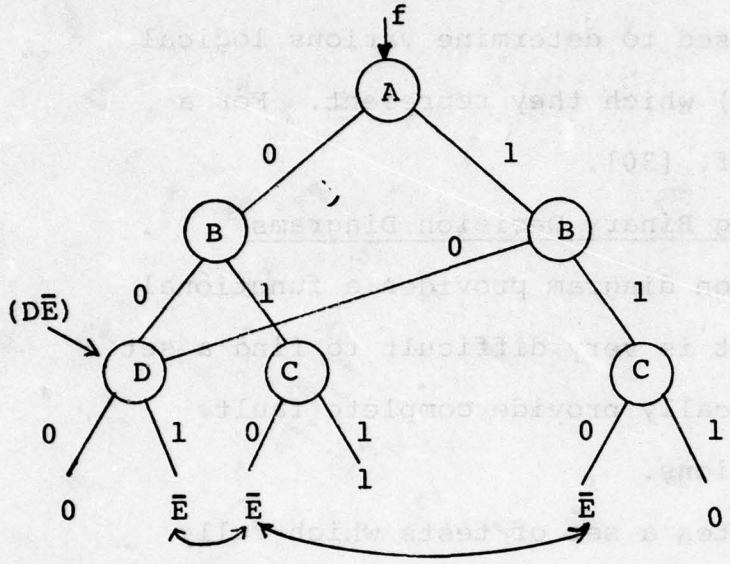


Fig. 4.4.1-7

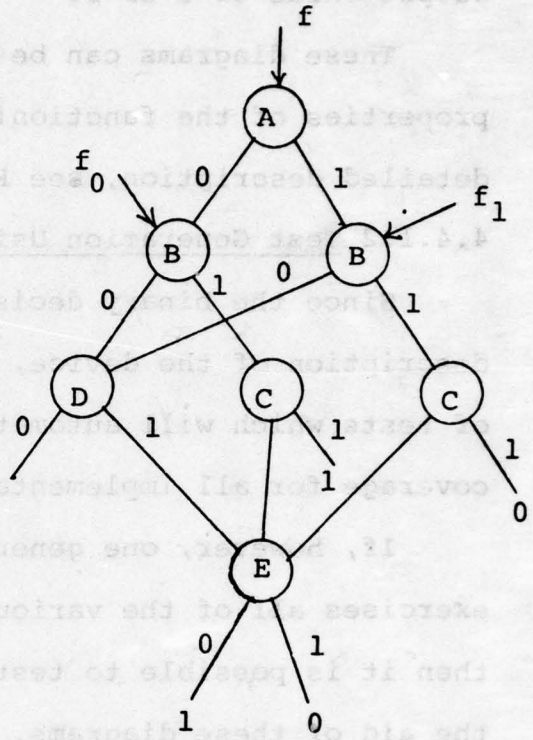


Fig. 4.4.1-8.

Typically, reductions of the diagram can be carried out by omitting redundant nodes and merging identical subfunctions, and then expanding each about one of its remaining variables until all paths terminate with a 0 or 1. Clearly, all paths will terminate in n steps.

4.4.1.1 A Property of the Binary Decision Diagrams

Each node in the diagram has two output branches and only one of them is activated for the given input. Hence, for any input exactly half of the branches in the diagram are activated

and, also each node has one and only one active path to an output value of 0 or 1.

These diagrams can be used to determine various logical properties of the function(s) which they represent. For a detailed description, see Ref. [30].

4.4.1.2 Test Generation Using Binary Decision Diagrams

Since the binary decision diagram provides a functional description of the device, it is very difficult to find a set of tests which will automatically provide complete fault coverage for all implementations.

If, however, one generates a set of tests which fully exercises all of the various nodes and branches of the diagram, then it is possible to test a reasonable functional block with the aid of these diagrams.

These diagrams are very useful to test stuck-at (SA) types of faults. Since at any time exactly half of the branches in a diagram are activated, the tracing of active paths may be less complicated compared to other testing techniques. For the diagram with n nodes in the worst case, the depth of the tree would be $\frac{n-1}{2}$. Hence, the maximum number of different choices for the test paths would be $\frac{n-1}{2}$. This number seems to be manageable compared to other methods in which the number of test paths grows exponentially.

These diagrams will be of little use if a fault creates a path between the nodes at the same level, or it creates a path between nodes at different levels than that which a binary

decision diagram allows. [The diagrams permit paths between adjacent levels.] For such a fault, the response would be unpredictable and the binary decision diagrams will not serve as a correct fault model.

However, a more comprehensive test set can be obtained by postulating various "diagram faults" analogous to the "stuck-at" faults in an actual implementation and then generating a test for discovering each of these faults. Again, such a procedure will ensure that a functional block is put through a variety of different modes of logical operations.

4.4.2 A Graph Theory Approach

This is a mathematical approach which is still in the infantile stage. Although a lot of work has been carried out in this area, little has been done in applying this technique in practice. The advantage of the technique is that it provides a mathematical background to the modeling of functional sub-blocks, thereby giving some mathematical insight into the problem.

4.5 Causes of LSI Microprocessor Failure [10]

LSI devices have all the reliability problems associated with small-scale integration (SSI) and medium scale integration (MSI) devices plus others. Since microprocessors are chips of larger area, devices are more prone to defects inherent in the semiconductor material thus increasing the probability and number of process defects such as pin holes and metalization faults. Larger packages are required to assemble the LSI devices,

thereby introducing more bonds and increasing the probability of bond failure. The larger packages are more difficult to seal theoretically, increasing their susceptibility to leaks.

Unlike the simple TTL integrated circuit, the LSI device must be tested as a system. This makes the testing more complex and less thorough relative to the testing of SSI and MSI components. Interaction between adjacent active elements of the LSI circuit can give rise to unwanted parasitic transistor action, coupled-capacitance effects, and under various combinations of logic patterns, can cause an LSI device to lose stored information--a phenomenon called pattern sensitivity.

Experience has shown that systems using LSI devices are more reliable than systems using discrete components. One contributing factor is the general improvement in semiconductors in recent years.

The results of certain failure studies for LSI show that approximately 45 percent of MOS LSI failures were chip related failures and only 28 percent were due to package and assembly-related failures. Twenty percent of the remaining failures were attributed to handling and 7 percent were due to other causes. Of the 45 percent chip-related failures, 20 percent were attributed to oxide faults. [Ref. 10]

The oxide in an MOS (metal oxide semiconductor) performs a dual function. It protects the semiconductor material and determines the operating parameters of the MOS circuit. Microprocessors, like all components, have unique failure mechanisms,

as well as mechanisms in common with other semiconductors. Typical causes of microprocessor failure [package and die related failures] may yield any failure mode depending on when and where on the chip they occur.

The assembly and package-related failure causes include 1) open-bonded wires, 2) lifted bonds, 3) lifted chips, and 4) loss of hermeticity.

Potential package-related failures are important for the following reasons. The large die size could affect the proper bonding of the die to the package. A large number of wire bonding pads and the number of external package pins required increase the probability of bad wire bonds occurring, and the large package size could present sealing problems which are noticed only by hermeticity and temperature cycling tests. The integrity of the die and wire bonds could be ensured by centrifuge tests, which must be performed with extreme care so as not to crack the package.

4.5.1 Failure Modes

Failure modes of LSI devices (memories, microprocessors and the like) are divided into two major categories:

a) catastrophic failures, and

b) soft failures.

The catastrophic failures can be attributed to the following: I) oxide rupture, II) interruption of lines, III) wire bond failures, and IV) corrosion due to contamination.

Soft failures, being hard to detect, can be simply out of specification conditions at certain operating conditions. In some instances, soft failures cannot be reproduced. For example, test system noise can cause failures and this noise cannot be reproduced. For this reason, characterization testing is important. Most soft failures are single bit failures. They occur due to one of the following reasons: slow access, loss of data in cells or multiple addressing. The advent of the microprocessor has added soft failures related to software and the interrelationship between software and hardware, such as

(1) pattern and pattern sequence sensitivity
(2) interrupt, such as trigger on wrong priorities for multilevel interrupt
(3) failures to execute instruction and/or interrupt
(4) loss of carry and bits during circulation of data
(5) instruction and instruction sequence sensitivity.

4.5.2 Electrical Testing

One of the critical areas of microprocessor reliability is that of electrical testing. How does one adequately test a Microprocessor Unit (MPU) to ensure that it has no shortcomings for all possible conditions of use? Electrically testing a microprocessor is very difficult due to the following reasons: (1) the random logic nature of an MPU, (2) the bus organization, (3) the on-chip interrelationship between functional blocks, (4) chip layout and (5) relationship between software and hardware and the like. [The microprocessor must be considered as

a monolithic system of untested parts, the constituent components of which are not accessible.]

For these reasons, it is important to perform first a characterization test program. The results of this test program are then used to develop a complete electrical test program that is both meaningful and viable.

In electrical characterization, a sample of a given device type is subjected to all practical combinations of supply voltages, temperatures, timing conditions and parametric variations.

The aim is to discover how the parts respond under these conditions and within what limits they remain functional.

The tests include stringent functional stressing by means of patterns, as well as timing and parametric variations under temperature extremes. Worst-case patterns and instructions with supply and timing variations are applied to the device to expose as many of its failure modes as possible and to determine its performance under the most severe conditions.

If the characterization is well planned, electrical characterization can provide much valuable data. The resulting volume of data must then be put into a form that is easy to interpret so that meaningful conclusions can be reached.

4.6 Testing of LSI Random Access Memories (RAM) [11]

Typically, RAM memories are manufactured using one of two technologies, bipolar and metal oxide semiconductor (MOS). The

MOS memories have a higher circuit density and, hence, have a larger memory capacity. Unfortunately, these memories store their data via the charge across a capacitor. Due to circuit leakage, this charge must be re-established at fixed intervals, otherwise the value of the bit stored will be lost. The process of re-establishing the data in the memory is called refreshing. Memories which require refreshing are called dynamic. The following parameters have a significant effect on how the memory is to be tested: 1) total capacity, 2) array configuration, 3) addressing layout, and 4) refresh parameters.

Faults can occur in the memory matrix, decoders, input buffers, read/write circuitry, data input circuitry or sense amplifiers. These faults can lead to functional failures such as inability to read or write, erroneous data storage, d.c. parametric failures such as unacceptable output levels, or dynamic failures such as slow access times. Specifically, the semiconductor RAM memories exhibit malfunctions such as:

(1) opens and shorts

(2) open decoders - the total memory cannot be truly addressed

(3) multiple wires - in the act of writing in one cell the chip actually writes in more than one cell

(4) pattern sensitivity - the contents of a cell get complemented due to read and write operations in "electronically adjacent" cells. Such an error may be a function of (a) the information being read or written, (b) the

cells being addressed and (c) the sequence in which these cells are addressed.

