

AD-A096 310

LEHIGH UNIV BETHLEHEM PA
TESTABILITY AND RELIABILITY OF LSI.(U)
JAN 81 A K SUSSKIND

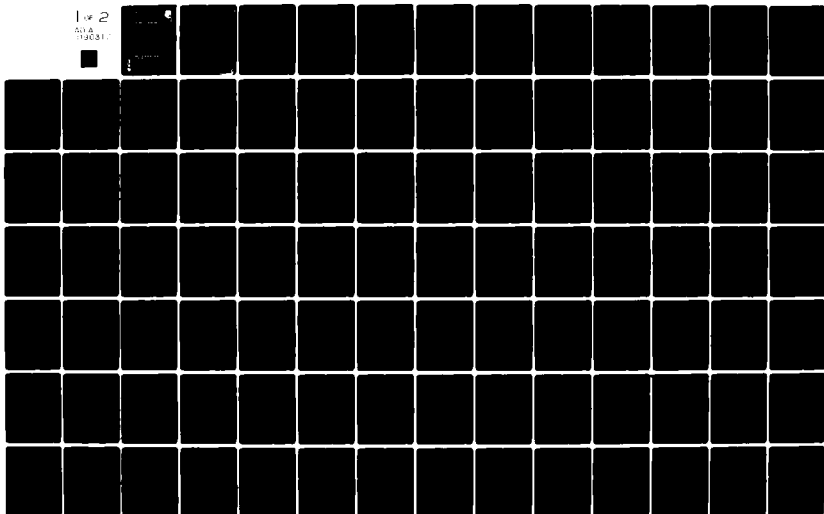
F/G 9/1

UNCLASSIFIED

RADC-TR-80-384

F30602-78-C-0292
NL

1 of 2
AD-A
110031



AD A 096310

RADC-TR-80-384
Final Technical Report
January 1981



TESTABILITY AND RELIABILITY OF LSI

Lehigh University

LEVEL II

Dr. Alfred K. Suskind

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

Laboratory Directors' Fund No. LD9408C1

DTIC
ELECTE
S **D**
MAR 13 1981
E

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

DDG FILE COPY

01 2 13 040

This report has been reviewed by the RADC Public Affairs Office (PA) 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-80-384 has been reviewed and is approved for publication.

APPROVED: *Mark W. Levi*

MARK W. LEVI
Project Engineer

APPROVED:

David C. Luke

DAVID C. LUKE, Colonel, USAF
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 (RBRP), 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)

17 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-80-384	2. GOVT ACCESSION NO. AD-A096310	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) TESTABILITY AND RELIABILITY OF LSI.	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report. 15 Sep 78 - 31 July 80	6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Dr. Alfred K. Susskind	8. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0292	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lehigh University Bethlehem PA 18015	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101 LD9408C1	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (RBRP) Griffiss AFB NY 13441	12. REPORT DATE January 1981	13. NUMBER OF PAGES 130
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same	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: Mark W. Levi (RBRP) This effort was totally funded by the Laboratory Directors' Fund.		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Testing Walsh Functions Built-In Test Testability Array Logic Reliability Logic Arrays Testability Measure PLA Exhaustive Testing BIT		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A number of studies related to testing, testability, and reliability were made and the major results are reported. One study was concerned with the relationship between testing and reliability, which lies in the lack of dependability caused by incomplete testing. An approach to calculating this lack of assurance, based on binary decision diagrams, was formulated and programmed. Initial experience with a first version indicates that care must be exercised in optimizing programming, if large networks are to		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

(over)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

205450

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

be handled economically.

A major part of the report is concerned with an exhaustive form of the 2^n different Walsh coefficients, of which two are particularly useful. The first is just the conventional counting of the number of ones in the response. The second is shown to detect any pin faults, in any combination, with a particularly simple test-equipment implementation that may well serve as a BIT (built-in-test) design.

The application of Walsh-coefficient verification to array logic is investigated. Simple PLA's are shown to be readily testable; arrays with flip-flops are also studied and methods for improving their testability are given.

The formulation of a testability measure (TM) is described. The TM is calculated on the basis of a gate-level network description. The actual calculations are based on approximations, but their validity can be verified by precise calculations. When the approximations are used, the task of evaluating the testability is much less than finding actual tests.

UNCLASSIFIED

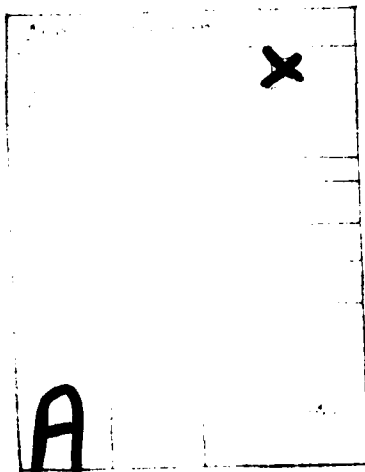
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

EVALUATION

The final report describes significant additions to the body of knowledge needed to define and measure the testability of LSI. The initial reach of the effort was toward a complete definition and measure of testability of real LSI and its relationship to reliability determination. Significant progress has been made toward that goal by creating the outlines for a new measure of testability of the logical representation of combinational logic. This must be used in conjunction with designs based on such structures as level sensitive scan design or its variations. A serendipitous result has been the discovery of a built-in-test method which is well adapted to inclusion in such designs and which combines great compaction of the test with a provable coverage of any number of input stuck-at faults or equivalent.

The effort used Laboratory Director Funding and was carried out in furtherance of the Device Reliability Research Program TPO.

Mark W. Levi
MARK W. LEVI
Project Engineer



SUMMARY

Testing of LSI is a challenging task which is currently not handled well enough. The difficulties arise primarily from the high logic density, which permits enormous complexity, and the severe access constraints due to pin limitations, which exclude the availability of a significant number of test points. In addition, currently modeling of faults cannot be done with high confidence. Furthermore, users of devices frequently have incomplete knowledge of the detailed make-up of a device or the make-up is so complex that a detailed description would not be useful.

To overcome these difficulties, design for testability has to be practiced and this work is based on that premise. The key to testability is a disciplined approach to design, guided by principles that are demonstrably cost effective from the testing point of view. Foremost among these is design which permits the conversion of sequential circuits to combinational circuits (CC's). This can be done by the scan-in/out, also called LSSD, design approach. We formulate the testing of CC's as the verification of Walsh coefficients. This type of testing does not hinge on fault modeling; only the assessment of its efficacy needs such a model. Its disadvantage is the required exhaustiveness, hence length, of the test. But the hardware required to execute it is very simple; so simple, in fact, that the required test generator and data storage can be incorporated into a VLSI chip, leading to a feasible form of BIT (built-in test).

From the point of view of the designer of the semiconductor chip, regular structures are highly desirable and therefore array logic has gained significant acceptance. We found that for simple PLA's (programmed logic arrays) exhaustive testing by verifying one or two Walsh coefficients is a rather thorough technique and conclude that simple PLA's are preferable over a more sophisticated array (the Associative Logic Matrix) with respect to testing by simple, and hence perhaps built-in, means.

For the case of arrays containing flip-flops, we consider testing by means of diagnosing sequences and indicate techniques for making these designs more readily testable than the ones currently proposed.

This report also describes a testability measure (TM) that can be applied to a gate-level description of a logic network. It is based on approximating well defined controllability and observability measures that reflect the ease with which tests can be found. The approximations make it feasible to execute the TM calculations with much less effort than would be required to find the actual test and so the TM should be helpful in appraising designs. The approach taken is such that the approximations can be compared with precise calculations.

We also describe a start in establishing the link between testing and reliability. This we do by calculating the number of those input patterns for which lack of testing has not assured the correct output. Our method is based on the use of binary decision diagrams, which makes it implementation independent. But the computational complexity still seems to be substantial and brief experience with a first program shows that care has to be exercised to allow complex logic networks to be handled at acceptable cost.

TABLE OF CONTENTS

	<u>PAGE</u>
SUMMARY	1
TABLE OF CONTENTS	3
1. TESTABILITY OF LSI AND ITS IMPACT ON RELIABILITY	5
1.1 Design for Testability	6
1.2 Testability Measure	8
1.3 Testability and Reliability	9
1.4 About this Report	13
2. TESTING BY VERIFYING WALSH COEFFICIENTS	17
2.1 Overview	17
2.2 Computation of Walsh Coefficients	22
2.3 Spectral Approach to Pin-Fault Testing	24
2.4 Use of other Walsh Coefficients	28
2.5 Lead-Fault Detection	29
2.6 Discussion	32
2.7 Tests for Shorts	33
3. TESTING OF LOGIC ARRAYS	40
3.1 Testing Simple PLA's	42
3.11 Crosspoint Defects	44
3.12 Stuck Lines	45
3.13 Shorts	46
3.14 More General Decoder Form and Application of C_0	48
3.15 Testing by Verifying C_{ALL}	52
3.2 Testing the Associative Logic Matrix	53
3.21 Effect of Other Faults in the ALM's	55
3.22 Discussion	67
3.3 Fault-Detection in Programmable Storage/ Logic Arrays	67
3.31 Effects of Faults	76
3.32 Change in the Number of States Due to Faults	79

	<u>Page</u>
3.0 TESTING OF LOGIC ARRAYS (continued)	
3.33 Modification of the Circuits	85
3.34 Testing Methods for Sequential Circuits	88
3.35 Application of Testing Methods to the FSM in the SLA	92
4.0 A TESTABILITY MEASURE	99
4.1 Formulation of the Testability Measure	100
4.2 The Basic Strategies	104
4.3 Controllability Calculations	106
4.4 Example of Controllability Calculations	109
4.5 Observability Calculation	110
4.6 Some Examples	115
4.7 Testability of Sequential Circuits	117

1. TESTABILITY OF LSI AND ITS IMPACT ON RELIABILITY

In light of today's state-of-the-art, we need not justify the basic premises on which the work reported on here is based:

1. LSI (large-scale integration) has already achieved a dominant role in the design of electronic equipment.
2. LSI will continue to offer more function per unit space, power, and cost.
3. Applications of LSI will continue to mushroom.
4. Because of the complexity of LSI building blocks, their testing is difficult. The main causes contributing to the difficulty are
 - a. The number of possible faults in a single unit is very large; a reasonable order of magnitude will soon be 10^5 .
 - b. Access to circuitry is severely limited by pin limitations.
 - c. Thorough tests require extensive test sequences. These are time consuming and require expensive test equipment. Thus careful testing is costly.
 - d. Users frequently lack information sufficient to permit thorough testing.
5. Although problems in testing LSI are now widely recognized, current market forces are such that many present-day commercial products are not "testable".

What do we mean by "testable"? Testability expresses the ease with which one can assure himself that the unit on hand operates correctly. Note that we did not refer to faults in our definition. This is so because we fear that not enough is known about faults in LSI to permit assurance that all faults are revealed by the popular testing schemes. In fact, as logic density increases, faults are becoming ever more complex: Pattern sensitivity, which has been recognized for a long time in such high-density parts such as ROM's and RAM's, will surely grow in significance; multiple faults are ever more likely; shorts--or phenomena akin to them--will be of greater concern; faulty manufacturing steps are increasingly likely. But much as we advocate that testability reflect

what we believe to be the true issue, namely the verification of functional integrity, we hasten to admit that this point of view is difficult to quantify, as we shall see later and also in Section 4 of this report.

Just as there is widespread agreement about the difficulty of testing LSI, so there is broad consensus in support of design for testability. By this is meant a collection of techniques that lead to implementations that are testable. We underline "collection" to place emphasis on the need to make use of a variety of techniques that together will achieve testability without penalizing performance and still leave the unit affordable. One must include here the various methods of self-test, be they via built-in test apparatus, or via the classical redundancy techniques exemplified by coding techniques. It is now clear that we have "sitting on the shelf" a great deal of know-how for effective utilization of redundancy; LSI and further advances in micro-electronics can make its application not only feasible, but also essential for real-time application. In the work reported on here, we did not pursue the direction of redundancy, but one would be remiss if he failed to keep in mind the important contributions that redundancy can make to testability.

1.1 Design for Testability

Having excluded (purely for the sake of limiting the scope of our work) further consideration of redundancy, we have looked at testability as broadly as possible. We recognize the value of already existing design guide lines for achieving testability, such as are given in Ref. [1.1], and feel that because of space limitations it would not be helpful to enter here into a discussion of those issues and remedies that [1.1] and others deal with and carefully document (see the review articles, Refs. 1.2 and 1.3). It has long been recognized that no other factor has so large an impact on testability as the extent to which flip-flops are used and how they are interconnected. Certainly the entire literature reflects that, and we emphasize that one of the most

potent tools for achieving a testable design is the scan-in/out scheme, also called LSSD (see Refs. 1.4 and 1.5). The central feature of LSSD is the means for eliminating the feedback when in the test mode, and surely it is the single most powerful measure that a designer could employ to make his unit testable.

Another very powerful technique for achieving testability is partitioning logic into blocks of manageable size while in the test mode. For good reasons it is often called "divide and conquer". Of course, this too, is not a new concept, but it also has not gained sufficient application in commercial products. Both scan-in/out and partitioning are now used--but by organizations other than the chip suppliers. The designer who must work with commercial building blocks is, at this time, unable to implement either scan-in/out or partitioning in the test mode, and yet their benefits are widely recognized. It has been the captive (in-house) IC facilities that have actually implemented them; the major semiconductor houses have so far failed to make them available.