(5) write recovery - memory may not produce information at the specified access time when each read cycle is preceded by a write cycle

(6) sense amplifier sensitivity - memory may not respond with the proper information after reading a long series of similar data bits followed by a single transition of the opposite data value

(7) poor retention - memory loses information in less than the stated hold time [the hold time is defined as the maximum period of time the data can be stored without re-establishing its value].

The problem of pattern sensitivity arises mainly due to high component density and related effects of unwanted interacting signals. Due to this problem, the following situations may occur: (1) store (write) a value in cell K, (2) read this value several times to verify that it is indeed stored in cell K, (3) read and write in cells other than cell K, and (4) read cell K and find the value which is now wrong. This effect can occur even though each cell is capable of being correctly addressed and can individually store a 0 and a 1.

To prove that a read/write RAM is totally functional, the following aspects must be verified:

(1) Every cell of the memory must be capable of storing a 0 and a 1;

(2) The cell addressing circuits, or decoders must correctly address every cell;

(3) The sense amplifier must operate correctly;

(4) There must be no interaction between memory cells;

(5) For dynamic MOS memories, the cells must be capable of storing data for a specified time without being refreshed.

It is a difficult task to design tests to cover the five factors given above. Tests must also take into consideration the chip configuration and electronic characteristics, since each design exhibits its own unique failure characteristics. It is often very difficult to determine exactly what fault modes exist in these chips. In practice there are many different types of test patterns available for testing these memories. Each has its place or use along with certain trade-offs as to the sufficiency of the test, and test time. Certain patterns are specifically aimed at certain problem areas. The most common of these patterns are discussed below.

4.6.1 Write and Read Ones and Zeros

This is the simplest of all memory test patterns and is widely used throughout the industry, although it is of limited value. An LSI memory array could possibly have totally non-functional decoders, with one memory cell selected and connected to the input-output lines and still appear good with this test. With all the decoders nonfunctional, there will always be one memory cell that is permanently selected because each

nonfunctional decoder must be in either a permanent one or zero state. When a logic one or zero is written, the one or zero will be stored in the permanently selected memory cell and upon checking the contents of the memory array for an all ones or zeros pattern, the contents of the one selected memory cell will always appear on the output as all ones or all zeros, indicating that the device is good when it is not.

The all ones and zeros pattern is, however, useful but its application is limited due to the problems mentioned above.

4.6.2 Marching Ones and Zeros

This is a basic test to ensure that the memory is functional (that is, the addressing is operational and each cell can be written and read in the input/output state). The memory is first written to the all-zeros state. Then sequentially, starting at the first address, the zero is read and a one is written. This sequence is continued to the last location (i.e. until the memory is full of ones). Then, starting at the last location, a one is read and a zero is written. The address is reduced in location and the sequence is repeated until the first location is reached. This overall sequence is then repeated with the data reversed.

As the memory is being scanned in the ascending direction, any effect on a location above will be detected when it is eventually read. If the effect is on a location below, it will not be detected until the memory is scanned in reverse. This, by no means, tests everything or all interactions, but does

reasonably assure that the memory is working and that no defective elements are present.

4.6.3 Walking Ones and Zeros

The most widely used and most generally known test for semiconductor memories is walking ones and zeros, sometimes called ripple. This test is much more extensive than the marching ones and zeros. Initially, all locations are written to a "background" pattern of all zeros. Then starting at the first location, a "test word" of one is written. All other locations in the memory are sequentially scanned and read to verify that they still contain the background pattern of all zeros. The "test word" one is then read and written back to zero. After the first iteration, it is known that writing a one in the first location does not affect any other location. This sequence is repeated for every location in the memory. At the completion of walking one through a field of zeros, the patterns are reversed and the zero is walked through a field of ones. Overall, this test sequence results in $2(n^2 + 4n)$ tests where n is the number of locations in the memory. This test pattern is useful for testing dc pattern sensitivity, functionality and proper address operation. However, because of infrequent data transitions during the read cycles, it is not a good test for access-time determination.

4.6.4 Galloping Ones and Zeros (GALPAT)

This is one test pattern that includes testing all possible address transitions. It uses the same data pattern sequence

as walking ones and zeros. Initially, all locations are written to a background pattern of zeros. Then, starting with the first location, a test word of one is written followed by a read sequence of read location two, read location one (test word), read location three, read location one, etc., until every pair of transitions is checked. The test word is moved to the second location and the sequence is repeated, checking all transitions with the second location. This is repeated for all locations. The patterns are reversed and the overall sequence is again repeated.

The GALPAT test provides alternating data output during successive read cycles and overcomes the access-time limitation of the previous pattern. Write/read operations which are performed only every n basic cycles, are less than rigorous; however, overall, the GALPAT is a very good test procedure. Other standard test patterns used in industry for testing LSI memories include: galloping write recovery, multiple address exercise test (MASEST) and checkerboard. Table 4.6-1 gives the summary of performances of the various standard testing patterns.

The n in the test time column of Table 4.6-1 refers to the number of memory cells in the LSI memory array.

4.7 Functional Testing of Microprocessors [3]

This approach involves the following procedure:

(1) Prepare a functional block diagram by partitioning the processor into basic functional blocks. To develop a detailed

	Function-ality	Addressing	Multiple Selection	dc Pattern	Access	Write Recovery	Test Time	Comments
Write/Read 1's/0's	X	X	X	X	X	X	$4 \cdot n$	better than no test
Marching 1's/0's (Read write forward/ backward)	✓	good	poor	poor	poor	poor	$10 \cdot n$	minimal test, fast, useful for de-bugging
Walking 1's/0's	✓	good	poor	✓	poor	poor	$2 \cdot n^2$	most widely used test
Galloping 1's/0's	✓	excellent	excellent	✓	excellent	poor	$8 \cdot n^2$	all possible transi-tions; read-read
Galloping write recovery	✓	excellent	excellent	✓	fair	excellent	$12 \cdot n^2$	all possible transi-tions; write-read
Multiple address selection exercise test (MASET)	poor	poor	good	poor	fair	X	$5 \cdot n$	quick test
Checkerboard	poor	poor	poor	poor	poor	poor	$4 \cdot n$	standard core memory test

TABLE 4.6-1 [Ref. 11]

NOTE: ✓ - Yes
X - No

functional diagram requires a gate level diagram, timing diagram and a block diagram showing the major functional areas of the microprocessor and interconnecting data and control paths. However, in many cases, microprocessor vendors supply only a timing diagram and a block diagram. This is inadequate to prepare a detailed functional block diagram and one has to rely on the best available information.

For partitioning the microprocessor, the test engineer should study the hardware architecture and software response specifications of the microprocessor under test. Architecture refers to the internal organization of the device: an ordered set of modules such as a program counter, arithmetic logic unit (ALU), accumulator, stack pointer, etc. Software response refers to applying a set of instructions to the microprocessor under test in order to monitor the operation of various modules. After familiarization with the above, an ordered set of test sequences can be developed in the microprocessor's programming language for testing the modules.

There is a wide variety of microprocessors available on the market today and each has its own architecture; of all the product types, 8 bit units like the Intel 8080, Motorola 6800, and MOS 6500 series, have gained the widest market acceptance. Block diagrams of the Intel 8080 and Motorola 6800 are as shown in Figures 4.7-1 and 4.7-2 respectively.

Typically, an 8 bit microprocessor [refer to Figs. 4.7-1 and 4.7-2] has two internal busses, an 8-bit bi-directional

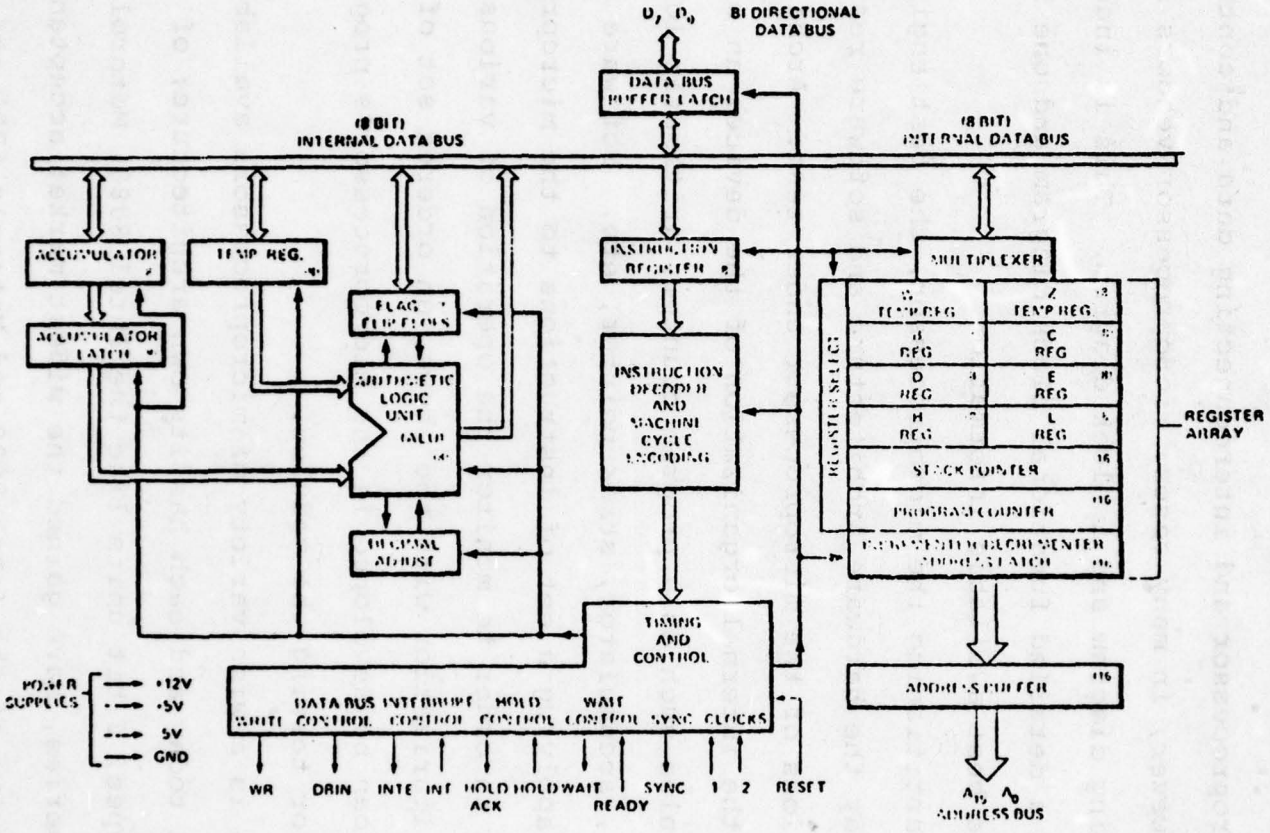


Fig. 4.7-1 8080 CPU Functional Block Diagram [38]