We expect that the availability of scan-in/out and partitioning will become much wider. There appear to be no alternatives that offer comparable benefits. We see the present situation analogous to that in software, where when more complex programs had to be written, there emerged "structured programming". And just as structured programming was made feasible by the advent of low-cost, high-performance semiconductor devices, so structured design will be feasible. Indeed, it will not only be feasible, but essential. To be sure, it entails some sacrifice of real-estate on the chip; not all of the gates and connections will be needed in operation, but they will be essential to the achievement of dependable electronics. We feel confident that the price for giving up 10 to 20% of the real estate will be acceptable, particularly when weighed against the alternative, which is bleak for those who require high reliability.

The third major method for aiding testability is to adhere as much as possible to regular structures and to make repeated use of the same

regular structure. This has other major design benefits in addition to aiding testability: circuit design is expedited and reduced in cost; mask design is similarly simplified; more extensive simulation becomes feasible; and turn-around time is reduced. The benefits of repeated use of regular structures are also widely recognized [1.6, 1.7] and array logic has been developed to implement this approach. From the point of view of testing, array logic offers the advantages of simplicity and ease of understanding. There is nothing so manageable as two-level logic (e.g., NOR/NOR) and the arrays that have been described in the literature or modest variants of it and are certainly not difficult to tackle for testing. But as we shall see in Section 3, not all arrays are equal from the point of view of testing.

In summary, at this time the top three means for achieving testability are

1. handling, or rather elimination, of feedback;
2. breaking up complex entities into manageable units;
3. repeated use of simple, powerful, but straightforward arrays.

1.2 Testability Measure

One of the major goals of this project has been the measurement of testability. We made more than one effort in that direction. At first, we thought one would find it useful to differentiate between the testability of a function and the testability of the realization at hand. But for good reasons we later found that this approach was not fruitful. The approach that we finally developed, which is fully described in Section 4, is essentially a measure of the ease with which a lead can be tested. We have had satisfactory experience with it, but it is still being experimented with (by a graduate student who will report on his experience at a later time.)

As our ideas developed, we found that the measure may well have good use in test generation, where those leads that score low in observability and controllability (or maybe just one of these) are prime

candidates for algorithmic test generation based on such schemes as the D-algorithm or one of its equivalents. We mean that when a lead scores low, it is probably very efficient to pick that lead as the "site" of a fault, i.e., to begin with the D cube of a fault on that line. Alternatively, we could algebraically find a test by (1) "provoking" a fault on that line and (2) "propagating" it to the output.

Of course, our testability measure, because it shows "where the shoe hurts", serves well as a means for pinpointing leads to which outside connections would be particularly useful.

The last part of Section 4 shows our approach for measuring the testability of sequential circuits. We have chosen an iterative-circuit scheme, but are not yet in a position to evaluate its potential.

We must emphasize that the notion of testability has meaning only in light of some assumptions of what is good and what is bad. If one accepts a method of testing that is essentially exhaustive, then all circuits with the same number of pins are equally testable. We believe that there will be many applications where exhaustive tests are not only feasible, but highly desirable. For one thing, they no longer require a fault model such as "stuck-at". We have developed such an exhaustive test scheme and describe it fully in Section 2. In conjunction with LSSD, we expect it to be a useful test method.

1.3 Testability and Reliability

Certainly testing, or rather lack of testing, can be considered a factor in establishing the reliability of a device or larger assembly. When looked at from this point of view, one might consider testing as establishing the dependability of the unit under consideration, and we have taken this point of view in approaching the connection between testing and reliability. When a device (or unit) is fully tested, then its input/output is fully verified, and we can fully depend on its

output. If the device is not fully tested, however, then there will be circumstances under which the proper input/output behavior is not assured. The fraction of such unverified cases is the measure of unreliability contributed by incomplete testing.

How shall dependability be computed? One way would be based on the fault coverage of a test, or rather test sequence. This is a number that is often computed in test generation and is derived by a simulator program that is used to score a proposed test sequence. In those cases where the elaborate computer aids involved in scoring tests are available, it may well be that fault coverage is a good means for assessing dependability. One might question some of the limitations of this procedure. Simulators usually assume that only single faults of the stuck-at type can occur. But we must recognize that any kind of dependability assessment is bound to involve some assumptions about the nature of what could go wrong, and in our work to be described below we also had to make some assumptions about the nature of possible malfunctions. A second difficulty with using conventional fault coverage for measuring dependability comes from the fact that it is not easy to translate the coverage into a quantitative measure related to input/output behavior. Not only does the simulator score fail to separate undetectable--and hence harmless--faults from those that do affect circuit behavior, but we know of no way, other than through solving complex Boolean equations, for quantitatively appraising the effect of faults over all possible input conditions. The effort to solve Boolean equations is substantial, and so we searched for an alternative approach.

Because our work is addressed to very large-scale integration, we judged it appropriate to consider situations where detailed gate models are not available or too complex to consider. We found that a promising tool for the analysis of switching circuits for this kind of situation is the binary decision diagram, BDD, first proposed by S. B. Akers [1.8], [1.9]. BDD's are a promising tool for handling large Boolean functions without taking resort to complex algebraic manipulations. One of their

features is that they can be used to perform functional rather than implementation-dependent analysis and, because they take the form of binary decision trees, BDD's lend themselves to efficient computer manipulations. These take the form of path tracing in binary trees, and standard algorithms are available in the literature for that purpose; see [1.10].

Our work with BDD's formed only a small part of our effort and at this writing we have only preliminary results that indicate that the approach based on BDD's does offer promise. Because the study of the dependability due to incomplete testing is not complete, we do not include a major, detailed section on it, but summarize the work in the following few paragraphs.

A program was written to answer the following two questions: Given the BDD of a combinatorial function (a) how many of the input combinations result in a false output due to a given fault pattern? (b) What are the tests for the given fault pattern? The assumed fault patterns were not only stuck-at faults in any multiplicity, but also bridging (short) faults which can be treated by the "dominant low (high)" model (under this model, if there is a short between two leads, then whenever one is low (high), so is the other).

The program was written for the PASCAL-20 compiler running on Lehigh University's DECsystem-20. The memory image of the program takes 14,800 36-bit words of main memory. Each node in the graph uses six words of memory, so that a full ten-variable tree requires 6,150 words of storage. In order to accommodate large BDD's, the program was structured to minimize memory requirements. This results in greater running time than necessary and, as is so often the case, other trade-offs between running time and memory requirements could have been made.

As test cases, particularly unfavorable trees and fault patterns were chosen (these may well be worst cases). For these cases, the data below list, with number of variables (independent inputs) as the parameter, the CPU time consumed for (a) calculation of the number of input combinations for which the output is incorrect (this is column A) and

(b) that calculation plus finding the tests that detect the given fault pattern (this is column B).

independent variables	Number of nodes	CPU time (in secs) consumed	
		A	B
1	3	0.06	0.13
2	5	0.06	0.12
3	9	0.06	0.15
4	17	0.07	0.17
5	33	0.08	0.23
6	65	0.12	0.36
7	129	0.17	0.68
8	257	0.3	1.38
9	513	0.54	2.99
10	1025	1.09	6.59
11	2049	2.12	14.46
12	4097	4.35	33.50

Several conclusions can be drawn from this table:

1. For more than seven independent variables, the running time grows exponentially; it doubles with every added input. Hence large BDD's must be avoided.

2. The calculation of dependability (column A) is significantly faster than finding tests. (If an algebraic approach to finding dependability were used, this would probably not be so.)

3. The data listed is for just one fault pattern. In actual application, a number of such runs (one or more per input lead) would be made. An effort should therefore be made to reduce the running time. Hence the tracing algorithms used should be made more efficient. (In the program used here, for example, no provision was made to avoid tracing a given subgraph more than once; avoidance of this repetition would have required additional memory.)

It seems appropriate here to point out that BDD's need not be limited to cases where the details of the implementation are omitted. We have been able to show that (via a subscripted algebraic notation) it is possible to represent a gate model of a combinational circuit by a BDD. But we fear that this results in such great proliferation of the nodes that practical applications will be gravely limited.

1.4 About this Report

Section 2 presents a novel testing scheme. We give the theory on which it is based (the Walsh-Rademacher transform for discrete functions) and show its implementation and its efficacy against stuck-at faults. We do recognize, as we stated before, that the stuck-at fault model is not necessarily appropriate in LSI, but we are forced to use it by default, i.e., it is the only analytically tractable fault model available and was therefore used to permit generalized statements.

In Section 3 we apply testing by verification of Walsh functions to array logic. Three types of arrays are investigated. The last of these, and the most complex of them, is particularly tailored to the realization of sequential functions. We treat its testing from the point of view of diagnosing sequences and recommend specific ways for improving its testability through design modifications.

The testability measure is developed in Section 4. We illustrate its application by means of simple examples and describe its extension to sequential circuits.

Each of the topics discussed requires substantial background. Because of time limitations, we were not able to make each section self-sufficient and would therefore expect that some readers will have to take recourse to additional readings. For that purpose, we have provided a number of references. These were selected with the aim of listing the most helpful articles and books, and no attempt was made to be exhaustive.

Contributions to the work reported on here were made by several graduate students. The work in Section 3 was carried out by Suk I. Yoo, and the novel ideas in Section 3.3 are entirely his. Stephen L. Kessler worked out some of the examples given in Section 4 and served as a valuable sounding board for the testability-measure concepts as they were developed. The analysis of binary decision graphs was programmed by J. Patrick McHugh and some of his results are cited in this report.

References

- 1.1 W. M. Consolla and F. G. Danner "An Objective Printed Circuit Board Testability Design Guide and Rating System," Rome Air Development Center, Griffiss Air Force Base, N.Y. 13441, RADC-TR-79-327, January, 1980. AD# B051738L.
- 1.2 T. W. Williams and K. P. Parker "Testing Logic Networks and Designing for Testability," COMPUTER, Oct. 1979, pp. 9-21.
- 1.3 J. P. Hayes and E. J. McCluskey "Testability Considerations in Microprocessor-Based Design," COMPUTER, March 1980, pp. 17-26.
- 1.4 E. B. Eichelberger and T. W. Williams "A Logic Design Structure for LSI Testability," Proc. 14th Design Automation Conf., June 1977, IEEE Publication T7CM1216-1C, pp. 462-468. See also other related papers given at the same session.
- 1.5 B. Koenemann, J. Much, G. Zwiehoff "Built-In Logic Block Observation Techniques," Digest of Papers, 1979 Test Conference, Cherry Hill, N.J., IEEE Publication 79CH1509-9c, pp. 37-41.
- 1.6 Special Section on Programmable Logic Arrays, IEEE Trans. on Comp., Sept. 1979, C-28,9, pp. 593-627.
- 1.7 C. Mead, L. Conway Introduction to VLSI Systems, Addison Wesley, Reading, MA, 1980. Particularly Section 3.10.
- 1.8 S. B. Akers, "Binary Decision Diagrams," IEEE Trans. on Elect. Comp., C-27, 6, 1978, pp. 509-516.

- 1.9 S. B. Akers, "Functional Testing with Binary Decision Diagrams,"
Digest of Papers, 8th Annual International Conf. on Fault-Tolerant
Computing, Toulouse, France, June 1978, IEEE Publication
T8CM1286-4C, pp. 75-82.
- 1.10 D. E. Knuth The Art of Computer Programming, Addison-Wesley,
Reading, MA 1973.

2. TESTING BY VERIFYING WALSH COEFFICIENTS

2.1 Overview

The testing scheme investigated here works as follows: Refer to Fig. 2.1. The n-input combinational circuit under test is driven by an n-bit counter that

- a. generates all 2^n distinct patterns, each exactly once;
- b. generates a parity signal p with value 0(1) if the present pattern of counter bits or a selected portion has an odd (even) number of zeros;
- c. signals the beginning and end of the pattern generation process.

The response z of the unit under test (UUT) is scored by a reversible counter that, when in the run mode, operates in accordance with the following table:

p	z	ARITHMETIC OPERATION
0	0	+1
0	1	-1
1	0	-1
1	1	+1

For each new number generated by the driving counter, one such arithmetic operation is performed by the reversible counter (RC). Prior to the start of the test, the driving counter causes the RC to be preset to zero and after the last (2^n -th) number has been generated by the driving counter the contents of the RC are examined by the comparison circuit. In the case where p is derived from all of the bits in the driving counter, if the RC has contents of zero (00...0), the circuit under test is faulty; otherwise the circuit is declared good. We shall call this the " C_{ALL} test". In the case where p is derived from a proper subset of the bits in the driving counter, the final contents of the RC are compared with a stored number, N. If the contents of the RC differ

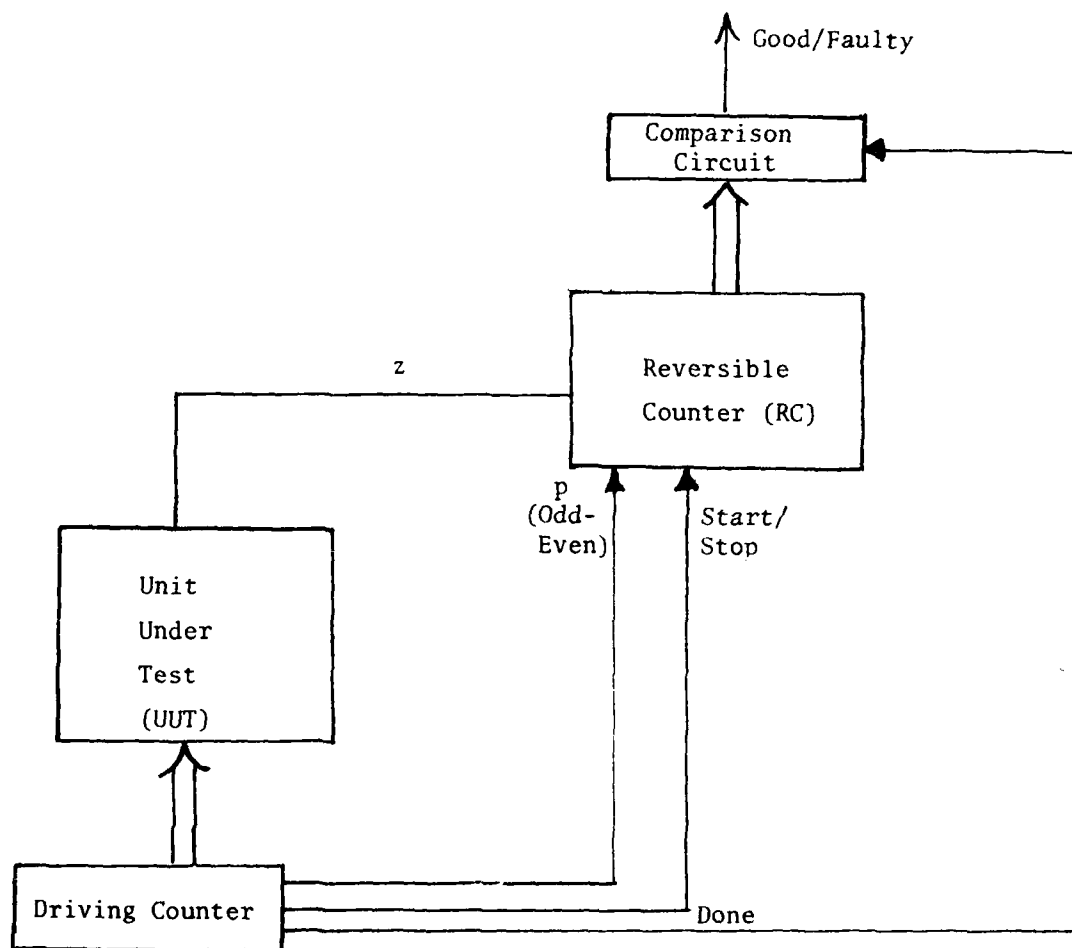


Fig. 2.1 Tester for Verifying Walsh Coefficients

from N, the UUT is declared faulty.

When p is determined by all of the bits in the driving counter, i.e., when a C_{ALL} test is made, our method of testing will be shown to detect all input stuck-at-constant-value faults (often called pin faults), in all possible combinations, singly or multiply. Moreover, since many of the possible faults in the interior of the circuit under test are equivalent to pin faults, this method will also detect these.

We illustrate the C_{ALL} test by means of the simple example in Table 2.1, where the fault-free circuit is a "majority" (also called "2-out-of-3" or "voter") circuit, and the faulty unit has input c stuck-at-1.

OUTPUT OF DRIVING COUNTER	P	RESPONSE OF FAULT- FREE UUT	CONTENTS OF RC	RESPONSE OF FAULTY UUT	CONTENTS OF RC
a b c					
0 0 0	0	0	+1	0	+1
0 0 1	1	0	0	0	0
0 1 0	1	0	-1	1	+1
0 1 1	0	1	-2	1	0
1 0 0	1	0	-3	1	+1
1 0 1	0	1	-4	1	0
1 1 0	0	1	-5	1	-1
1 1 1	1	1	-4	1	0

Table 2.1 Example of Test Operation

The C_{ALL} test can be used on any combinational circuit, except that a circuit which when fault-free would end with the RC having a final value of zero must have its design somewhat modified. A particularly simple modification consists of the addition of a single new input q and an AND-circuit such that it realizes the modified function F^m , defined as

$$F^m(x_1, x_2, \dots, x_n, q) = F(x_1, x_2, \dots, x_n)\bar{q} + x_1^*x_2^*\dots x_n^*q$$

Here F is the n -variable function that the original circuit was intended to realize and x_i^* denotes either x_i or $\bar{x}_i$. When $q = 0$, $F^m = F$, and this is the mode in which the circuit is used in its application. In testing, however, the new terminal q is made to take on both the value 0 and the value 1. Thus the driving counter has $n + 1$ stages for testing an n -input modified logic block.

For internal stuck-at lead-fault detection, a scheme similar to that illustrated in Fig. 2.1 is used, and the detection of these faults can be made concurrently with the detection of pin faults. The same driving counter is used; a second RC is added, and its odd-even terminal is controlled by a second p -signal that is derived from a proper subset of the bits in the driving counter. A particularly interesting case is where p is set to the constant 1. A more complete treatment of lead-fault detection is given in Section 2.5, and the detection of shorts is discussed in Section 2.7.

The virtues of our method of testing are:

1. The equipment needed for testing is very simple, as is evident from Fig. 2.1.
2. The data storage and test-program requirements are so small that they can be economically hardwired, as is also evident from Fig. 2.1.
3. Test preparation requires negligible effort. For pin-fault detection, one merely needs to determine in the part-design process if F , the object function, needs to be modified into F^m as specified above. The cost of the modification, if needed, is small. For lead-fault detection, a simple calculation determines the final value in the RC to be monitored.
4. Because of the above three features, all the test equipment can be built into an LSI chip at modest cost. Thus the technique offers a feasible means for BIT (built-in-test).
5. In the case of pin-fault detection, the validity of the test procedure is shown to depend only on the function mechanized. (In the case of internal leads, it also depends on the structure

of the internal fan-out.) Thus our method is substantially independent of the detailed gate implementation and device technology used.

6. All networks with an equal number of input pins become equally testable.
7. The driving counter can be shared by a multiplicity of test set-ups. Its length is determined by the circuit to be tested with the largest number of inputs.
8. The sequence in which the 2^n distinct input patterns are generated is irrelevant, so that a variety of counter types (polynomial binary, Gray code, etc.) is applicable. Even linear feedback shift-registers with maximum cycle length are applicable, providing provision is made to generate the all-zero (00...0) pattern. This may be advantageous in BIT implementations where the LSSD (also called scan-in/scan-out) design approach is used, because then shift registers are already available.

The weaknesses of our method of testing are:

1. The test duration limits the size of the network to be tested to somewhere between 20 and 25 input pins. A 20-pin network requires about 10^6 tests, which at 1 MHz would take 1 sec. This limitation is, however, not severe.
2. To achieve complete and assured lead-fault detection, the designer is somewhat constrained with respect to the internal network fan-out. This limitation is, however, not severe.
3. The method is limited to combinational logic and must therefore be combined with LSSD in practical applications.
4. The method is presently clumsy when fault location (rather than detection) is desired.

Our method of testing is based on the verification of so-called Walsh coefficients. That is, the test determines whether or not the function realized by the network on hand has the same partial "Walsh spectrum" as the nominal function.

Because Walsh spectra are not widely used in logic design or switching, a few words about them are in order. A Walsh spectrum of a function consists of a number of coefficients that can be made complete, so that the set of Walsh coefficients can be made to uniquely represent the given function. Thus the Walsh spectrum is similar to the Fourier spectrum, and both are examples of transformations. In the case of Walsh spectra, however, there is no physical interpretation similar to harmonics; the coefficients are merely numbers.

What little we need to know here about the theory of Walsh spectra is given in the next section. Those interested in a thorough mathematical treatment will find it in Ref. 2.1, while definitions in engineering terms are given in Ref. 2.2.

2.2 Computation of Walsh Coefficients

For the case of switching functions F of n variables, the Walsh coefficients are easily computed. First of all, the logical value 0 (1) is associated with the arithmetic value -1 (+1). Second, with F there are associated exactly 2^n points in the domain of F , which we call the "vertices of the function". Each coefficient is computed by multiplying the value (either +1 or -1) of the function at each vertex with the corresponding value (either +1 or -1) of the corresponding Walsh function, defined below, and then summing all the partial products over all 2^n vertices. Thus each coefficient has a value that lies in the range -2^n to $+2^n$.

The 2^n Walsh functions, W_i , are defined as follows: $W_0 = 1$; the remaining functions are formed from products of the (arithmetic) values of the independent variables. There are as many W_i , $i \neq 0$ as there are non-empty subsets of the set of all independent variables, i.e., a total of $2^n - 1$, and they are derived from all possible (arithmetic) products of the n variables: one at a time, two at a time, etc. For example, the functions W_2 and $W_{1,3}$ for $n = 3$ each have the eight values listed below, derived from x_2 and x_1 together with x_3 , respectively.

$x_1 x_2 x_3$	W_2	$W_{1,3}$	F	$W_2 F$	$W_{1,3} F$
0 0 0	-1	+1	0	+1	-1
0 0 1	-1	-1	0	+1	+1
0 1 0	+1	+1	0	-1	-1
0 1 1	+1	-1	1	+1	-1
1 0 0	-1	-1	0	+1	+1
1 0 1	-1	+1	1	-1	+1
1 1 0	+1	-1	1	+1	-1
1 1 1	+1	+1	1	+1	+1

Table 2.2 Examples of Walsh Functions and Walsh Coefficients.

In the above chart we have also entered the logic values of the majority function and indicated the values of the (arithmetic) products $W_2 \cdot F$ and $W_{1,3} \cdot F$ at each vertex. By performing the summations, we can obtain two of the eight Walsh coefficients of the majority function: C_2 is the sum of the entries in the second column from the right, hence $C_2 = 4$; C_{13} is the sum of the entries in the rightmost column. (That the W_i are a normal set similar to the exponential functions in Fourier transforms is illustrated in the example by summing $W_2 \cdot W_{1,3}$ over all vertices; the result is indeed zero.)

In some calculations, it is convenient to keep in mind that in a logical sum

$$F = T_1 + T_2 + \dots + T_m$$

that is disjoint, i.e., where no more than one term T_i can be true, a simple arithmetic sum represents C_0 of F :

$$C_0^F = C_0^1 + C_0^2 + \dots + C_0^M = \sum C_0^i$$

Here C_0^F (which is based on $W_0 = 1$ for all vertices) denotes the Walsh coefficient C_0 of F and C_0^i the Walsh coefficient C_0 of T_i .

Henceforth, we will call the Walsh coefficient C_0 the "first" coefficient and will denote the Walsh coefficient that is based on the Walsh function $W_{1,2,\dots,n}$, which is based on all the variables, by C_{ALL} .

We call attention to the fact that C_0 is a direct measure of the number of true vertices of the function. This is true because

$$C_0 = V_T - V_F$$

where $V_T(V_F)$ is the total number of vertices where $F = 1(0)$. But $V_T + V_F = 2^n$, so

$$C_0 = V_T + V_T - 2^n = 2V_T - 2^n$$

For the example majority function, $V_T = 4$, so $C_0 = 2 \times 4 - 8 = 0$. For the exclusive - OR function $F = a \oplus b$, $V_T = 2$ and so $C_0 = 2 \times 2 - 4 = 0$ also. In fact, it is easy to show by induction on the number of variables that $C_0 = 0$ for all parity functions $a \oplus b \oplus \dots \oplus z$.

In this report we shall not need to consider the inverse transformation, i.e., the process of finding F , given all of its Walsh coefficients. Suffice it to say that the inverse transformation can be performed and, like the process of finding the C_i given F , it is unique.

2.3 Spectral Approach to Pin-Fault Testing

Since testing is the verification of a function and each function has a unique Walsh spectrum, we can say that the role of fault detection is to determine whether or not the box on hand has the Walsh spectrum intended. As one might expect, unless some assumptions are made about the allowable deviations of a network from its nominal behavior, one would have to determine all of its C_i - all 2^n of them. (This is analogous to verifying that a network has the correct response at every input vertex--a frequently impractical approach.) We will

show that under the assumption of pin faults, however, the computation of the single Walsh coefficient C_{ALL} suffices for fault detection. This is stated more precisely in the following two theorems:

Theorem 2.1. Given a function $F(x_1, x_2, \dots, x_n)$ with $C_{ALL} \neq 0$. If in the implementation of F one or more of the x_i is stuck, the value of C_{ALL} of the implemented function becomes zero, no matter what the pattern of the stuck inputs is.

Theorem 2.2. Given a function $F(x_1, x_2, \dots, x_n)$ with $C_{ALL} = 0$. Then the function $F^m(x_1, x_2, \dots, x_n, q) = F + x_1^* x_2^* \dots x_n^* q$ has $C_{1,2,\dots,n,q}^m = C_{ALL}^m = \pm 2$, and if in the implementation of F^m any one or more of the x_i or q is stuck, the value of C_{ALL}^m of the implementation becomes zero, no matter what the pattern of the stuck inputs is.

Proof of Theorem 2.1

Consider the function F expressed as a sum of standard products in the usual Boolean notation:

$$F = a_0 \bar{x}_1 \bar{x}_2 \dots \bar{x}_{n-1} \bar{x}_n + a_1 \bar{x}_1 \bar{x}_2 \dots \bar{x}_{n-1} x_n + a_2 \bar{x}_1 \bar{x}_2 \dots x_{n-1} \bar{x}_n + a_3 \bar{x}_1 \bar{x}_2 \dots x_{n-1} x_n + \dots + a_{2^{n-1}} x_1 x_2 \dots x_{n-1} x_n \quad (2.1)$$

where each a_i has value 0 or 1 and is the entry in the corresponding row of the truth table. In this expansion call those products with an even (odd) number of barred variables "even products" ("odd products").