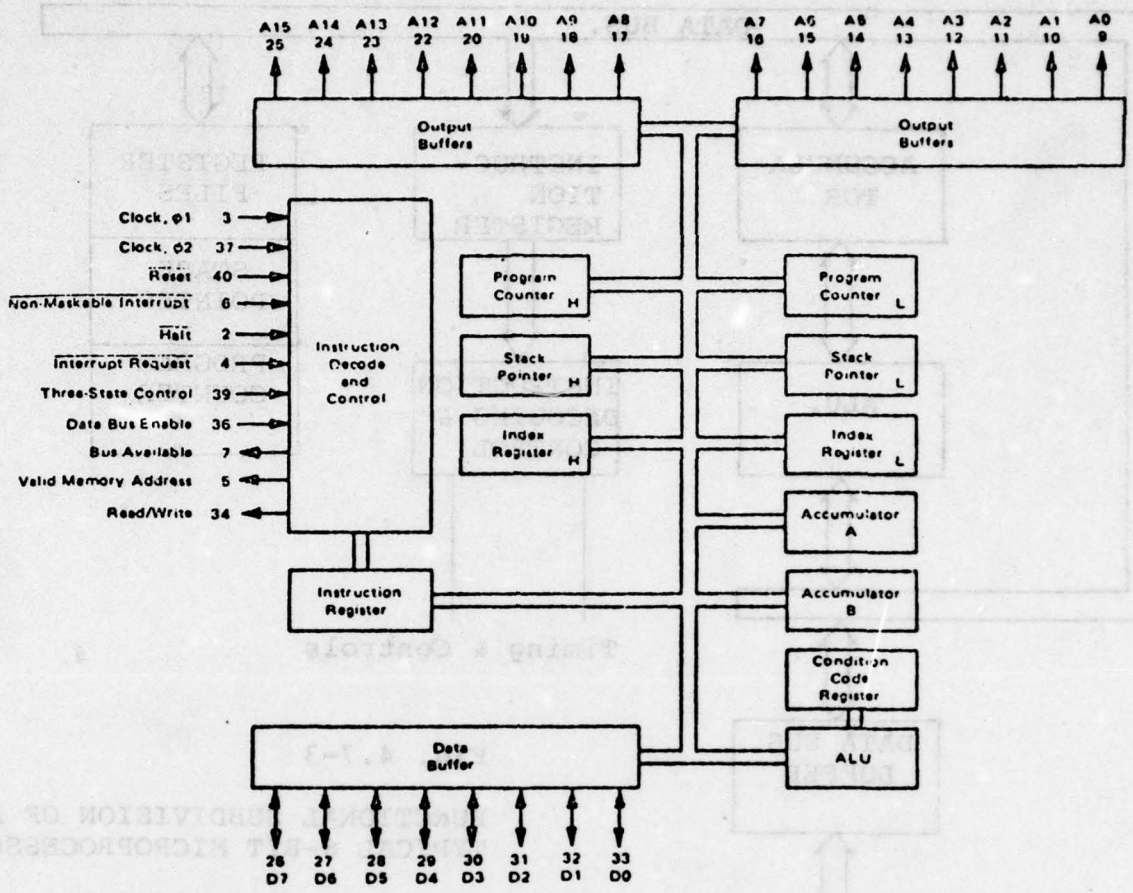


Fig. 4.7-2 Mc6800 Block Diagram [39]

data bus and a 16-bit unidirectional address bus. The typical functional block diagram of such a processor would resemble Fig. 4.7-3. The data bus carries both instruction codes and data. The instructions are decoded and executed in connection with the appropriate controls and the data goes to both the arithmetic logic unit and the accumulator to be manipulated by specific

---

This test philosophy was developed by General Electric for RADC.

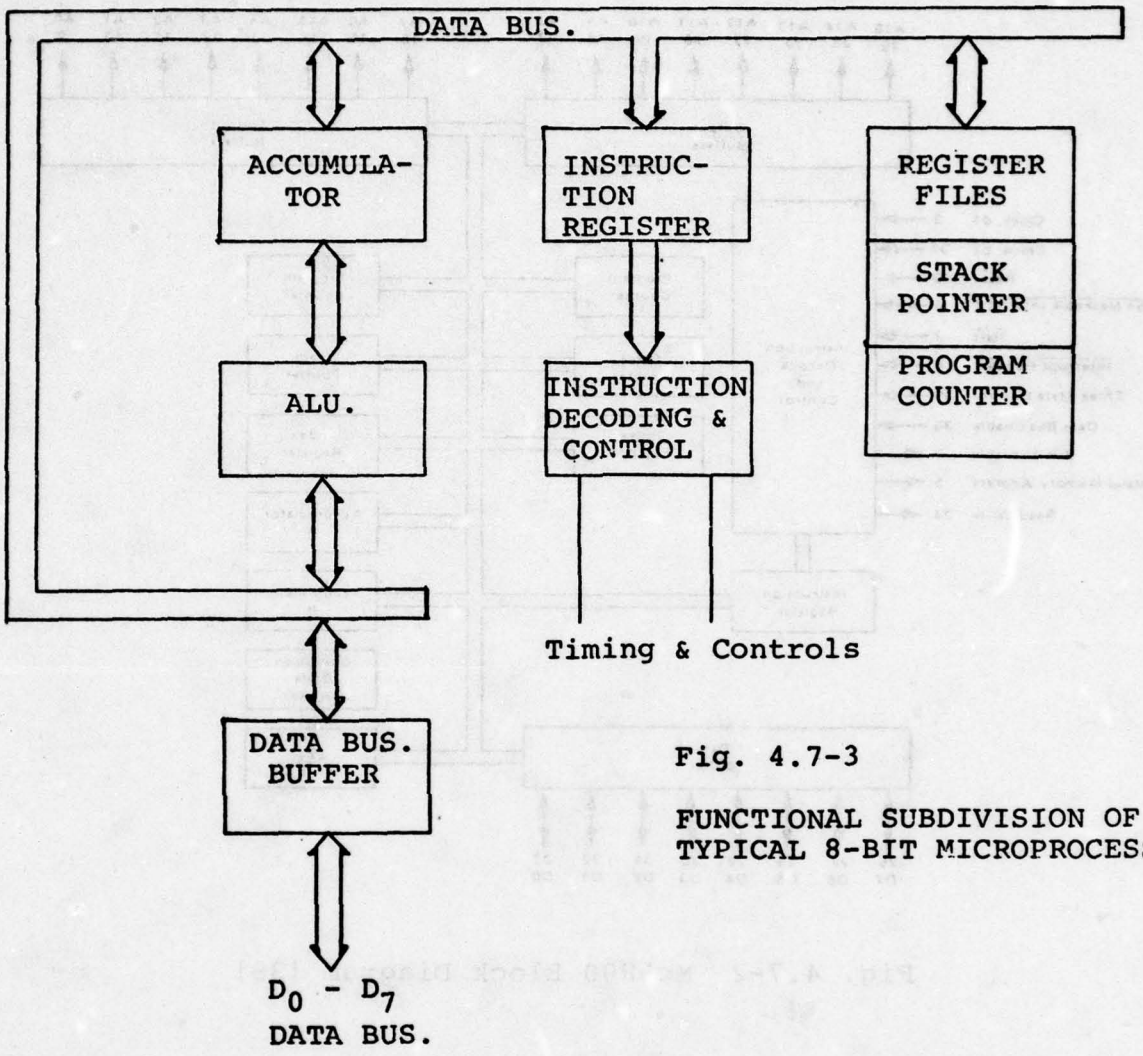


Fig. 4.7-3

FUNCTIONAL SUBDIVISION OF A TYPICAL 8-BIT MICROPROCESSOR

arithmetic or logic operations. The address bus links with the main memory where both instruction codes and data are stored. Stack pointers, program location counters and register files also supply information to the address bus.

(2) Test each functional block using proven test patterns. [This testing should yield a high test confidence

level.] Some of the functional blocks can be exhaustively tested using only a few test vectors.

The following is a typical procedure to achieve this.

4.7.1 Pin Independence

This is achieved by verifying that while each pin assumes a one and a zero state, all of the other pins - either individually or collectively - are in the complement state. The states of the input pin have to be monitored unless they are sensitized to the outputs.

4.7.2 Testing of Control

Such a testing should verify:

(1) Each control line performs the intended function. This is achieved by activating each control line and verifying whether address, data and status lines assume the correct states.

(2) Each control line will perform independently of the previous instruction. This is achieved by activating the control lines in a gallop type test and checking the response.

(3) The proper priority is maintained when two or more control lines are activated at the same time.

(4) Some of the control lines will have high impedance capability (Tri-State lines). [For example, some of the tri-state control lines in Mc6800 are Data Bus Enable, Bus Available and Valid Memory Address (VMA).] These lines and their associated circuitry should be activated and checked for proper operation.

Tri-state lines are checked by placing them in a high

impedance state and measuring the leakage current. This should be tested with the input of the tri-state buffer activated to both a one and a zero.

4.7.3 Data Lines

(1) These are tri-state lines and they should be subjected to the same test as in step 4.7.2-4.

(2) Line independence - This is accomplished by checking that while each line assumes a one and a zero state, the rest of the lines - either individually or collectively - are in the complement state.

4.7.4 Testing of Multiplexers

Test if a multiplexer can pass both a one and a zero in each selection position.

4.7.5 Testing of Arithmetic Logic Unit (ALU)

(1) Testing of serial adders/subtractors:

(I) With carry = 1, apply all possible inputs, i.e. 0 and 0, 0 and 1, 1 and 0 and 1 and 1 to each input pair.

(II) Repeat (I) with carry = 0.

Steps I and II should be carried out for both the add and subtract modes.

Typically, for the 8 bit microprocessors the subtractor is a one's or two's complement subtractor.

Two's complement subtraction is performed by complementing the subtrahend and:

(I) For single precision arithmetic, adding with carry-in equal to "one."

(II) For multiple precision arithmetic, adding with carry-in equal to borrow from preceding bytes.

Hence, a test for subtraction requires verification of the addition function and then verification of the complementing circuitry. This requires the following tests:

(I) Verify the eight possible inputs to each bit of the adder.

(II) Verify that the complementing circuitry will complement both a one and a zero for each bit.

(III) Verify decimal adjust circuitry.

(2) Testing ALU for logic operations:

(I) For each input pair of the ALU, apply the following input conditions:

(a) 0 and 0, 0 and 1, 1 and 0 and 1 and 1 during Exclusive OR operations

(b) 0 and 0, 0 and 1, and 1 and 0 during logical OR operations

(c) 0 and 1, 1 and 0 and 1 and 1 during AND operations.

(II) Verify that for shift left and for shift right operations, both a "0" and a "1" are shifted from each bit into a "1" and a "0" respectively, in each adjoining bit. This requires four shift left and four shift right combinations, i.e. this would verify 0 to 0, 0 to 1, 1 to 0 and 1 to 1 transitions.