Now make the association of $\bar{x}_i$ with -1, x_i with +1, $a_i = 0$ with -1, and $a_i = 1$ with +1. If one then interprets the above Boolean equation in ordinary arithmetic, he has the right-hand side precisely in the form of the computation of the Walsh coefficient C_{ALL} . Equation (2.1) can be written as the difference between two arithmetic summations:

$$C_{ALL} = \sum_{\substack{\text{all } i \\ \text{associated} \\ \text{with even} \\ \text{products}}} a_i - \sum_{\substack{\text{all } j \\ \text{associated} \\ \text{with odd} \\ \text{products}}} a_j \quad (2.2)$$

To illustrate this notation, consider $n = 2$, so that

$$F = a_0 \bar{x}_1 \bar{x}_2 + a_1 \bar{x}_1 x_2 + a_2 x_1 \bar{x}_2 + a_3 x_1 x_2 \quad (\text{Boolean})$$

$$C_{ALL} = (a_0 + a_3) - (a_1 + a_2) \quad (\text{arithmetic})$$

Suppose now that in the realization of F an input x_i is stuck-in-1 (stuck-in-0). Then for every nominal product $m_k = x_1^* x_2^* \dots \bar{x}_i \dots x_n^*$ ($m_k = x_1^* x_2^* \dots x_i \dots x_n^*$) containing $\bar{x}_i (x_i)$ the associated coefficient in (2.1), a_k , is set equal to the coefficient a'_k of the product $m'_k = x_1^* x_2^* \dots x_i \dots x_n^* (m'_k = x_1^* x_2^* \dots \bar{x}_i \dots x_n^*)$ which differs from m_k only in the literal $\bar{x}_i (x_i)$. For example, if in a two-variable function the input x_2 is stuck-in-1, a_0 is replaced by a_1 and a_2 by a_3 , so that the faulty function F_{x_2-1} can be written in terms of the coefficients a_i of the fault-free function as follows:

$$F_{x_2-1} = a_1 \bar{x}_1 \bar{x}_2 + a_1 \bar{x}_1 x_2 + a_3 x_1 \bar{x}_2 + a_3 x_1 x_2$$

The value of C_{ALL} of F_{x_2-1} is $(a_1 + a_3) - (a_1 + a_3) = 0$. In general, with x_n stuck-in-1, a_0 is replaced by a_1 , a_2 by a_3 , a_4 by a_5 , a_6 by a_7 , etc. With x_{n-1} stuck-in-0, for another example, a_2 is replaced by a_0 , a_6 by a_4 , a_{10} by a_8 , etc. Observe that any one stuck-at fault replaces each of one half of all the coefficients by another coefficient that is unit-distant from it. Also, observe that the original coefficient and its replacement for the fault-free case are in different summations in (2.2). Therefore, after the replacement of every a_k by the corresponding a'_k , the two summations in (2.2) become identical, so that their difference is zero, and hence C_{ALL} in the presence of the fault has value zero. If more than one input is stuck, the replacement is even

more widespread. In the two-variable example, with x_1 -stuck-1 and x_2 -stuck-0, for example, a_0 and a_1 and a_3 are replaced by a_2 , so that again C_{ALL} takes on the value zero. In general, two stuck lines cause groups of four coefficients to become equal, three stuck lines cause groups of eight coefficients to become equal, etc. But no matter what the size of each group of coefficients that is made equal by the (possibly multiple) fault, in the fault-free case half of that group belong to the first summation in (2.2) and the other half to the second summation, so that C_{ALL} takes on the value zero, as claimed in the theorem.

Proof of Theorem 2.2

The expansion of F^m in standard products differs from that of F only in multiplication of each term in (2.1) by $\bar{q}$ and the addition of one term:

$$F^m = a_0 \bar{x}_1 \bar{x}_2 \dots \bar{x}_n \bar{q} + a_1 \bar{x}_1 \bar{x}_2 \dots x_n \bar{q} + \dots + a_{2^{n-1}} x_1 x_2 \dots x_n \bar{q} + x_1^* x_2^* \dots x_n^* q \quad (2.3)$$

Note that the last term has a coefficient of 1, and all the other products ending in q have coefficient 0 and are therefore omitted.

By definition

$$C_{1,2,\dots,n,q}^m = C_{ALL}^m = \pm 2 + \sum_{\substack{\text{all } i \\ \text{associated} \\ \text{with even} \\ \text{products in} \\ (2.3) \text{ and } \bar{q}}} a_i - \sum_{\substack{\text{all } j \\ \text{associated} \\ \text{with odd} \\ \text{products in} \\ (2.3) \text{ and } \bar{q}}} a_j \quad (2.4)$$

Suppose $x_1^* x_2^* \dots x_n^*$ is even (odd), i.e., the number of barred variables is even (odd). The term $+2(-2)$ accounts for all vertices where $q = 1$, as will now be shown. Among the vertices where $q = 1$, 2^{n-1} are even (odd) and at all but one of these the logic value of F is 0, so that these contribute $-(2^{n-1}-1) + 1 = -2^{n-1} + 2$ ($2^{n-1}-2$). At all of the

2^{n-1} odd (even) vertices where $q = 1$, $F = 0$, and so these contribute $(-2^{n-1}) (-1) = +2^{n-1}(-2^{n-1})$ to the Walsh coefficient.

The two summations in (2.4) are precisely those that make up C_{ALL} of F , the unmodified function, and by assumption they cancel one another. Thus $C_{1,2,\dots,n,q}$ has the value $+2$ (-2) for the fault-free case, as claimed.

Since modified function F^m has C_{ALL}^m which is nonzero, Theorem 2.1 applies to it and therefore the second claim is valid.

2.4 Use of other Walsh Coefficients

The method of fault detection that we have described makes use of only one Walsh coefficient, C_{ALL} , or its extension, C_{ALL}^m . What about the other Walsh coefficients?

The following theorem is not difficult to prove:

Theorem 3. If a realization of $F(x_1, x_2, \dots, x_n)$ has been subjected to pin faults on any subset of the lead set $L = \{i, j, \dots, k\}$, then the Walsh coefficient C_M of the function realized by the faulty network will have value zero if the set M equals or contains the set L .

From this, it follows that if for the nominal (fault-free) function $C_M \neq 0$, then the computation of C_M will end up with the value zero if the network on hand suffers from pin faults on any subset of the set L . If for the nominal function $C_M = 0$, then one can modify F to

$$F^m = F_q + x_1^* x_j^* \dots x_k^* q$$

This makes the nominal function have $C_M \neq 0$, and yet the faults in L will result in $C_M = 0$. In other words, Theorems 1 and 2 deal with the special case of $L = M = \{1, 2, \dots, n\}$.

But note that whereas a fault set will set C_M to zero, no claim is made that if $C_M = 0$, then a subset of the fault set L is present. In fact, we have found examples where $C_M = 0$ when the actual fault set

was not contained in L. Thus we have not been able to use the other Walsh coefficients for fault location, i.e., determination of which pins are faulty. We have, however, been able to apply the coefficient C_0 for the detection of interior faults, as is shown in the next section.

2.5 Lead-Fault Detection

We will make use of the term inversion parity, IP, of a path. By that we shall indicate whether the number of inverters along that path from the given starting point to the network output is even or odd. For example, in Figure 2.2, the IP starting at lead 2a is odd along both paths starting at 2a, whereas the IP starting at lead 2b is even. Hence, the IP starting at lead 2 is odd for some paths and even for others.

If the IP of every path starting at a lead L is the same (such as in the case of the two paths starting at lead 2a), then we say for short "L has unique IP", and then we can always write the network output in terms of the logic signal x_L on lead L in a sum-of-products form which contains x_L only in uncomplemented (complemented) form if the IP of L is even (odd).

Suppose x_L appears only in uncomplemented form; the function realized by the network is then said to be "positively unate in x_L ". In this case if lead L is stuck-at-1 (s-a-0), the result will be an increase (decrease) over the fault-free case in the number of input patterns for which the output takes on the value 1. On the other hand, if the function realized is negatively unate in x_L (it appears only in complemented form) and L is s-a-1 (s-a-0), the result will be a decrease (increase) in the number of vertices for which the output takes on the value 1. In either case, uniqueness of the IP of a non-redundant lead implies that a fault will surely change the total number of ones in the function realized by the network.

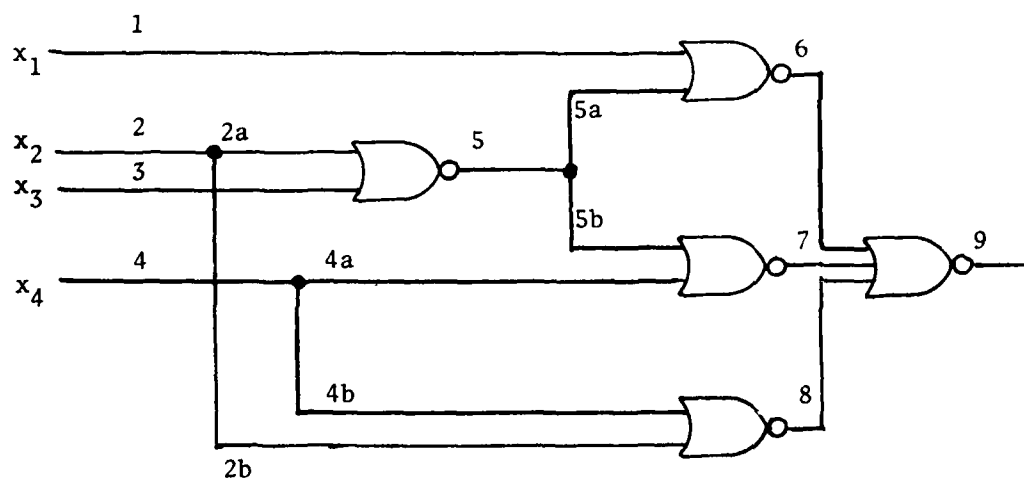


Fig. 2

We are now ready to prove the following:

Theorem 2.4. In a network form so restricted that after every initial fan-out point (in traversing from the independent inputs to the output) every lead has unique IP, any combination of stuck-at faults on all the leads up to the initial fan-outs can be detected by verifying C_{ALL} and any single stuck-at fault on leads following fan-out points can be detected by calculating C_0 .

To show the validity of the theorem, note that faults on leads up to the first fan-out points are equivalent to pin faults, so that C_{ALL} does indeed check for these as stated in Theorems 2.1 and 2.2. The rest follows from the previous discussion.

Theorem 2.4 should not be interpreted to imply that faults on leads before initial fan-out can be detected as well as simultaneously occurring single faults on leads following initial fan-out.

Theorem 2.4 gives us sufficient conditions for designing networks that can be fully tested for stuck-at faults by verifying only C_0 and C_{ALL} . The conditions do not appear severe. They are met, for example, by the three-level NAND networks discussed in Ref. 2.3. Moreover, every two-level network satisfies the conditions of Theorem 2.4.

Note that the implementation of checking for single faults following initial fan-out need not take the form of full-fledged computation of C_0 . This is so because $C_0 = V_T - V_0 = 2V_T - 2^n$, and hence it is sufficient merely to count the number of cases where the output has value 1 as the input takes on the 2^n different combinations.

Whether the computation is done according to the first form or the second is a matter of implementation convenience. If the determination of C_0 is to be made simultaneously with that of C_{ALL} , a separate response counter is needed, one for each coefficient. Then the counter determining C_0 need not be bidirectional if the second form is used (one simply counts 1's in the response), and is therefore simpler. On the other hand, if C_0 and C_{ALL} are to be determined sequentially by

running all 2^n input combinations twice, then the bidirectional counter used in determining C_{ALL} can be left unchanged for use in computing C_0 by setting $p = 1$ and calculating according to the first form.

2.6 Discussion

Our work has shown that two simple calculations--those of C_0 and C_{ALL} --suffice to do a rather good job of testing. Our proofs could claim only single-fault coverage for faults on most internal leads, but we know from our (limited) experience that many combinations of simultaneous internal faults are also detected. At this writing, the total fault coverage would have to be determined through simulation. This, of course, is a costly process and violates the spirit of this work, which aims to make testing--both its planning and execution--as simple as possible. To achieve this, we place constraints on the designer that are believed to be sufficiently lenient to make the application of our concepts practicable. Should it turn out that the demand for unique IP of leads past initial fan-out is too restrictive and either one or more additional output pins are available or built-in test (BIT) is to be implemented, then one can treat the lead with non-unique IP as an additional output and calculate its Walsh coefficients C_0 and C_{ALL} . This, of course, increases the number of computations required, and again serial or parallel determination of the set of Walsh coefficients can be used.

To enlarge the fault detection capability of our scheme, we suggest that when determining C_{ALL} we do not merely test the RC for the final contents of 00...0, but rather compare it with the value that the fault-free function has. Then the UUT will only pass if it has the correct value of C_{ALL} . Experience has shown that many additional faults will then be detected, and if we check both C_0 and C_{ALL} , we anticipate very good fault coverage. In fact, if we want to achieve testing for short-circuits, it will be useful to compare the actual value of C_{ALL} and C_0 with the nominal values, as will be shown in the next section.

2.7 Tests for Shorts

An important cause of faults in dense logic circuits are shorts and tests for detecting their presence cannot, in general, be found with the same ease as tests for stuck-at faults. In this section, we take a novel approach to the problem of test generation for shorts by exploring the short-detecting capability of Walsh coefficients. Our work is based on the following premises:

1. We start with no more than the functional description of the logic, not a gate model. Thus we will be dealing with "shorts between input pins."

2. When there is a short between a lead pair, the lead with the smaller signal dominates. In the positive logic convention, this means that a pair of leads where one is nominally in the zero state and the other in the one state will have both leads in the zero state when there is a short between them. We will briefly indicate later how to modify our analysis when the larger signal dominates.

Analytically, our premises lead to the following formulation. Assume that we wish to express the effect of a short between leads a and b. We begin by expanding the function mechanized by the logic block, f, around a and b:

$$f = \bar{a}\bar{b}R_0 + \bar{a}bR_1 + a\bar{b}R_2 + abR_3 \quad (2.5)$$

The R_i 's are called residues and are, in general, functions of the remaining variables. For example, we expand the majority function of five variables (a,b,c,d, and e), which is true when three or more of the inputs are true:

$$f = \bar{a}\bar{b}(cde) + \bar{a}b(cd+ce+de) + a\bar{b}(cd+ce+de) + ab(c+d+e)$$

Here $R_0 = cde$; $R_1 = R_2 = cd + ce + de$; $R_3 = c + d + e$.

Due to a short between a and b, when their signal levels are nominally different, the function realized will be the same as when

both are low. So (2.5) becomes, in the presence of the short between a and b:

$$f^{a,b} = \bar{a}\bar{b}R_0 + \bar{a}bR_0 + a\bar{b}R_3 + abR_3 \quad (2.6)$$

2.71 Verification of C_0

In accordance with Section 2.2, because our expansions in (2.1) and (2.2) are disjoint, we have

$$C_0 = C_0^0 + C_0^1 + C_0^2 + C_0^3 \quad (2.7)$$

where C_0^i is the first Walsh coefficient of R_i , C_0 is the first Walsh coefficient of function f , and the addition in (2.7) is arithmetic. Similarly, from (2.6)

$$C_0^{ab} = C_0^0 + C_0^0 + C_0^0 + C_0^3 = 3C_0^0 + C_0^3 \quad (2.8)$$

Verifying C_0 will test for a short unless $C_0 = C_0^{ab}$, and this occurs if

$$2C_0^0 = C_0^1 + C_0^2 \quad (2.9)$$

To illustrate, we return to the majority function, for which $C_0^0 = 2 - 2^3 = -6$; $C_0^1 = C_0^2 = 8 - 8 = 0$; $C_0^3 = 14 - 8 = -6$. We find that (2.9) is not met, and draw the conclusion that verifying C_0 of the majority network "3-out-of-5" will check for a short between leads a and b. Moreover, since the majority function is symmetric, it follows that verifying C_0 will detect any pair of input shorts. (Indeed, reference to Pascal's triangle shows that this result is not limited to five variables.)

But let this pleasing result not lead to overly optimistic expectations. Consider the parity function, $f = a \oplus b \oplus \dots \oplus z$. Its residues R_i are also parity functions (or complements of parity functions), which means that they each have as many true vertices as false ones. Consequently, for all residues R_i , $C_0^i = 0$ and so (2.9),

the condition of failure, is met. Thus verifying C_0 of parity functions cannot detect shorts and so it is wise to consider other Walsh coefficients.

2.72 Other Walsh Coefficients

We can generalize on the basis of (2.5) and (2.6) as follows. Let C_j^i denote Walsh coefficient C_j of R_i and $C_{a,b,j}$ the Walsh coefficient of the function on hand, where j denotes any subset of the set of all variables excluding a and b . Then, again because (2.5) is a disjoint decomposition, we have

$$C_{a,b,j} = C_j^0 - C_j^1 - C_j^2 + C_j^3 \quad (2.10)$$

where the signs denote ordinary arithmetic. Due to a short between a and b , we have the circuit behaving as in (2.6), which has the Walsh coefficient

$$C_{a,b,j}^{a,b} = C_j^0 - C_j^0 - C_j^0 + C_j^3 = C_j^3 - C_j^0 \quad (2.11)$$

Then verifying $C_{a,b,j}$ detects the short unless (2.10) equals (2.11), i.e.,

$$2C_j^0 = C_j^1 + C_j^2 \quad (2.12)$$

For the example of the parity function, we find for $j = \{c,d,\dots,z\}$, $C_j^0 = C_j^3 = -C_j^1 = -C_j^2$. Consequently, (2.12) is not met and verifying C_{ALL} of parity functions serves as a test for all pairs of pin shorts, because parity functions are symmetrical in the input variables.

An interesting property of Walsh coefficients can now be derived. From (2.5)

$$C_j = C_j^0 + C_j^1 + C_j^2 + C_j^3 \quad (2.13)$$

and from (2.6)

$$C_j^{a,b} = C_j^0 + C_j^0 + C_j^0 + C_j^3 = 3C_j^0 + C_j^3 \quad (2.14)$$

Verifying C_j will fail to detect the short between a and b if (2.13) equals (2.14), or

$$2C_j^0 = C_j^1 + C_j^2$$

But that is exactly (2.12) above, and so we conclude that:

Theorem 2.5. Test for a short between a and b based on $C_{a,b,j}$ has the same power as one based on C_j .

Theorem 2.5 leads one to conclude that C_0 and C_{ALL} are particularly powerful for detection of shorts, since they provide the same information about shorts between m and n as $C_{m,n}$ and $C_{ALL-\{m,n\}}$, respectively, for all pairs m and n. Keep in mind that $C_{m,n}$ ($C_{ALL-\{m,n\}}$) also provides information about shorts not between m and n; that information is not necessarily obtained by testing only C_0 (C_{ALL}).

The power of $C_{a,j}$, where j is any subset of the variables not including a and b, for detecting a short between a and b can be assessed as follows. From (2.5)

$$C_{a,j} = -C_j^0 - C_j^1 + C_j^2 + C_j^3$$

$$C_{b,j} = -C_j^0 + C_j^1 - C_j^2 + C_j^3$$

and from (2.6)

$$C_{a,j}^{a,b} = -C_j^0 - C_j^0 + C_j^0 + C_j^3$$

Coefficient $C_{a,j}$ will fail to reveal a short between a and b if $C_{a,j} = C_{a,j}^{a,b}$ or

$$C_j^1 = C_j^2 \quad (2.15)$$

Also from (2.6)

$$C_{b,j}^{a,b} = -C_j^0 + C_j^0 - C_j^0 + C_j^3$$

and so $C_{b,j}$ will fail to reveal a short between a and b under the same conditions as $C_{a,j}$.

The more Walsh coefficients one verifies, the more likely he is to detect a fault, no matter what that fault is. To what extent one will want to go beyond verifying C_0 and/or C_{ALL} will depend on (1) the particular short patterns he anticipates; after all, it is probably unlikely that any short pattern whatsoever can occur and (2) the particular function on hand. It should be clear that by writing appropriate expansions such as (2.5) and (2.6), it is straightforward to decide whether or not a given pattern of shorts will be detected by a given Walsh coefficient.

Finally, we want to emphasize that our initial assumption of "dominating zeros" can be easily changed to "dominating ones", i.e., where a pair of shorted leads has a one whenever one or more has a nominal one. For example, under the dominating ones assumption (2.6) becomes

$$f^{a,b} = \bar{a}\bar{b}R_0 + \bar{a}bR_3 + a\bar{b}R_3 + abR_3 \quad (2.6A)$$

and then the faulty network will yield the Walsh coefficient

$$C_{a,b,j}^{a,b} = C_j^0 - C_j^3 - C_j^3 + C_j^3 \quad (2.11A)$$

so that the condition for failure to detect the short between a and b becomes

$$2C_j^3 = C_j^1 + C_j^2 \quad (2.12A)$$

Similar changes can be made in the other results derived above.

2.73 Function Modification

As we showed before, a simple functional modification can make a given function "stuck-at Walsh-testable". Here we show by an elementary example that a simple functional change can also make a given function "short Walsh-testable".

Consider again the parity function, which we showed is not testable for shorts by verifying C_0 . Suppose we insist on using C_0 for testing purposes. If we then build the function of $n + 1$ variables

$$f' = (a \oplus b \oplus \dots \oplus z)\bar{q} + \bar{a}\bar{b} \dots \bar{z}q$$

and set q to 0 in the operational mode, verifying C_0 while letting q take on both values under test does in fact detect a short between any lead pair. This can be shown as follows:

$$\begin{aligned} C_0^0 &= f' \Big|_{a=b=0} = C_0 [(c \oplus d \oplus \dots \oplus z)\bar{q} + \bar{c}\bar{d} \dots \bar{z}q] \\ &= 2(2^{n-3} + 1) - 2^n \end{aligned}$$

$$\begin{aligned} C_0^1 &= f' \Big|_{a=0, b=1} = C_0 [(1 \oplus c \oplus d \oplus \dots \oplus z)\bar{q}] = f' \Big|_{a=1, b=0} = C_0^2 \\ &= 2(2^{n-3}) - 2^n \end{aligned}$$

Clearly condition (2.9) is not met and so C_0 for the modified function is a valid test for shorts between any lead pair. (The same simple modification also works under the assumption of dominating ones.)

Unlike the case of tests for stuck-at pins, there is no one simple modification that one can suggest; there are just too many possible sets of shorts ($2^n - 2$) and too many functions. In general, one will be able to go a long way by use of simple functions, such as AND (used in the above example) and OR.

References

- 2.1 K. G. Beauchamp, "Walsh Functions and Their Applications," Academic Press, 1975.
- 2.2 R. B. Lackey and D. Meltzer, "A Simplified Definition of Walsh Functions," IEEE Trans. on Computers, C-20, 2, February 1971, pp. 211-213.
- 2.3 K. K. Chakrabarti, A. K. Choudhury, and M. S. Basu, "Complementary Function Approach to the Synthesis of Three-Level NAND Network," IEEE Trans. Comp., Vol. C-19, 6, June 1970, pp. 509-514.

3. TESTING OF LOGIC ARRAYS

The work reported on in this section is an outgrowth of our studies summarized in the previous section, where we treated the problem of testing as equivalent to verifying Walsh coefficients. We began by investigating the extent to which the verification of a small number of Walsh coefficients results in good fault coverage in logic arrays.

It turns out that for ordinary (two-level) PLA's, verification of the coefficient C_0 (the one based on the Walsh function $W_0 = 1$, analogous to d-c in Fourier analysis) is a very powerful test and is likely to be a practical means for thoroughly and yet economically testing ordinary PLA's. Except for the detection of pin faults, C_0 is more powerful than checking C_{ALL} . As shown in Section 3.1, this is true particularly when faults other than classical stuck-at faults are considered, and in Section 3.1 we describe in some detail testing for shorts between lines as well as errors in "programming", i.e., missing and extra devices in the various arrays.

The analysis of a more complex logic array, the associative logic matrix (ALM), reveals that shorts in this array represent a particularly troublesome failure because it can result in sequential behavior. As we show in Section 3.2, while ordinary, two-level PLA's can be thoroughly tested by verifying C_0 and perhaps also verifying C_{ALL} , one cannot have the same high level of confidence when these tests are applied to an ALM. Our study shows that testability is substantially increased by the addition of an extra output and this is not surprising, since one can expect logic arrays to be particularly well-suited for making testing manageable by the incorporation of extra logic. But if shorts can be expected in ALM's, there can result feedback, hence sequential behavior, hence operation for which static tests, such as verification of Walsh coefficients, cannot serve as a sure means for fault detection.

In the third part of this section, we describe our analysis of the effects of faults on a third type of logic array, called Storage/Logic Array (SLA). Here we deal with a sequential circuit, and so testing by verification of Walsh coefficients is not applicable. We have again considered shorts and "programming" errors, as well as stuck-at faults. We have found that as originally proposed, SLA's could have faults that effectively increase the number of states in the array. This makes testing particularly difficult, and we show a design change that would eliminate the possibility of an increase in the number of states. It is then readily possible to test SLA's through the use of distinguishing sequences and verification of transitions, provided the nominal machine is strongly connected and has a single distinguishing sequence. We consider means for assuring that these conditions hold and suggest a simple design modification toward this end.

Readers who have not had the opportunity to study logic arrays will probably find it helpful to refer to the references listed at the end of this section and appropriately keyed in with the text in the following material.

3.1 Testing Simple PLA's

A simplified schematic of the basic PLA array is given in Fig. 3.1. We show n-p-n transistors which, under the positive-logic convention (parameter representing logical value is larger for logical 1 than for logical 0) and with the parallel, grounded emitter configuration shown mechanize the NOR function. Thus our configuration realizes the NOR-NOR-INVERT or NOR-OR logic form. Since NOR is the product of the inverted inputs (e.g., $\overline{A + B} = \bar{A}\bar{B}$), it follows that the array shown in Fig. 3.1 realizes functions in the form of a sum of products, where the products consist of the complements of the connected inputs. In particular, Fig. 3.1 shows the following functions

$$f_1 = x_1\bar{x}_3 + x_2\bar{x}_4$$

$$f_2 = x_2\bar{x}_4 + \bar{x}_1\bar{x}_2\bar{x}_3$$

$$f_3 = x_1\bar{x}_3 + x_2x_3x_4$$

The horizontal lines at the output of the first level of gating, which mechanize the individual products, are frequently called the word lines and we shall use that term here. The inputs x_i and x_j are sometimes called the decoder outputs because in some PLA applications these inputs are not single literals, but products of more than one variable (typically two). This has been shown to be advantageous in some applications and will be discussed later.