(3) Testing of Flip Flops. Check flip flops for 0 to 0, 0 to 1, 1 to 0 and 1 to 1 transitions.

(4) Verify that the carry-in has no effect on the ALU during logic functions.

4.7.6 Checking of ALU Flag Signals

Check that special outputs such as carry, carry generate, carry propagate and overflow from an ALU operate properly, i.e. assume both a one and a zero state and that they occur at the proper time. If Boolean equations are provided for their generation, verify that each of the terms in the equations affects the outputs.

4.7.7 Verification of Instruction Set

Execute each instruction or op code at least once to verify the instruction set. Checking that only the intended instruction is performed verifies that the decode circuitry is functioning properly.

4.7.8 Testing of Processor Registers

This includes the following:

(1) Verify register independence. This is achieved by writing into one of the registers and checking that others are unaffected.

(2) Verify bit independence. This is achieved by checking each bit for the zero and one state with respect to all other bits which are in a complement state.

(3) Verify integrity of unique registers. (Here, by unique, we mean registers such as accumulators, stack pointers, index registers, storage registers, etc.).

This is achieved by ensuring that transitions from 0 to 0, 0 to 1, 1 to 0 and 1 to 1 are possible for each bit of each static register.

If the device under test contains a RAM or if the unique registers can be configured as a RAM, the following items should be checked.

4.7.9 Testing of Processors Containing RAM

(1) Address Uniqueness. If a memory has n words, verify that there are n independent word locations or verify that unique registers are independent. This is achieved by writing into an address or register and verifying that it was the only address or register affected. The standard RAM tests used for this are walking-one, walking-zero, galloping-one, galloping-zero, or write recovery.

(2) Bit Independence. This is achieved by verifying each bit for a zero and a one state with respect to all other bits which are in the complement state. This is accomplished by a walking-one, walking-zero type of test.

(3) Cell Integrity. Check that transitions from 0 to 0, 0 to 1, 1 to 0 and 1 to 1 are possible for all bits. For multiple bit RAM's, this can be achieved by walking-one, walking-zero test. For single-bit RAM's, separate tests have to be performed for the 0 to 0 and 1 to 1 transitions.

(4) Cell Independence. For dynamic RAMs check for inter-cell disturbance. Maximize the number of internal transitions to test for cell to cell interaction. This can be achieved by a galloping-one, galloping-zero type of test.

(5) Data Retention for Dynamic RAMs. This is the same as the test number 4.7.10 [for certain devices, it may be necessary to check the following items:

(6) Write Recovery. Verify that transitions from write to read do not cause access time failures. This is verified by checking all possible transitions from write to read.

(7) Read Modify Write Recovery for Dynamic RAMs. Check that transitions from a read to a write do not cause incorrect information to be written into the device. This is achieved by checking all possible transitions from read to write.

(8) Sense Amplifier Recovery for Dynamic RAMs. Test for sense amplifier frequency response by repeatedly reading 1-0 data patterns at minimum read cycle time.

4.7.10 Verification of Dynamic Registers and Busses.

This test should check that enough charge is transferred in a minimum transfer time and that sufficient charge is available after the maximum storage time. This is achieved by varying the power supply voltages and clock amplitudes, periods, widths, and delays to set up the worst case condition mentioned above.

4.8 Available Test Methods for LSI Testing

4.8.1 Signature Analysis [29]

This technique utilizes a portable tester which essentially compresses a multiple-bit burst into a form that can be easily handled, without an undue amount of software. One method used in large systems is transition counting [described in section (6)]. The signature analysis method is based on the data compression technique called cyclic redundancy check codes (CRC) borrowed from the telecommunications field. A cyclic redundancy check code is a form of a check sum produced by a pseudorandom

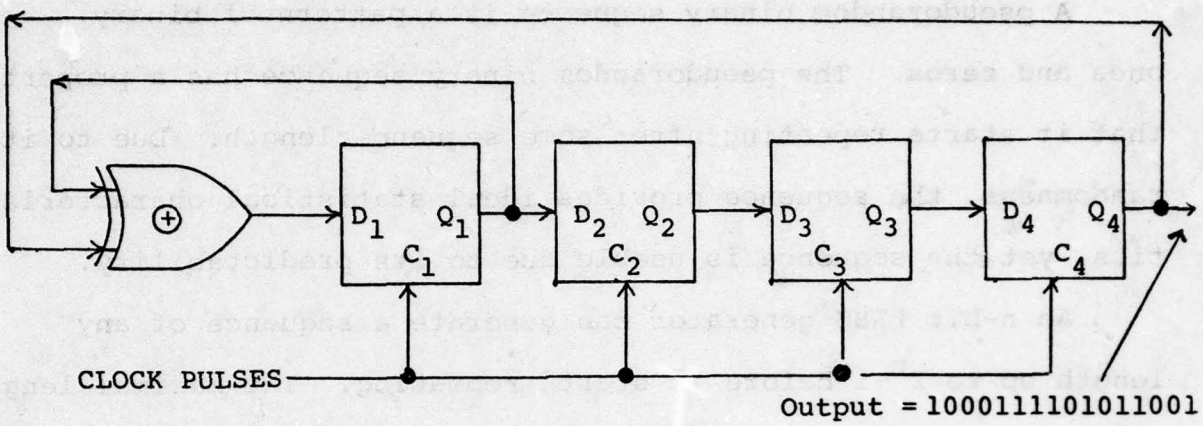
binary sequence (PRBS) generator.

4.8.1.1 Pseudo Random Binary Sequences (PRBS)

A pseudorandom binary sequence is a pattern of binary ones and zeros. The pseudorandom binary sequence has a property that it starts repeating after some sequence length. Due to its randomness, the sequence provides ideal statistical characteristics, yet the sequence is usable due to its predictability.

An n-bit PRBS generator can generate a sequence of any length up to 2^n-1 before it starts repeating. The maximal length generator is the one that repeats exactly after 2^n-1 bits. Such a generator will generate all possible n-bit sequences except a string of n zeros. For example, consider a fifteen bit sequence 100011110101l001. This bit pattern is produced by a four bit maximal length PRBS generator (since 15 = 2^4-1). In this sequence, all possible bit patterns occur only once and then the sequence starts repeating.

The construction of a PRBS generator is based on Galois field arithmetic. The Galois field of two elements has an alphabet of two symbols, 0 and 1, together with modulo-2 addition [i.e. 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 0] and multiplication [i.e. 0·0 = 0, 0·1 = 0, 1·0 = 0, 1·1 = 1]. For this reason, in a PRBS generator there exists only two types of operating elements. The first is a modulo-2 adder [also known as exclusive OR gate] and the other is a simple flip flop (say D-type) which acts as a time delay of one clock period. A shift register can be constructed by connecting



	Cycle	Q_1	Q_2	Q_3	Q_4	$D_1=Q_1 \oplus Q_4$
Initial State	0	0	0	0	1	1
	1	1	0	0	0	1
	2	1	1	0	0	1
	3	1	1	1	0	1
	4	1	1	1	1	0
	5	0	1	1	1	1
	6	1	0	1	1	0
	7	0	1	0	1	1
	8	1	0	1	0	1
	9	1	1	0	1	0
	10	0	1	1	0	0
	11	0	0	1	1	1
	12	1	0	0	1	0
	13	0	1	0	0	0
	14	0	0	1	0	0
Begin to Repeat:	15	0	0	0	1	1

Fig. 4.8.1.1-1

these flip flops in series, as shown in Fig. 4.8.1.1-1. By taking the outputs of various flip-flops, exclusive ORing them and feeding the result back into the register input, a feedback

shift register is obtained that will produce a pseudo random sequence. With the proper choice of feedback, the sequence will be maximal length. The fifteen bit sequence considered earlier was produced by the PRBS generator of Fig. 4.8.1.1-1, with the flip flops initially in the 1000 state. [Note, the all-zero state is not allowed.] The table in Fig. 4.8.1.1-1 shows the pseudo random sequence. This list contains each of the sixteen ways of arranging four bits except four zeros.

Figure 4.8.1.1-2 shows the same feedback shift register with an external input. In this case one can superimpose input data onto the pseudo random sequence. The superimposed data changes the sequence generated by the generator.

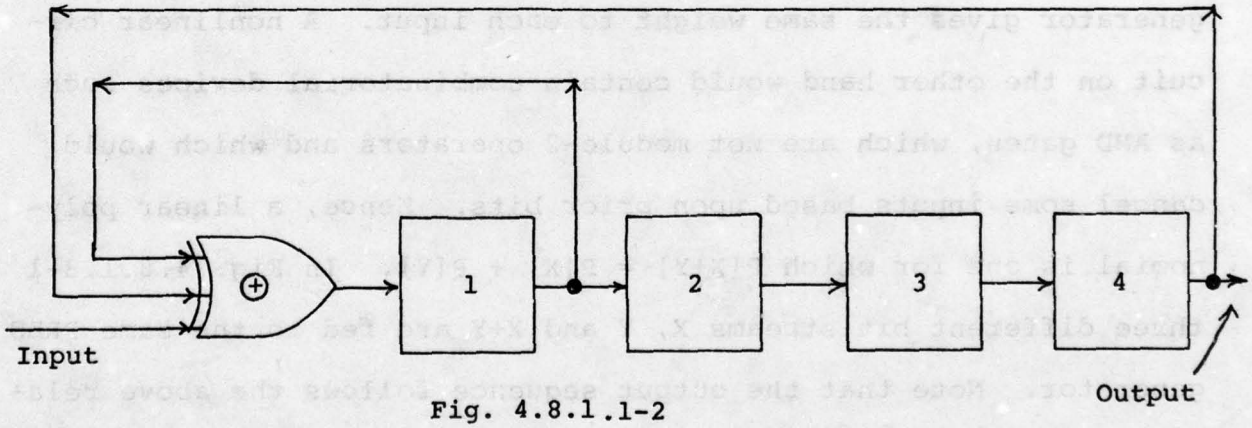


Fig. 4.8.1.1-2

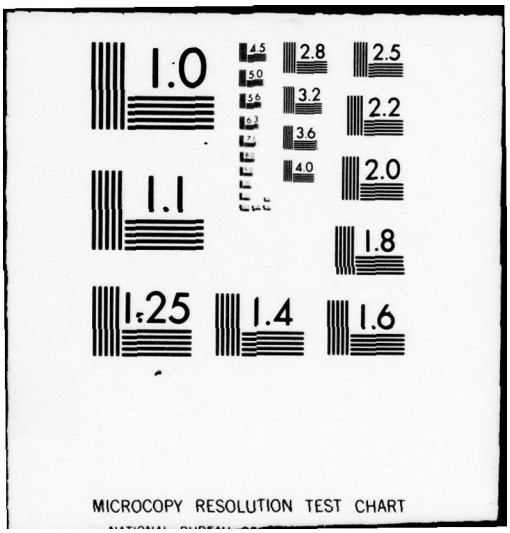
4.8.1.2 Shift Register Arithmetic

Let x(t) denote an input sequence; let D be a transform operator such that x(t) = Dx(t-1). Multiplying by D is equivalent to delaying data by a unit of time. In Fig. 4.8.1-2, the