We show below that the basic PLA can be tested by checking the value of C_0 at each output. This method of testing allows us to detect not only the commonly assumed single stuck-at faults, but also a variety of multiple faults, errors in programming, and even shorts.

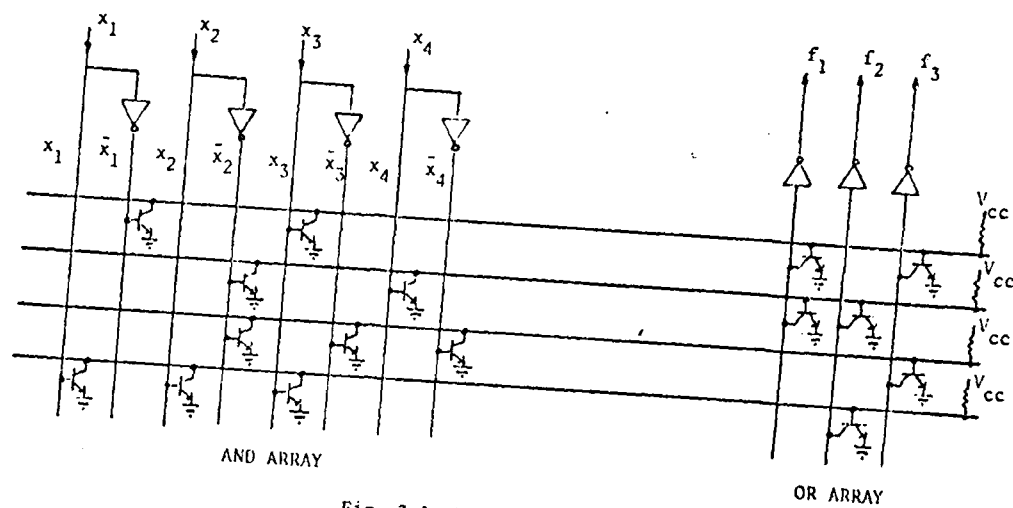


Fig. 3.1 Basic PLA

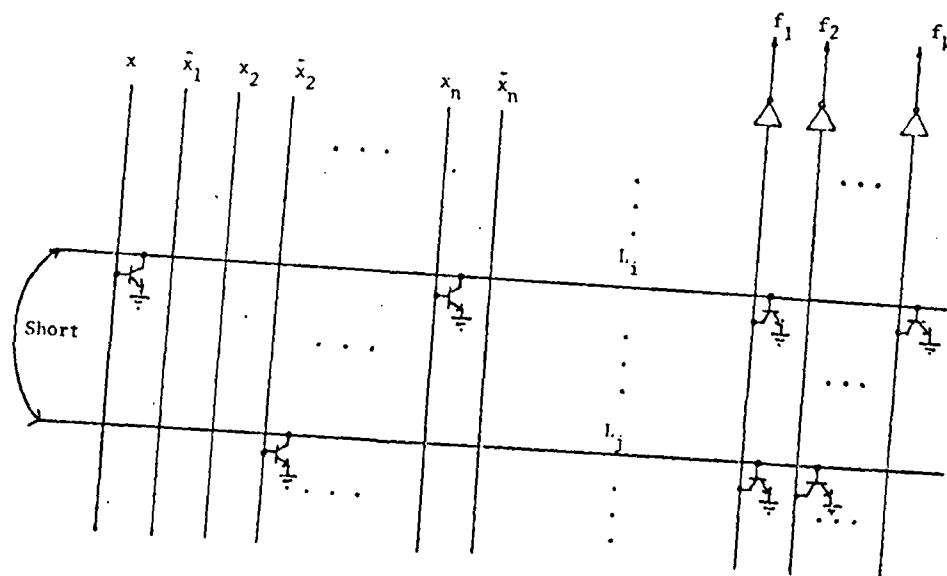


Fig. 3.2 Typical Pair of Word Lines

3.11 Crosspoint Defects

If in the AND array there is a missing device and the missing device should provide a connection to input line x_i^* , then the logical product mechanized on the word line to which the transistor should have been connected will have the variable $\bar{x}_i^*$ missing. In other words, the product $P = \bar{x}_i^* x_j \cdots x_s$ will become $P' = x_j \cdots x_s$. We say that this fault causes a growth because P' covers a larger subcube (implicant) than P , and so the function to which the word line is connected will have its true body enlarged (providing the connection to x_i^* was not redundant). It is easy to see that when more than one device is missing in the AND array, there results a growth in one or more of the functions realized in the array. In particular, if a word line has no connection in the AND array, then all functions to which that word line is connected in the OR array are set to the logical constant 1.

It follows that checking C_0 will detect any combination of missing devices in the AND array.

If there is a device missing in the OR array and that device should connect row r to function k , f_k , then the product realized on row r is no longer an implicant of f_k . We say that the missing device causes a drop because the true body of f_k has been diminished by the dropping of a product in the sum (unless the missing product was redundant). More than one missing device in the OR array will cause various kinds of drops, and any multiplicity of these will always be detected by checking C_0 .

Missing devices in both the AND array and the OR array, however, are not necessarily detected by checking C_0 . Consider, for example, the realization of the majority function $M = AB + AC + BC$, with word lines L_1 through L_3 realizing the products AB , AC , and BC , respectively. If there is a device missing between input line $\bar{A}$ and L_1 as well as devices missing where connections to lines L_2 and L_3 should be made, the faulty function realized is $M' = B$. Because M' has four ones as does M , checking C_0 will fail to reveal the assumed multiple fault pattern.

An extra device in the AND array connected to input x_j^* will result in the product realized on the corresponding word line having $\bar{x}_j^*$ added. We will call this a shrinkage, because the augmented product will cover a smaller subcube (implicant), and so the function to which the word line with the extra device is connected will have its true body diminished. Unless there is redundancy, the effect of the extra device will be detected by checking C_0 . Similarly, multiple extra devices in the AND array will also be detected. For the special case where the extra device connects to x_j^* and the word line also has a connection to $\bar{x}_j^*$, the result is a drop of the nominal product, and C_0 will detect this case as well.

One or more extra devices in the OR array add extra products to the function(s) realized. These faults are detected by checking C_0 .

No general statement can be made about the effectiveness of checking C_0 in the presence of extra devices in both the AND array and the OR array.

In summary, we have shown that checking C_0 detects all single crosspoint defects as well as a variety of multiple defects, but not all possible combinations.

3.12 Stuck Lines

An input line stuck in 1 causes all word lines connected to that input line to be set to logical zero. This causes one or more drops and is detected by checking C_0 . An input line x_i^* stuck in 0 causes every product with nominal $\bar{x}_i^*$ to become independent of $\bar{x}_i^*$ ($P = \bar{x}_i^* x_j \cdots x_k$ becomes $P' = x_j \cdots x_k$). Hence this fault causes one or more growths and these are detected by checking C_0 .

It is easy to see that output lines stuck are detected by checking C_0 .

A word line stuck in 1(0) is the extreme case of a growth (drop) i.e., the corresponding product has grown to the logical constant 1(0).

3.13 Shorts

Consider Fig. 3.2 and let there be a short between word lines L_i and L_j . This short has an effect only when the inputs are such that the nominal product on L_i is true (false) and that on L_j is false (true). In the first case, L_i is nominally high (low) while L_j is nominally low (high). The short, however, makes both lines low in both cases, so that the word lines act as though there were devices in all places where there are input-line connections to L_i or L_j . In other words, the behavior due to the short is equivalent to extra devices in the AND array. As was pointed out above, any combination of extra devices in the AND array is detected by checking C_0 , and thus shorted word lines in any combination are detected.

Next, we consider those lines that feed the inputs to the inverters; we will call these lines "function lines". Since shorts between lines cause the lower line to dominate, both shorted lines will carry logical 0 when either has a nominal 0. In the case of shorted function lines, this is equivalent to both lines having devices in all the places of the OR array where either has a device. As was shown before, extra devices in the OR array are detected by checking C_0 , and therefore so are shorts between function lines.

A short between a word line L_i and a function line that nominally is not connected to L_i will result in lowering the function line when one or more of the transistors connected to L_i conducts. Since the function realized on L_i is a product P_i (of the complements of the variables connected to transistors on that word line), the word line is low whenever one or more of the connected variables is true, and so the output f_j in the presence of the short becomes $f'_j = f_j + \bar{P}_i$. Thus the true body of the faulty output is enlarged, and verifying C_0 for output f_j does check for the assumed type of short.

To illustrate, suppose there is a short in the OR array of Fig. 3.1 between function line 2 and the topmost word line. Then f_2 becomes one whenever $\bar{x}_1$ is high or x_3 is high. Thus we get the faulty function $f'_2 = x_2\bar{x}_4 + \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1 + x_3 = x_2\bar{x}_4 + \bar{x}_1 + x_3$, which has 13 ones, whereas the fault-free output f_2 has only six ones.

In the case where the short involves a function line that is connected to the word line L_i in the fault-free circuit, the effect is that of a short between base and collector of the transistor that makes the connection. This causes the function line to become stuck in 0, i.e., the output will be stuck in 1. This is detected by checking C_0 .

When an input line connected to x_i^* is shorted to word line L_i , then L_i is low whenever x_i^* is low or one or more of the other inputs x_j^* through x_k^* to L_i is low. Thus instead of realizing the product $P_i = \bar{x}_j^* \cdots \bar{x}_k^*$, the word line realizes $P_i' = x_i^* \cdot P_i$, which is called a shrinkage, if the fault-free circuit has no connection to x_i^* ; if it does have a connection to x_i^* , then there is a short between base and collector of the connecting transistor. This makes L_i stuck in 0, i.e., it causes a drop. Checking C_0 on the corresponding output will detect the fault in both cases.

Since shorts between lines cause the lower line to dominate, both of the shorted lines will be low when either is nominally low. If both

of the shorted lines, x_i^* and x_j^* , are connected to word line L_m , then the faulty product on L_m would be $P'_m = (\bar{x}_i^* + \bar{x}_j^*) \cdot x_p^* \cdots x_q^*$ instead of the nominal product $P_m = \bar{x}_i^* \cdot \bar{x}_j^* \cdot x_p^* \cdots x_q^*$. If one of the lines, x_i^* (x_j^*), is connected to word line L_n , then $P'_n = P_n + P_n^0 \cdot \bar{x}_j^* (\bar{x}_i^*)$, where the nominal product is $P_n = P_n^0 \cdot \bar{x}_i^* (\bar{x}_j^*)$. In either case, the true body of the product is enlarged, and so is that of the corresponding output f_i . It is apparent that checking C_0 for the output f_i always detects shorts between input lines.

Similarly to before, when an input line connected to x_i^* and a function line are shorted, then the function line will mechanize $\bar{f}'_j = \bar{f}_j \cdot x_j^*$ ($f'_j = f_j + \bar{x}_i^*$), because the lower line always dominates the higher one. (Whenever x_i^* is low, the function line will be low; the function line is still low regardless of the x_i^* value whenever the nominal function line is low). The true body of output j is thus enlarged. This short can also be detected on some other output. If the input line x_i^* is connected to word line L_k , then the product realized on word line L_k in the presence of the short will become $P'_k = P_k + P_k^0 \cdot f_j$, where the nominal product is $P_k = P_k^0 \cdot \bar{x}_i^*$. This growth in P_k can be detected on any output that is connected to L_k . Since in both cases the true body of the output is enlarged, it is easy to see that the shorts assumed here are also detected by checking C_0 .

3.14 More General Decoder Form and Application of C_0

Sometimes it is advantageous to use more than one literal (typically two) as the inputs to the PLA's. For simplicity, consider Fig. 3.3, where two literals are used as the inputs to the AND array and n-p-n transistors are used in the NOR-OR logic form in the PLA. From Fig. 3.3, where the decoders are of the form shown in Fig. 3.4, we have

$$P_1 = \text{Product mechanized on } L_1 = (x_1 + \bar{x}_2)(\bar{x}_1 + x_2)(\bar{x}_3 + \bar{x}_4)(x_5 + \bar{x}_6)$$

$$P_2 = \text{Product mechanized on } L_2 = (x_1 + x_2)(x_3 + \bar{x}_4)(x_5 + \bar{x}_6)(\bar{x}_5 + \bar{x}_6)$$

$$P_3 = \text{Product mechanized on } L_3 = (x_1 + \bar{x}_2)(\bar{x}_1 + \bar{x}_2)(\bar{x}_3 + x_4)(\bar{x}_3 + \bar{x}_4)(x_5 + x_6)$$

$$P_4 = \text{Product mechanized on } L_4 = (x_1 + x_2)(\bar{x}_1 + x_2)(x_3 + x_4)(\bar{x}_5 + x_6)$$

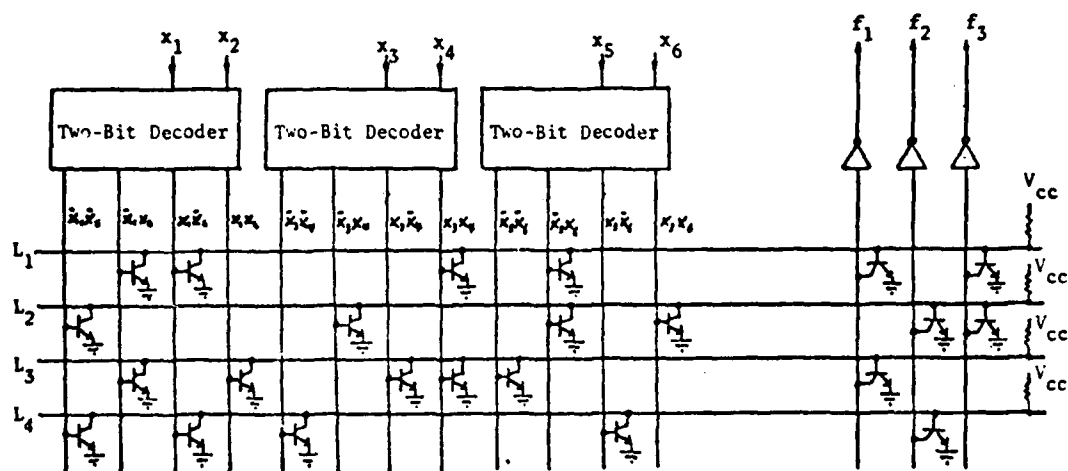


Fig. 3.3 PLA with Two-Bit Decoders

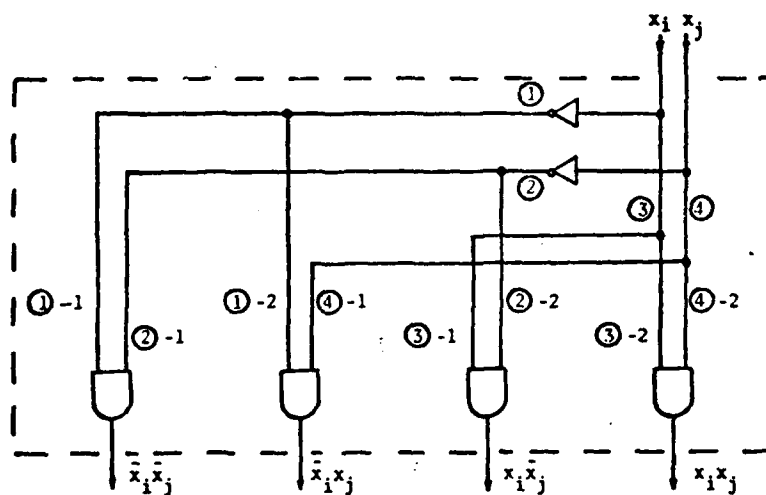


Fig. 3.4 Two-Bit Decoder

$$f_1 = P_1 + P_3$$

$$f_2 = P_2 + P_4$$

$$f_3 = P_1 + P_2$$

In general, the product P_m realized on the word line L_m is

$$P_m = (x_i^* + x_j^*)(x_k^* + x_l^*) \cdots (x_m^* + x_n^*), \text{ where } i \neq j, k \neq l, \text{ and } m \neq n$$

3.141 Single or Multiple Missing Devices

If the missing device is in the AND array and the missing device should provide a connection to the input line $x_i^*x_j^*$, then the product

$P = (\bar{x}_i^* + \bar{x}_j^*)(x_k^* + x_l^*) \cdots (x_m^* + x_n^*)$ will become $P' = (x_k^* + x_l^*) \cdots (x_m^* + x_n^*)$, so that this fault causes a growth. Similarly, more than one device missing in the AND array results in a growth in one or more of the functions realized in the array.

If the missing device is in the OR array and the device should connect row r to function f , then the function $f = P_r + P_i + \cdots + P_j$ will become $f' = P_i + \cdots + P_j$, so that this fault causes a drop. It is evident that more than one device missing in the OR array results in various kinds of drops. In all cases C_0 verification can serve as a test.

3.142 Single or Multiple Extra Devices

An extra device in the AND array connected to input $x_i^*x_j^*$ will result in the product realized on the corresponding word line having $(\bar{x}_i^* + \bar{x}_j^*)$ added, which causes a shrinkage. (The nominal product

$P = (x_l^* + x_m^*) \cdots (x_p^* + x_q^*)$ will become $P' = (\bar{x}_i^* + \bar{x}_j^*)(x_l^* + x_m^*) \cdots (x_p^* + x_q^*)$ due to the fault). Similarly, multiple extra devices in the AND array result in a shrinkage in one or more of the functions and this is detected by checking C_0 in the array.

If an extra device in the OR array connects row r to function f , then the function $f = P_i + \cdots + P_j$ will become $f' = f + P_r$, so that

the true body is enlarged by the fault. Multiple extra devices in the OR array enlarge the true body of one or more of the corresponding outputs. Clearly these faults are all detected by checking C_0 .

3.143 Stuck Lines in the PLA

As explained in Section 3.12, most of the stuck lines are equivalent to single (multiple) missing devices or single (multiple) extra devices, and it is not difficult to see that stuck lines not equivalent to missing or extra devices are also detected by checking C_0 .

3.144 Shorts in the PLA

In all cases of shorted lines, the same arguments as given under shorts in Section 3.13 are applicable here, except that the input literals in the products are not x_i^* but $(x_i^* + x_j^*)$ for $i \neq j$. Therefore, we can say that the effectiveness of testing by verifying C_0 is not reduced when a two-bit decoder is used.

3.145 Stuck Lines in the Decoders

Refer to Fig. 3.4. Assuming that all pins are fault-free, input lines to the AND gates in the decoder stuck in 0 are equivalent to output lines from the AND gates stuck in 0, which are equivalent to input lines to the AND array of the PLA stuck in 0.

If the fan-out line x_i^* to the AND gate (i.e., $\textcircled{i} - 1$ or $\textcircled{i} - 2$ in Fig. 3.4) is stuck in 1, then the output from the corresponding AND gate becomes x_j^* , while nominally it is $x_i^* x_j^*$, so that the word line L_m connected to the output from the decoder realizes $P'_m = \bar{x}_j^* (x_l^* + x_m^*) \cdots (x_p^* + x_q^*)$ as its product, whereas its nominal product is $P_m = (\bar{x}_i^* + \bar{x}_j^*) (x_l^* + x_m^*) \cdots (x_p^* + x_q^*)$. If the fan-out point x_i^* to the AND gate (i.e., $\textcircled{i}$ in Fig. 3.4) is stuck in 1, then the two outputs from the AND gates become x_j^* and $\bar{x}_j^*$, where the nominal outputs are $x_i^* x_j^*$ and $x_i^* \bar{x}_j^*$, respectively. This causes the word line L_m to realize $P'_m = \bar{x}_j^* \cdot x_j^* \cdot (x_l^* + x_m^*) \cdots (x_p^* + x_q^*) = 0$, if both outputs from the decoder are connected to the word line L_m . Otherwise (i.e., only one output connected),

the faulty product becomes $P'_m = \bar{x}_j^*(x_\ell^* + x_m^*) \cdots (x_p^* + x_q^*)$ or $P'_m = x_j^*(x_\ell^* + x_m^*) \cdots (x_p^* + x_q^*)$. Since in either case the fault causes a drop, this is detected by checking C_0 .

3.15 Testing by Verifying C_{ALL}

As shown in a previous Chapter 2, checking C_{ALL} of the output f completely tests for single or multiple pin faults. These are faults in which, if input x_i is stuck-in-1, input $\bar{x}_i$ is stuck in 0. For simplicity, we consider here only the PLA where all inputs to the AND array are provided by one-bit decoders and limit our discussion to faults other than pin faults, since these are surely detected by C_{ALL} testing. Note that in all of the previous sections of this chapter we never considered pin faults. These are not, in general, detected by checking C_0 . But as will be seen below, verifying C_{ALL} is a weak test for PLA's other than for detecting pin faults.

3.151 Stuck Lines

Single or multiple input lines x_i^* to the AND array stuck in 1 (0) are detected by checking C_{ALL} of the output f_i if and only if all word lines feeding the output f_i in the OR array are not connected to an input $\bar{x}_i^*$. Single or multiple output lines f_i in the OR array stuck in 1 (0) are always detected by checking C_{ALL} , while single or multiple word lines stuck in 1 (0) are not detected by checking C_{ALL} of the output f_j , unless the stuck lines are the only ones feeding the output f_j .

3.152 Missing or Extra Devices

As shown in Section 3.11, some cases of missing or extra devices are equivalent to stuck lines. Some of these faults can be detected by checking C_{ALL} , but not all.

3.153 Shorts

Only a few of the possible shorts can be detected by checking C_{ALL} .

3.2 Testing the Associative Logic Matrix

Greer's Associative Logic Matrix [3.9] makes possible the efficient realization of multiple output, multiple level, combinational and sequential networks by means of the regular interconnection structure of read-only memory and programmable logic arrays. For the implementation of complex multiple-output Boolean functions, which frequently can be expressed efficiently in more than two levels of logic, Associative Logic Matrices (ALM's) may well be advantageous over Programmable Logic Arrays (PLA's), which are typically restricted to two-level logic.

The ability to implement networks involving more than two levels of logic is achieved in the ALM through the use of "internal function logic". This logic involves additional bit lines which serve the dual role of forming logical sums (or products) and providing the resulting signals as inputs in the formation of subsequent functions. For simplicity, we restrict the realization of associative logic to four-level combinational circuits. All connections in the array are "wired-NOR"ed by means of n-p-n transistors. In Fig. 3.5, the internal function g is $g = x_1x_2 + x_3$, one output is $f_1 = x_4g + \bar{x}_1\bar{g} + x_1\bar{x}_2\bar{x}_4$, and another output is $f_2 = x_4g + \bar{x}_2\bar{g}$.

The structure of the ALM differs from that of the PLA in the addition of the G-array, which realizes the internal functions. Hence the ALM consists of the AND array, the OR array, and the G-array, as shown in Fig. 3.5. The rightmost bit line of the G-array will be called the "collector line of the G-array" and the other two lines, which are used as inputs in the formation of the output functions, will be called the "output line g of the G-array" and the "output line $\bar{g}$ of the G-array", respectively, as shown in Fig. 3.5.

To ease fault-detection in the ALM's, we will add extra logic. It consists of one extra output, f_e , which is fed by all word lines connected to the output line $\bar{g}$ of the G-array. Thus f_e is of the form $f_e = gx_1^* \cdots x_j^* + \cdots + gx_k^* \cdots x_l^*$. (In Fig. 3.5, the extra output is $f_e = gx_4$).

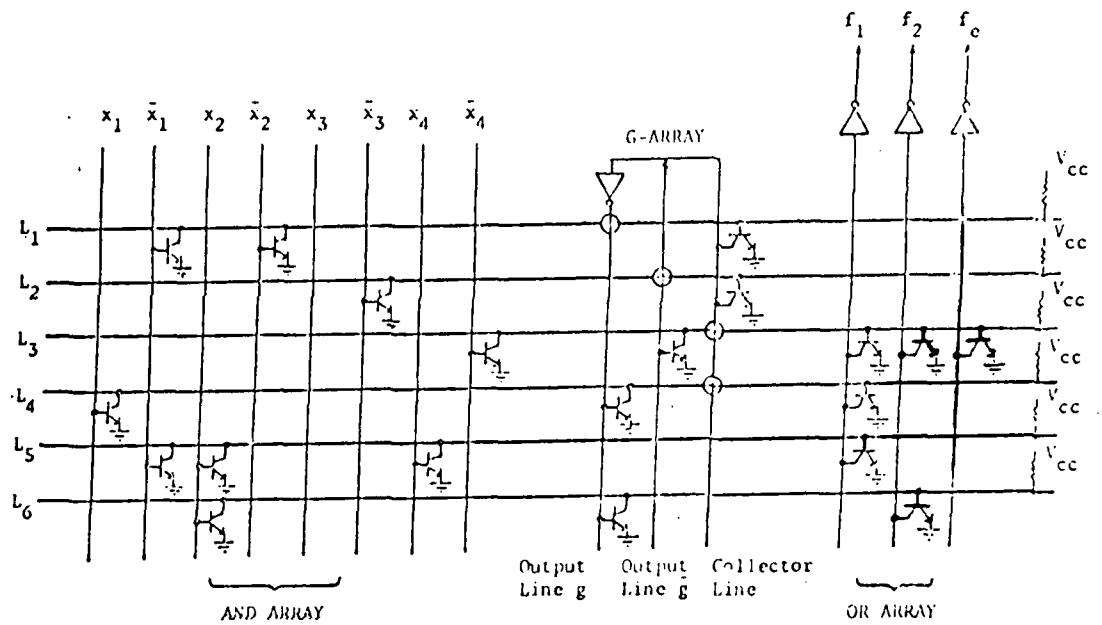


Fig. 3.5 Associative Logic Matrix

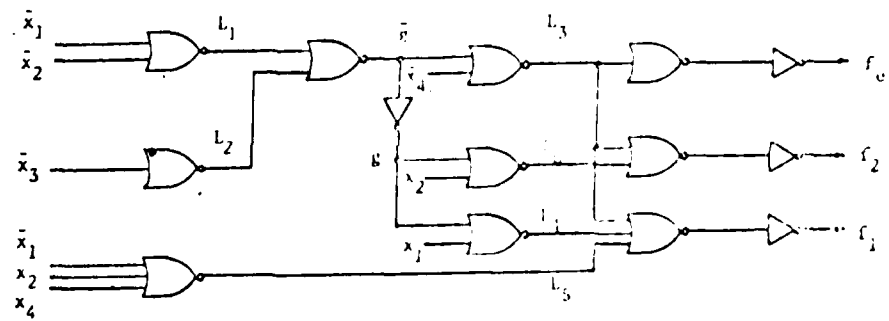


Fig. 3.6 ALM of Fig. 3.5 Redrawn

To avoid duplicating previous explanations in Section 3.1, we list those faults that have the same effect in ALM's as in PLA's and omit further discussion of these:

1. Single or multiple missing (extra) devices in the AND array, where the corresponding word lines do not feed the internal function (i.e., the corresponding word lines are not connected to the collector line of the G-array).
2. Single or multiple missing (extra) devices in the OR array.
3. Stuck lines not in the G-array.
4. Shorts between word lines, unless one or both of the shorted lines feed the internal function.
5. Shorts between input lines in the AND array that have no path to the collector line of the G-array.
6. Shorts between output lines in the OR array.
7. Shorts between a word line and an output line in the OR array.
8. Shorts between a word line and an input line in the AND array.
9. Shorts between an input line in the AND array and an output line in the OR array.