AD-A070 733 CLARKSON COLL OF TECHNOLOGY POTSDAM N Y F/G 14/2
A SURVEY OF LSI TEST METHODOLOGY.(U)
MAY 79 U V GUMASTE, R M MATTHEYSES F30602-75-C-0082
UNCLASSIFIED RADC -TR-79-85 NL

2 OF 2
AD
A070733

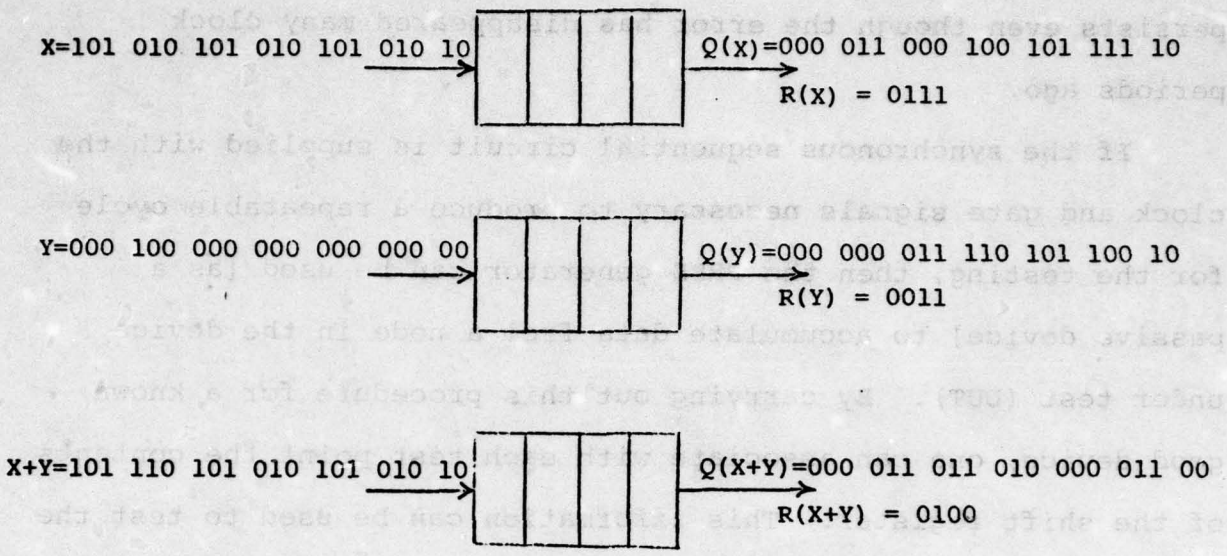
END
DATE
FILMED
8-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART

data entering the register is the sum of samples taken after one clock period and four clock periods, along with the input data itself. Hence, the feedback equation can be written as $D^4x(t) + Dx(t) + x(t)$ or simply $x^4 + x + 1$. The characteristic polynomial for the feedback shift register is the inverse of the feedback equation. Let the feedback polynomial for the n stage (n flip flops) register be denoted by f(x). The characteristic polynomial is given by $x^n f(x-1)$. Hence, the characteristic polynomial for the shift register in Fig. 4.8.1.1-1 is $x^4 + x^3 + 1$ which is the inverse of the feedback equation.

Feeding data into a PRBS generator is equivalent to dividing the data stream by the characteristic polynomial of the generator.

Since the PRBS generator is constructed from the Galois field elements [Modulo-2 adder], it is a linear sequential circuit. This generator gives the same weight to each input. A nonlinear circuit on the other hand would contain combinatorial devices such as AND gates, which are not modulo-2 operators and which would cancel some inputs based upon prior bits. Hence, a linear polynomial is one for which P[X+Y] = P[X] + P[Y]. In Fig. 4.8.1.3-1 three different bit streams X, Y and X+Y are fed to the same PRBS generator. Note that the output sequence follows the above relationship, i.e. Q(X+Y) = Q(X) = Q(Y). Y is a single impulse bit delayed in time with respect to the other sequences and it may be noticed that X and X+Y differ in only bit, viz. 4th bit from the left. However, Q(X+Y) looks nothing like Q(X). Let R(X), R(Y) and R(X+Y) be the remainders in the registers generating X, Y and X+Y sequences respectively.



Note: (1) Contents of the above registers are initialized to 0000
(2) Principle of superposition holds, Q(X+Y) = Q(X)+Q(Y)
(3) Here, "+" means exclusive OR operation

Fig. 4.8.1.3-1

If we stop after entering only twenty bits of the sequence and compare the remainders, they would be R(X+Y) = 0100 and R(X) = 0111.

4.8.1.3 Error Detection Using PRBS Generator

In Fig. 4.8.1.3-1 the x input can be taken to represent an input data stream; X+Y can be considered as representing an erroneous input with Y an error sequence. Since by stopping the PRBS at any time and comparing the remainder in the register with the expected bit pattern, single bit errors will always be detected. This error detection capability does not depend on the length of the input sequence. In Fig. 4.8.3-1 R(X+Y) is different from the correct R(X) and the effect of the error

persists even though the error has disappeared many clock periods ago.

If the synchronous sequential circuit is supplied with the clock and gate signals necessary to produce a repeatable cycle for the testing, then the PRBS generator can be used [as a passive device] to accumulate data from a node in the device under test (DUT). By carrying out this procedure for a known good device, one can associate with each test point the contents of the shift register. This information can be used to test the failing device. Since the PRBS remainder depends on every previous bit that has entered the generator, this is an identifying characteristic of the data stream; hence, this labeling of the node is termed as signature and the technique is termed as "signature analysis."

In conclusion, a feedback shift register with n stages will detect all errors in a data stream of n or fewer bits since the entire sequence will remain in the register. (For mathematical proof, see Ref. [29].) For data streams of greater than n bits in length, the chance of detecting an error using a PRBS is very high for generators of reasonable length. The undetected errors are predictable and, furthermore, such an error detecting method will always detect a single-bit error regardless of the length of the data stream.

4.8.2 Transition Counting [1]

Consider a digital circuit, which under fault-free conditions produces an output sequence Z in response to an input

sequence X. When testing a circuit, assume the output response observed was $\underline{Z}'$ to an input sequence X. If $\underline{Z} \neq \underline{Z}'$, then the fault is said to be detected. There are two problems associated with this testing: (1) the entire fault free response $\underline{Z}$ has to be stored for comparison with $\underline{Z}'$, and (2) $\underline{Z}'$ must be compared with $\underline{Z}$ bit by bit. This requires a sizeable amount of hardware, especially if this comparison is to be made at the DUT's clock rate.

In order to circumvent these problems, one can apply a function f to the output response of the DUT and compare $f(\underline{Z})$ with $f(\underline{Z}')$. Several different choices for the function f exists, namely, "the number of 1's in the sequence," or "the number of 0 to 1 and 1 to 0 transitions in the sequence." The latter is called the transition count (TC) and is employed in some popular commercial testers.

Properties of Transition Count Testing.

Let $\underline{Z} = Z(1)Z(2)...Z(n)$ be any n-bit binary sequence [for simplicity assume that the circuit has only one output). Then the transition count $C(\underline{Z})$ of $\underline{Z}$ is given by the equation

$$C(\underline{Z}) = \sum_{i=1}^{n-1} Z(i) \oplus Z(i+1)$$

where $\sum$ denotes an arithmetic summation and $\oplus$ denotes an exclusive OR operation. For example, consider the circuit in Fig. 4.8.2-1. The input sequence and the transition count for each signal are as shown. The output sequence is $\underline{Z}$ = 1001 and

$C(\underline{Z}) = 2$. If the output of G_2 is s-a-1, we obtain $\underline{Z}' = 0001$ and $C(\underline{Z}') = 1$. Hence, this fault changes the transition count from 2 to 1 and therefore the fault is detected.

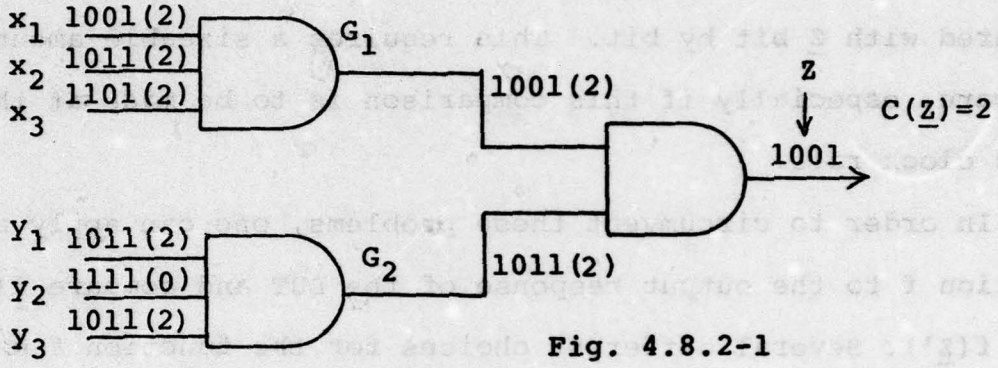


Fig. 4.8.2-1

Since actual responses need not be stored, the testing is quite simple. The transition count for the DUT can be determined by feeding outputs of the DUT into counters. Hence, very simple Automatic Test Equipment can be used. For an n-bit output sequence, an m bit counter is required, where m is proportional to $\text{Log}_2 n$.

Typically, a circuit is tested using this method by first applying a long test sequence to a fault free circuit and recording the transition counts at each of the circuit outputs. This information is used in determining the location of a fault. The input sequence is generated by a hardware pseudo-random number generator.

This method does not require any modeling and associated computer test generation and/or simulation. Automatic Test

Equipment (ATE) used in this technique is very simple compared to the equipment which must process test programs and fault dictionaries. This ATE can therefore operate at much higher test rates.

One of the problems associated with this method is that it may not yield a high test confidence level (TCL) since only some function of the output sequence is verified and not the entire sequence. For example, if a number of ones in a sequence is a test criterion, then it could happen that the sequence may have the desired number of ones, but, however, they may not be located in the proper positions in the sequence.

Since the device under test is tested at a very high rate on input patterns which are not necessarily functional, there is a chance that races or hazards may occur in the DUT. In this case, it is possible that the DUT will be classified as faulty when, in fact, it is not. Careful programming of the input specification will tend to minimize this problem.

4.8.3 Signature Analysis Versus Transition Counting

Plots of probability of error using a transition counter and PRBS generator (Fig. 4.8.3-1) show that the transition count method appears worst on single bit error which is where the PRBS generator never fails. Overall, the transition counter seems to detect at least half of all the errors; however, even a single bit shift register could do this. The four-bit PRBS generator will always detect better than 93 percent

of all the errors. Adding one more bit to the feedback shift register would halve the rate of misses.

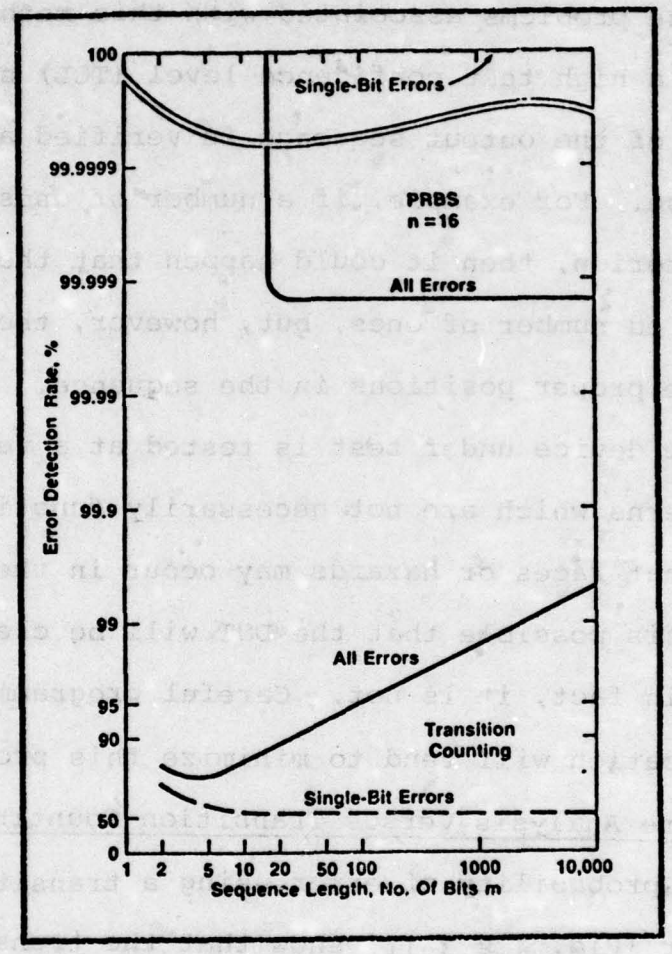


Fig. 4.8.3-1 [Ref. 29]

4.8.4 Comparison Testing

This technique is typically used in device testing. A vector test applies a set of test patterns (usually supplied by the designer of the microprocessor) to both the MPU under test and a known good MPU. Outputs from both devices are compared against one another to determine the functionality of the device under test. This approach results in considerable savings in buffer memory size as it is not necessary to provide an expected output pattern. Partial diagnosis information is available since the designer usually derives the test pattern by aiming at a specific block of logic inside the MPU or at a specific sequence of instruction. However, the technique suffers from some drawbacks. For example, the dependence on a designer to specify the test pattern tends to limit the flexibility to change or modify the input test pattern, and also makes the degree of testing somewhat questionable. In addition it is questionable as to whether the known good device is functionally good. Another question which needs to be answered is the following: Is a comparison test showing "pass" results necessary because both the known good device and DUT are equally unable to meet the test conditions? This problem can be handled by operating the known good device in a totally separate "benign" environment in which all timing relationships (except period), and all biases are at the most favorable values. Worst case or stress conditions are seen only by the DUT.

Another criticism regarding this method is that the DUT cannot be tested at speeds faster than the known good device's speed. As a practical matter, by selecting premium reference parts for a known good device, a comparison system can be operated at speeds considerably above the specification values for incoming inspection. Thus, for both production tests and incoming inspection, this peculiarity of comparison testing is of little practical importance. However, the engineering user, concerned with modifying or advancing device technology, may encounter another problem, in that it is clearly not possible to test the very first part of a new design using any comparison technique, unless an alternative technology or a simulated device is available.

4.8.5 Algorithmic Pattern Generation

In this method, the defined sequence of patterns can be created using a high speed pattern generator under microprogram control. The input and output patterns are then generated during the functional test and compared with those from the device under test. The method may be very complex, as in the case of full hardware simulation, or relatively simple, as in the case of partial or sectional simulation.

Due to the modular structure inherent in the microprocessor, the algorithmic pattern generation method lends itself to the modular sensitization technique. Each module is sensitized by a sequence of generated stimuli which simulates the

actual microprocessor instruction and the devices true output response is controlled by the pattern generator.

This method allows users to generate their own patterns and modify them at ease. The algorithmic pattern technique can also solve the problem of the overhead data transfer time. Algorithmic generation of patterns occurs at a speed comparable to the device speed, thus a substantial amount of overhead time, experienced by the storage pattern method, is eliminated.

Some of the commerical test systems (e.g. Macrodata MD-501) are specifically designed for algorithmic pattern generation as well as pattern storage testing.

4.8.5.1 Stored Response Method

This method can be subdivided into two classes: (1) learned pattern response, and (2) predicted pattern response. In either technique, the DUT is tested with a complex test pattern (on a cycle by cycle basis) stored in the bulk memory. In the learned response technique coded instructions are executed on a known-good device to learn the fault free output response. Both input and output responses are saved in the bulk memory and are used later on the test device.

The predicted response technique, on the other hand, does not require a "known good device" for generation of the test patterns. Such methods as logical or functional simulation are used to predict the input and output responses required for assembling the test pattern. The predicted response technique was used by the Hughes Aircraft Co. to characterize the 8080

microprocessor. The key technique of this method is the development of a software test pattern generator which includes a detailed functional simulator. This simulator compiles a series of microprocessor mnemonic commands into complete test patterns which includes both input and output responses from the DUT.

4.8.6 Computer Simulation

In this method a computer with an appropriate software simulation program is used as a model for the DUT. The output patterns from this model are then compared with those generated by the DUT.

This scheme requires minimum device programming effort and allows full detection and possible isolation of any catastrophic fault of the stuck-at-1 or stuck-at-0 type.

The method is well suited if one assumes that faults in MPU are only of s-a-1 and s-a-0 type. However, in reality, the situation might be very different since MPU can have functional faults, instruction pattern sensitivity type of faults, etc., which may not be discovered by the software simulation. For this reason a computer simulation may not be able to yield a high test confidence. If, however, one decides to consider an MPU model with stuck at (SA) faults and also some of the above mentioned faults, then the simulation task will be enormous. Some unpredicted faults like instruction pattern sensitivity may be very difficult to model in software. This method also lacks the flexibility to change or modify the input/output (I/O) patterns due to the fixed sequence of the

pattern provided by the simulation program. The test system based on this method will tend to be expensive as simulation programs are difficult and time-consuming to write and require a powerful computer to execute the simulation program at a reasonable speed. Also, excessive buffer memory may be required to store all simulated I/O patterns.

CHAPTER 5

LSI TEST SYSTEMS

5.1 Introduction

There are two major categories of automatic test equipment (ATE) called Stored Program ATE and Random Comparison ATE. The stored program tester typically contains a minicomputer and bulk storage such as a disc. The test sequence is stored on a vector by vector basis or as a high level program, interpreted by the computers stored program ATE, which [typically] also stores the expected response and a fault dictionary. The actual test sequence can be obtained by using an algorithm procedure [e.g. D-Algorithm, ATVG program, etc.].

Since the test vectors must be processed in a minicomputer, the average rate of applying test vectors to the device under test varies from about 200 K-Hz to 20 K-Hz, although in the burst mode [i.e. for short sequences] higher rates are possible. Due to a number of factors, such as the effectiveness of the stored programs, the need for simulation processing and a slow rate of test application, stored program test sequences are typically not of great length.

Random Comparison Type ATE use pseudo random test patterns as test vectors. These patterns can be generated in many ways and are sometimes not totally random. The vectors are not stored in a minicomputer, but are generated at very high test rates (1-40 MHz). Errors on the output of the DUT

can be detected using a comparison scheme as shown in Fig. 5.1-1.

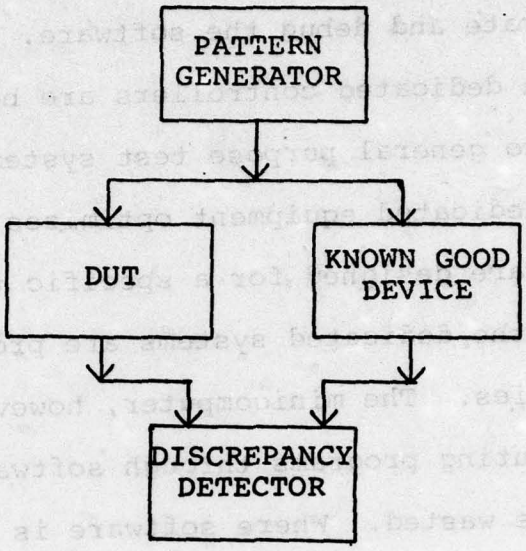


Fig. 5.1-1

Here the DUT and a known good copy of the DUT are inserted into the ATE. The pattern generator then applies several pseudo random patterns to both the devices and the outputs are compared by the discrepancy detector. A mismatch would indicate a fault in the DUT.

From an applications viewpoint the ATE can be subclassified into two categories - the dedicated tester and the general purpose tester. The dedicated tester is an off-the-shelf test system. It often includes software testing and diagnostic packages for specific tasks. For the general purpose system, the user selects the sources of stimuli, such as the function generator, etc., and measurement devices, such as voltmeters and digital analyzers. The maker of the test equipment then

assembles these subsystems and provides the required program control, interconnections and interfaces for the units, and perhaps will generate and debug the software.

Systems with dedicated controllers are becoming more popular compared to general purpose test systems. This is due to the fact that dedicated equipment optimizes testing, since their controllers are designed for a specific task. In practice, many of the dedicated systems are programmed simply by read-only memories. The minicomputer, however, may have to waste time in executing programs through software, and memory space could also be wasted. Where software is needed, it is the major cost in testing.

5.2 Evolution of Test Systems

5.2.1 First Generation Test System [Data Shuffling Test Systems]

Figure 5.2.1-1 shows a block diagram of a first generation test system. It contains 1) a CPU, 2) a main memory, 3) an interim buffer memory, and 4) test execution electronics. The interim buffer memory stores the test pattern and executes the test in a burst mode; the memory can be loaded from either CPU, main memory or a disc.

This system was adequate in testing random logic devices in the early 1970's, since at that time LSI devices were simple and took about 500 machine cycles to carry out the tests at a high confidence level. However, the system could not be used to test a complex LSI device due to the fact that the system did not have any intelligence in the interim buffer memory.

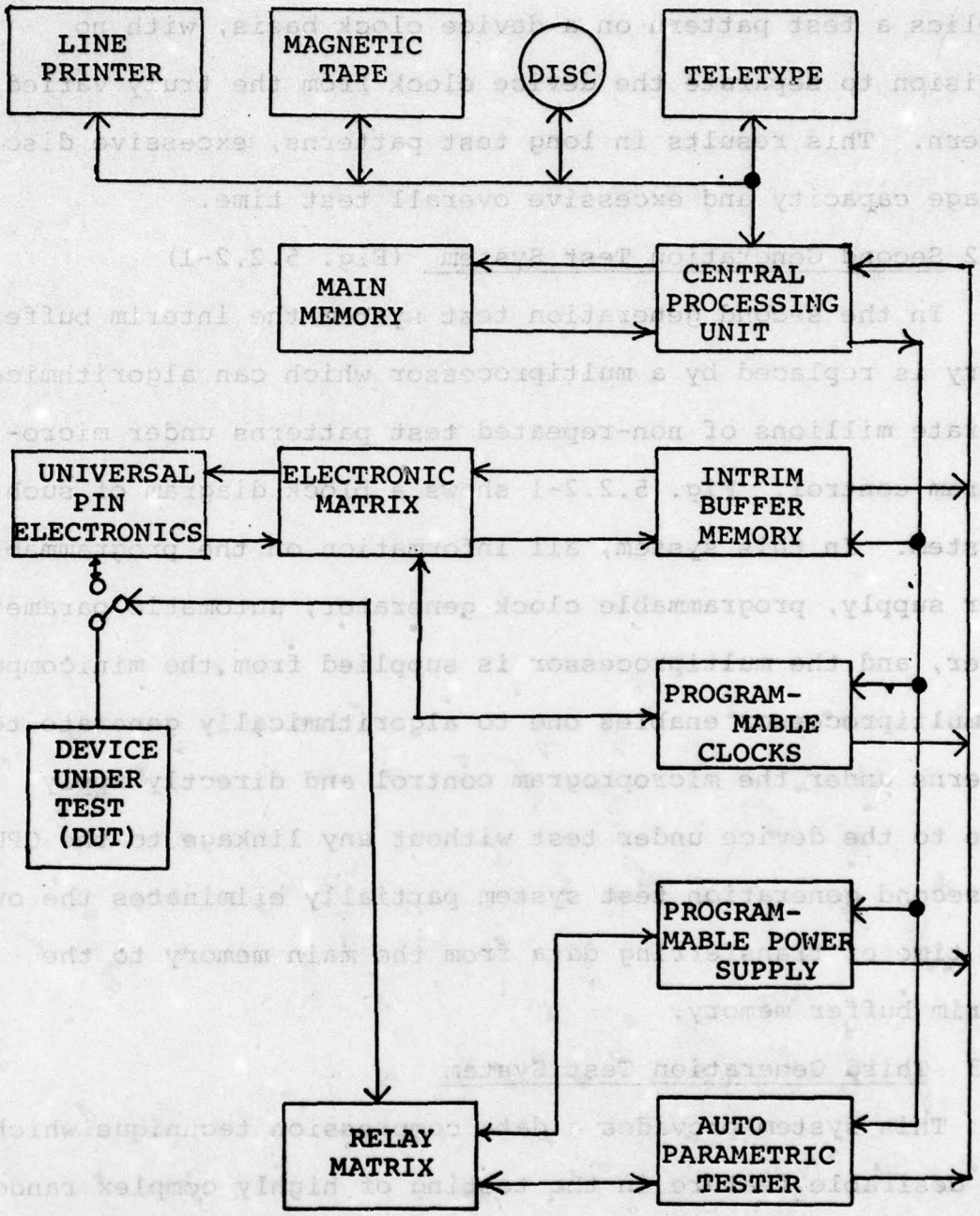


Fig. 5.2.1-1. First Generation Test System [Ref. 36]

It also required more than one information transfer from the disc to the interim buffer memory, and this increased the overhead time considerably. The interim buffer memory also

supplies a test pattern on a device clock basis, with no provision to separate the device clock from the truly varied pattern. This results in long test patterns, excessive disc storage capacity and excessive overall test time.

5.2.2 Second Generation Test System (Fig. 5.2.2-1)

In the second generation test system the interim buffer memory is replaced by a multiprocessor which can algorithmically generate millions of non-repeated test patterns under micro-program control. Fig. 5.2.2-1 shows a block diagram of such a system. In this system, all information on the programmable power supply, programmable clock generator; automatic parameter tester, and the multiprocessor is supplied from the minicomputer. The multiprocessor enables one to algorithmically generate test patterns under the microprogram control and directly apply these to the device under test without any linkage to the CPU. The second generation test system partially eliminates the over-head time of transferring data from the main memory to the interim buffer memory.

5.2.3 Third Generation Test System

This system provides a data compression technique which is a desirable feature in the testing of highly complex random logic devices such as communication chips. For example, millions of clock cycles are required to test communication chips of which only a few thousand cycles would have different input or output patterns. In this case it is advan ageous to store a few thousand active test patterns and not millions of

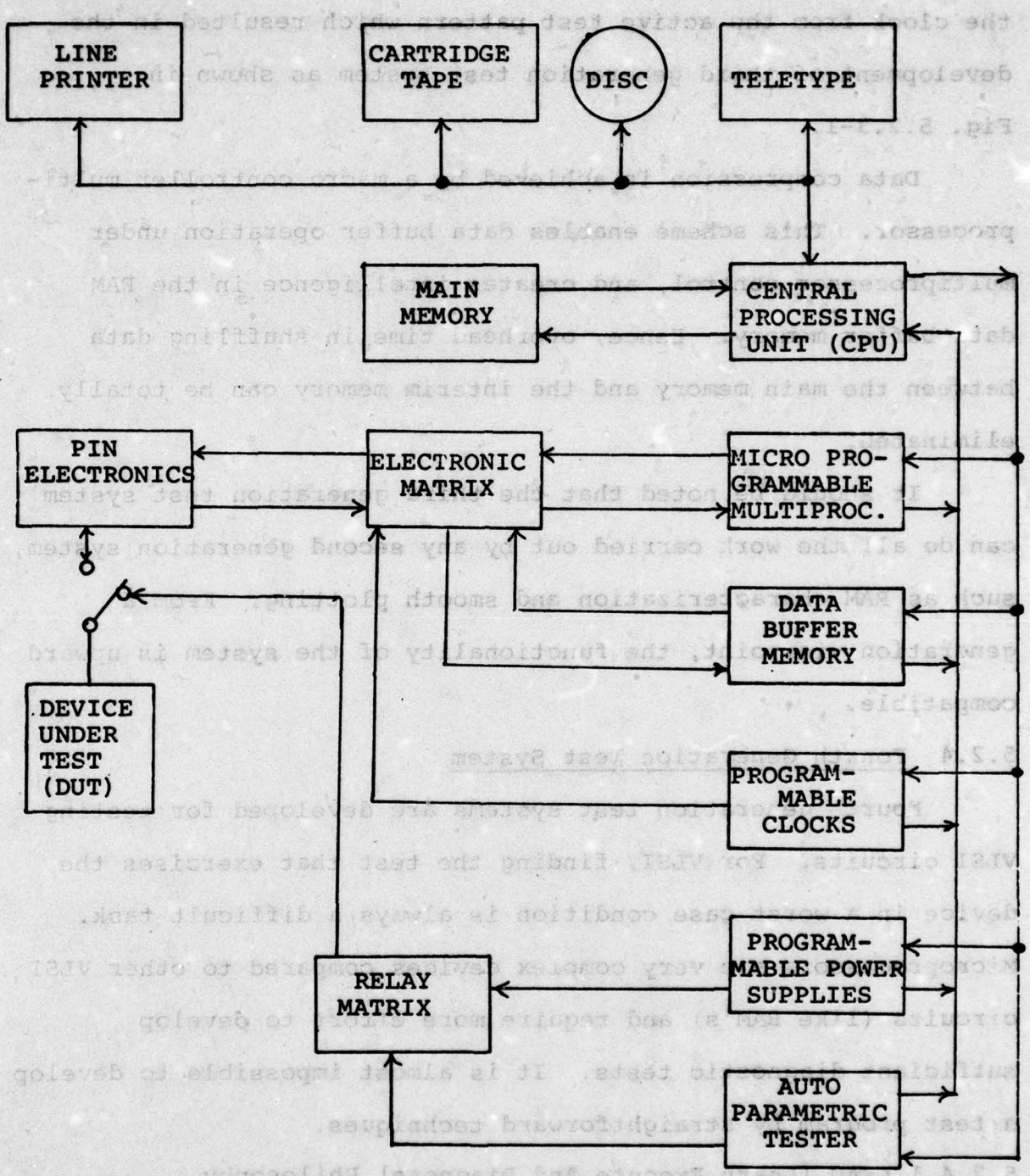


Fig. 5.2.2-1. Second Generation Test System [Ref. 36]

repetitive patterns. This requires a capability to separate the clock from the active test pattern which resulted in the development of third generation test system as shown in Fig. 5.2.3-1.

Data compression is achieved by a macro controller multiprocessor. This scheme enables data buffer operation under multiprocessor control, and creates intelligence in the RAM data buffer memory. Hence, overhead time in shuffling data between the main memory and the interim memory can be totally eliminated.

It should be noted that the third generation test system can do all the work carried out by any second generation system, such as RAM characterization and smooth plotting. From a generation viewpoint, the functionality of the system is upward compatible.

5.2.4 Fourth Generation Test System

Fourth generation test systems are developed for testing VLSI circuits. For VLSI, finding the test that exercises the device in a worst case condition is always a difficult task. Microprocessors are very complex devices compared to other VLSI circuits (like RAM's) and require more effort to develop sufficient diagnostic tests. It is almost impossible to develop a test program by straightforward techniques.

5.2.4.1 LEAD [Learn Execute And Diagnose] Philosophy

This philosophy is developed by Fairchild Systems Technology for microprocessor testing. A major complication to testing

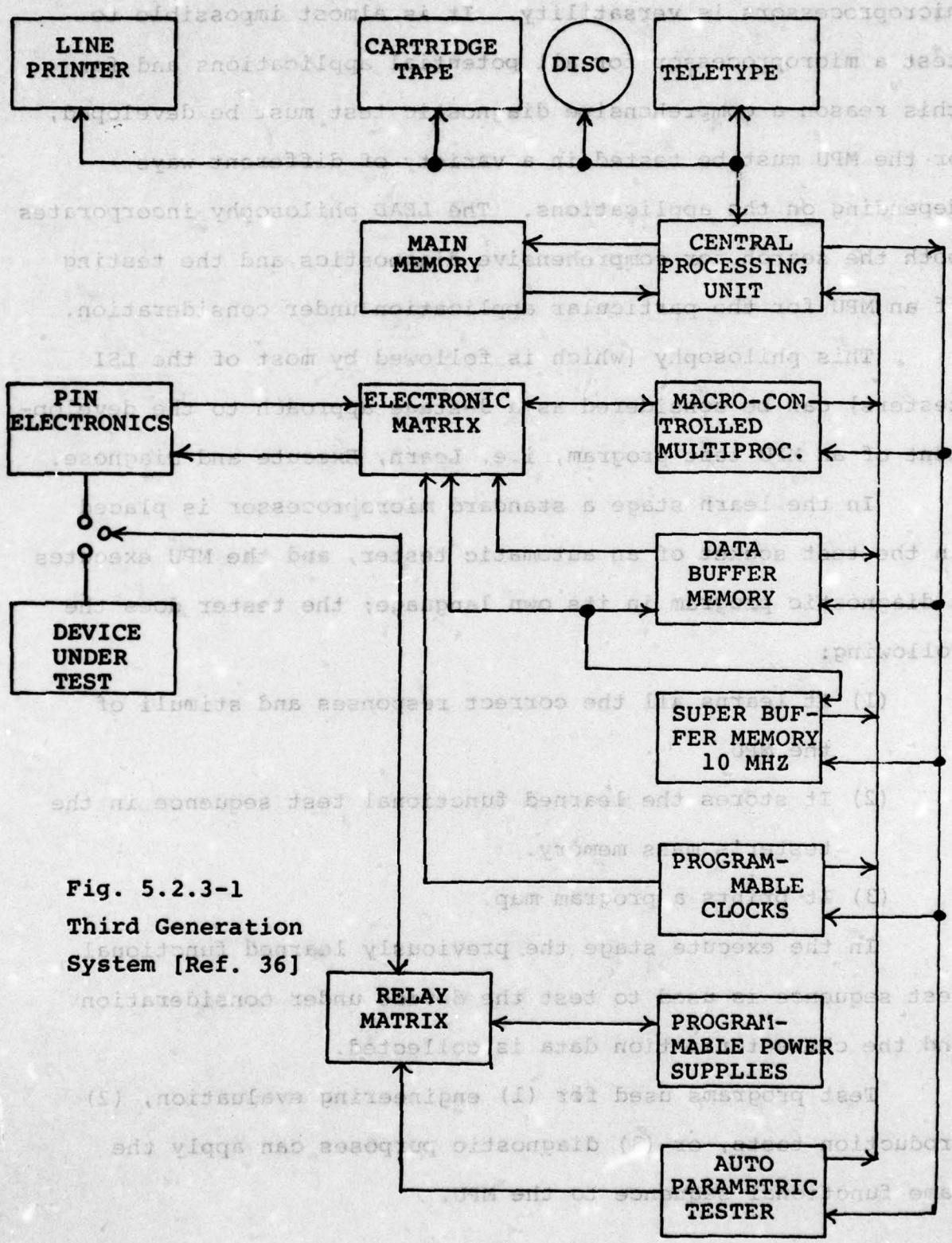


Fig. 5.2.3-1
Third Generation System [Ref. 36]

microprocessors is versatility. It is almost impossible to test a microprocessor for all potential applications and for this reason a comprehensive diagnostic test must be developed, or the MPU must be tested in a variety of different ways depending on the applications. The LEAD philosophy incorporates both the search for comprehensive diagnostics and the testing of an MPU for the particular application under consideration.

This philosophy [which is followed by most of the LSI testers] can be considered as a 3-stage approach to the development of an MPU test program, i.e. Learn, Execute and Diagnose.

In the learn stage a standard microprocessor is placed in the test socket of an automatic tester, and the MPU executes a diagnostic program in its own language; the tester does the following:

(1) It learns all the correct responses and stimuli of the MPU.

(2) It stores the learned functional test sequence in the tester's mass memory.

(3) It prints a program map.

In the execute stage the previously learned functional test sequence is used to test the device under consideration and the characterization data is collected.

Test programs used for (1) engineering evaluation, (2) production tests, or (3) diagnostic purposes can apply the same functional sequence to the MPU.

In the diagnose stage the characterization data is processed and compared with the expected data. Failed data can be traced to the actual instruction sequence with the help of the program map printed during stage 1.

5.2.4.1 Tester Requirements

The LEAD strategy requires a tester that completely simulates the natural environment of the MPU. This environment consists of (1) a total computing system, (2) peripheral devices, and (3) peripheral interfaces. Communication between the MPU and a tester is bidirectional for both data control functions. The tester should be capable of (1) changing tester pins from input to output and vice versa, (2) be able to simulate the microprocessor's memory and peripheral devices, and (3) be able to store all the activity of each pin of the microprocessor.

REFERENCES

General References

[1] M.A. Breuer, A.D. Friedman. Diagnosis and Reliable Design of Digital Systems, Computer Science Press, Inc., 1976.

[2] H. Troynagle, Jr., B.D. Carroll and J.D. Irwin. An Introduction to Computer Logic, Prentice-Hall, 1975.

[3] Digital Microcircuit Characterization and Specification, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, New York 13441, RADC-TR-77-91-Final Technical Report, March 1977, AD# A038 969.

[4] R.F. McPeak. The Development of Microprocessor Electrical Characterization Methods, Oct. 76, Contract Number JPL-954491.

Testing of Logic Circuits

[5] A.K. Susskind. Diagnostics for Logic Networks, IEEE Spectrum, Oct. 1973.

[6] W.G.W. Kreawels. Structural Testing of Digital Circuits, Philips Technical Review, Volume 35, No. 10, 1975, pp. 261-270.

[7] D.W. Bray. The ATVG Program: A Test Vecotr Generator for Sequential Networks, Workshop on Fault Detection and Diagnosis in Digital Systems, Lehigh University, 1971.

[8] P. Agrawal and V.D. Agrawal. Probabilistic Analysis of Random Test Generation Method for Irredundant Combinational Logic Networks, IEEE Transactions on Computers, Vol. C-24, No. 7, July 75, pp. 691-695.

[9] H.D. Schnurmann, E. Lindbloom and R.G. Carpenter. The Weighted Random Test Pattern Generator, IEEE Transactions on Computers, Vol. C-24, No. 7, July 1975, pp. 695-700.

Testing of Large Scale Integrated Circuits (LSI)

[10] E.R. Hnatek. Microprocessor Device Reliability, Microprocessors, Vol. 1, No. 5, June 1977, pp. 299-303.

[11] E.D. Colbourne, G.P. Coverleg and S.K. Behera. Reliability of MOS LSI Circuits, Proceedings of the IEEE, Vol. 62, No. 2, Feb. 1974, pp. 244-259.

[12] F.K. Justice and R.H. Lightkep. The Secret to Digital Circuit Testing, Sperry Technology (USA), 1974, V. 77, pp. 8-14.

[13] L.P. Henckels and R.M. Haas. Hardware or Software Simulation: A Comparison Between Two Techniques for Digital Testing, IEEE Proceedings, 1975, pp. 355-358.

[14] D. Izumi. The Challenge of Microprocessor Chip Testing, WESCON Technical Papers, Sept. 1975, pp. 1-4.

[15] J.F. Campbell. A New Real-Time Function Test Generation System for Complex LSI Testing, WESCON Technical Papers, Sept. 1975, pp. 1-4.

[16] G. Vaughn. Functional Testing of LSI Gate Arrays, 11th Design Automation Workshop Proceedings, 1974, pp. 258-265.

[17] J. Grason. Design Aids and Hardware Testing of Microprocessor System Circuit Packs, pp. 95-99.

[18] W. Barraclough, A.C. Chiang, and W. Sohl. Techniques for Testing the Microcomputer Family, Proceedings of the IFEE, Vol. 64, No. 6, June 1976.

[19] D. Hackmeister and C.L. Chiang. Microprocessor Test Technique Reveals Instruction Pattern Sensitivity, Computer Design, Dec. 1975, pp. 81-85.

[20] A.C.L. Chiang. Test Schemes for Microprocessor Chips, Computer Design, April 1975, pp. 87-92.

[21] A.C.L. Chiang. Two New Approaches Simplify Testing of Microprocessors, Electronics, Jan. 22, 1976, pp. 100-105.

[22] S.P. Morse. A Tutorial Paper on Software Approaches to Testing of Microprocessor Systems, 1975 Semiconductor Test Symposium No. 75CH1041-3C, pp. 53-58.

[23] D.H. Smith. Microprocessor Testing - Method or Madness? IEEE 1976 Semiconductor Test Symposium, No. 76H1174-1C, pp. 27-29.

[24] R.H. Carlstead and R.E. Huston. Test Techniques for ECL Microprocessors, IEEE 1976 Semiconductor Test Symposium, No. 77CH1261-7C, pp. 32-37.

[25] R. Huston. Microprocessor Function Test Generation on Sentry 600, IEEE 1974 Semiconductor Test Symposium, No. 74CH0909-2C, pp. 216-238.

[26] M.R. Barber and A. Zacharias. Integrated Circuit Testing, Bell Laboratories Record, May 1977, pp. 124-134.

[27] L. Badagliacca, and R. Catterton. Combining Diagnosis and Emulation Yields Fast Fault Finding, Electronics, Nov. 10, 1977, pp. 107-110.

[28] B. Schusheim. A Flexible Approach to Microprocessor Testing, Computer Design, March 1976, pp. 67-72.

[29] R.A. Frohwerk. Signature Analysis: A New Digital Field Service Method, Hewlett-Packard Journal, May 1977, pp. 2-8.

[30] S.B. Akers. Binary Decision Diagrams, IEEE Transactions on Computers, Vol. C-27, No. 6, June 1978, pp. 509-516.

[31] J. Losq. Efficiency of Random Compact Testing, IEEE Transactions on Computers, Vol. C-27, No. 6, June 1978, pp. 516-525.

[32] S.B. Akers. Partitioning for Testability, Proceedings of the IEEE, 1976 International Symposium on Fault Tolerant Computer, pp. 121-128.

[33] S. Bisset. LSI Tester Gets Microprocessor to Generate Their Own Test Patterns, Electronics, May 25, 1978, pp. 141-145.

[34] B. Schusheim. LEAD: A Microprocessor Testing Tool, Fairchild Systems Technology, Application Note 49, Dec., 1975.

[35] T.J. Snethen, D.C. Jessep. The Circuit Failure Modelling Challenge for LSI Proceedings of the 1973 IEEE International Symposium on Circuit Theory, pp. 425-430.

[36] W.C.W. Mow and A.C.L. Chiang. Evolution of LSI Test Systems, Macrodata Corporation Report, pp. 359-362.

[37] R. McCaskill. Test Approaches for Four Bit Microprocessor Slices, IEEE 1976 Semiconductor Test Symposium, pp. 22-26.

[38] Intel 8080 Microcomputer Systems User's Manual.

[39] J.L. Hilburn and P.M. Julich. Microcomputers/Microprocessors Hardware, Software and Applications, Prentice-Hall Series in Automatic Computation, 1976.

[40] S.E. Grossman. Automatic Testing Pays Off for Electronic System Makers, Electronics, Sept. 1974, pp. 95-109.

☆U.S. GOVERNMENT PRINTING OFFICE: 1979-614-023/157

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.