3.21 Effect of Other Faults in the ALM's

3.211 Single or Multiple Missing Devices

If a missing device in the AND array should provide a connection to the input line x_i^* and the word line L_k to which the device should have been connected is one of the word lines feeding the internal function g , then the product realized on L_k will become $P_k' = x_m^* \cdots x_n^*$ instead of $P_k = \bar{x}_i^* \cdot P_k'$, which we have called a growth. This fault will enlarge the true body of the internal function g and also that of the extra output f_e . Checking C_0 on f_e will detect the fault. It is easy to see that multiple missing devices in the AND array will be detected by checking C_0 on f_e if at least one of the corresponding word lines feeds the internal function g .

If a missing device in the G-array should provide a connection to the output line g^* , then the product realized on its corresponding word line L_m will become $P'_m = x_1^* \cdots x_m^*$ instead of $P_m = \bar{g}^* \cdot P'_m$, and hence there is a growth. Since this will enlarge the true body of the output f_i fed by the word line L_m , checking C_0 on f_i will detect it.

A missing device in the G-array which should provide a connection to the collector line drops the product on its corresponding word line, and so the true body of the extra output f_e is reduced, which will be detected by checking C_0 on f_e . Moreover, either multiple missing devices in the AND array and the output line g^* of the G-array or missing devices in the OR array and the collector line of the G-array will be surely detected by checking C_0 . But multiple missing devices in both the output line g^* and the collector line of the G-array are not necessarily detected by checking C_0 .

3.212 Single or Multiple Extra Devices

If an extra device in the AND array connects the word line L_k to the input x_1^* and the word line L_k does feed the internal function g , then the product on L_k will become $P'_k = \bar{x}_1^* \cdot P_k$, which we have called a shrinkage. This will reduce the true body of the internal function g and also that of the extra output f_e . Checking C_0 of f_e will detect this fault. Multiple extra devices in the AND array will be surely detected by checking C_0 of the extra output f_e , if at least one of the corresponding word lines feeds the internal function g .

One or more extra devices in the output line g^* of the G-array, except for special case A discussed below, will be detected by checking C_0 of the output f_k fed by the corresponding word line, because the fault causes a shrinkage and reduces the true body of the output f_k . It is not difficult to see that multiple extra devices in both the AND array and the output line g^* will be detected by checking C_0 . One or more extra devices in the collector line of the G-array, except for case B treated below, cause one or more products realized on the corresponding word lines to become additional implicants of the internal

function g , and so the true body of the extra output f_e is enlarged, which will be surely detected by checking C_0 of f_e . Multiple extra devices in both the OR array and the collector line of the G-array will also be detected by checking C_0 .

Case A

If one or more extra devices are connected to the output line g^* of the G-array and at least one of the corresponding word lines nominally feeds the internal function g , then this fault will cause feedback. Consider Fig. 3.5 and Fig. 3.6, which is a conventional representation of Fig. 3.5. If an extra device connects the output line g to the word line L_1 , then the extra device will cause feedback, as shown in Fig. 3.7. (This situation is illustrated by the top circle in Fig. 3.5). Suppose x_1 , x_2 , and x_3 are, respectively, 1, 1, and 0, so that nominally $L_1 = 1$, $L_2 = 0$ and $g = 1$. With the fault, however, if g were 1, the three inversions around the closed loop would complement g , so the value of g could not remain 1 and in fact the value of g would oscillate. This is the kind of fault that a static type of test just cannot detect; only waveform observation will be sure to result in detection.

If an extra device connects the output line $\bar{g}$ to a word line L_2 , then it will cause feedback as shown in Fig. 3.8. (This situation is illustrated by the circle on Line L_2 of Fig. 3.5.) This feedback over an even number of inversions can be detected by a sequence of two tests. The first makes both L_1 and L_2 low, so that $\bar{g} = 1$. This is a stable condition in the presence of the feedback. The second test makes L_2 nominally high while keeping L_1 low, so that nominally $\bar{g} = 0$. If feedback is present, however, the second test will leave $\bar{g}$ unchanged, i.e., it remains 1. While the assumed fault is detectable, simply checking C_0 will not always work.

Case B

If one or more extra devices connect word lines to the collector line of the G-array and at least one of the corresponding word lines is nominally connected to the output line g^* , then this will cause

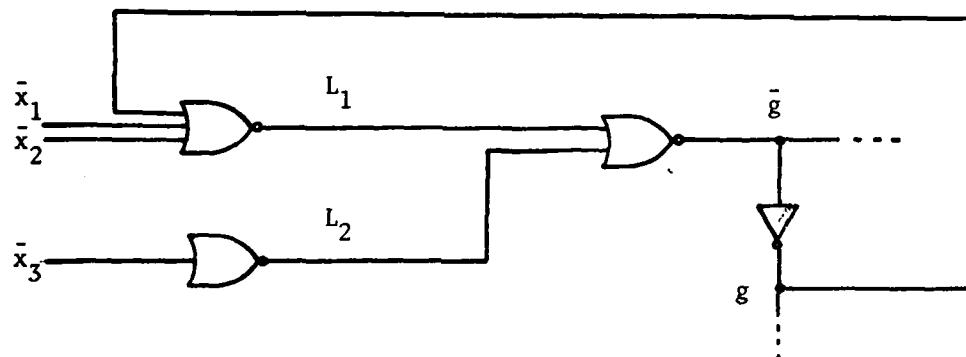


Fig. 3.7 Feedback from g Due to Extra Device

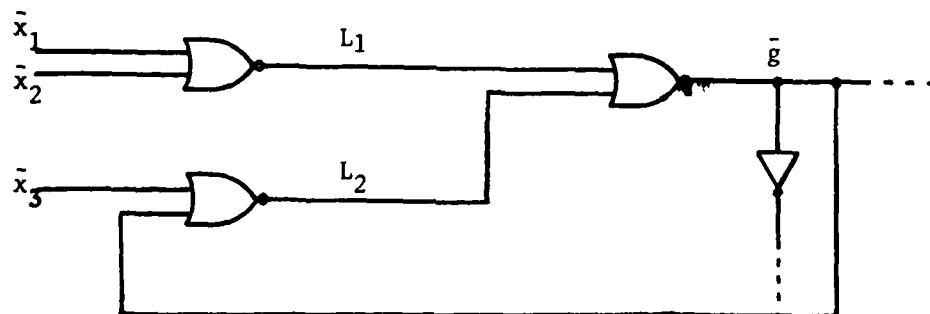


Fig. 3.8 Feedback from $\bar{g}$ Due to Extra Device

feedback. Refer to Figs. 3.5 and 3.6. If an extra device connects the word line L_3 to the collector line of the G-array, then the extra device will cause feedback as shown in Fig. 3.9. (This situation is illustrated by the circle on the third line of Fig. 3.5.) As before, this feedback can be detected by a sequence of two tests. The first makes both $\bar{g}$ and $\bar{x}_4$ low, so that $L_3 = 1$, which is a stable condition in the presence of the feedback. The second test makes $\bar{g}$ nominally high while keeping $\bar{x}_4$ low, so that nominally $L_3 = 0$. If feedback is present, then the second test will leave L_3 unchanged, i.e., it remains 1. However, simply checking C_0 will not necessarily detect this fault.

If an extra device connects the word line L_4 to the collector line of the G-array, then the extra device will cause feedback as shown in Fig. 3.10. (This is illustrated by the circle on the fourth line of Fig. 3.5.) As before, when an input combination which makes all of L_1 , L_2 , and x_1 low is given, the value of L_4 will oscillate due to the feedback. A static type of test cannot detect this fault; only waveform observation can.

3.213 Stuck Lines in the G-array

Since an output line g^* stuck in the G-array is equivalent to one or more missing devices or one or more extra devices in the G-array, this will be easily detected by checking C_0 .

If the collector line of the G-array is stuck at 1(0), then the fault will be equivalent to both the output line g stuck at 0(1) and the output line $\bar{g}$ stuck at 1(0). The output line $\bar{g}$ stuck at 1(0) reduces (enlarges) the true body of the extra output f_e , so that checking C_0 of f_e will detect the collector line stuck. If a word line L_k is stuck and L_k is one of the word lines feeding the internal function g , then the fault will surely enlarge (reduce) the true body of the internal function g and also that of the extra output f_e . This fault will be detected by checking C_0 of f_e .

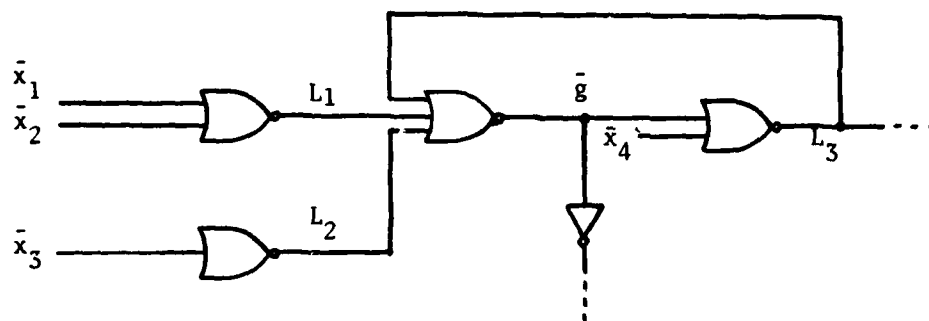


Fig. 3.9 Feedback Due to Extra Device on Collector Line

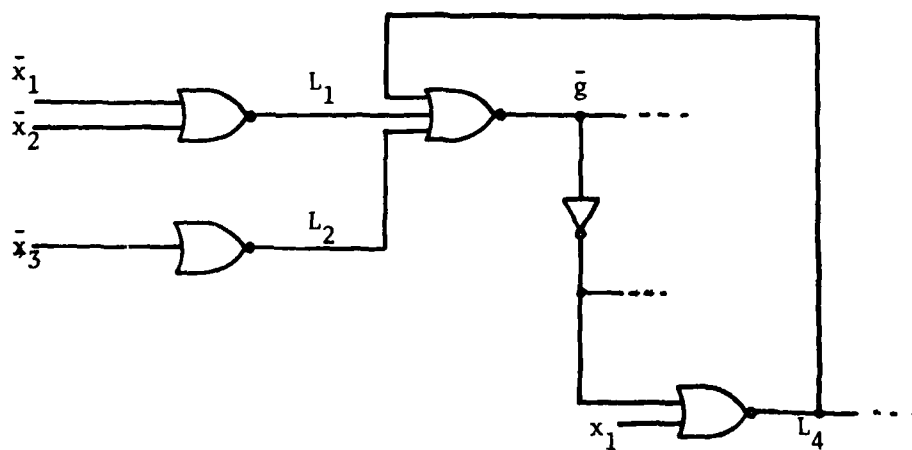


Fig. 3.10 Feedback that Can Cause Oscillation

3.214 Shorts Between Word Lines

If both of the shorted word lines feed the internal function, then the short between lines is equivalent to extra devices in both word lines. Extra devices in both lines will reduce the true body of the internal function g and also that of the extra output f_e , so that checking C_0 of f_e will surely detect the fault.

Similarly, if one of the shorted lines feeds the internal function g and the other, L_k , is not connected to the output line g^* of the G-array, then both the true body of the internal function and that of the output fed by the word line L_k will be reduced, because the short is equivalent to extra devices in both lines. This will also be detected by checking C_0 .

However, if one of the lines feeds the internal function g and the other is connected to the output line $\bar{g}$ (g) of the G-array, then the fault will (not) always be detected by checking C_0 . Refer to Fig. 3.5 and Fig. 3.6. If the word line L_1 is shorted to the word line L_4 , which is connected to the output line g of the G-array, then the circuit will be changed to that shown in Fig. 3.11, where the signals on L_1 and L_4 will oscillate. This is not detected by a static type of test. If the word line L_1 is shorted to the word line L_3 connected to the output line $\bar{g}$ of the G-array, then this will be detected by checking C_0 on the extra output f_e . Refer to Fig. 3.12. Once L_1 becomes low both L_1 and L_3 will be stuck at 0 due to the short, so that the true body of f_e will be reduced.

3.215 Shorts Between Input Lines in the AND Array

As explained in the discussion of the PLA, shorts between input lines enlarge the true body of the products realized on all word lines connected to those input lines. If one or more word lines connected to those input lines feed the internal function g , then checking C_0 on the extra output f_e will detect this fault, because the fault will enlarge the true body of the internal function g as well as the extra output f_e .

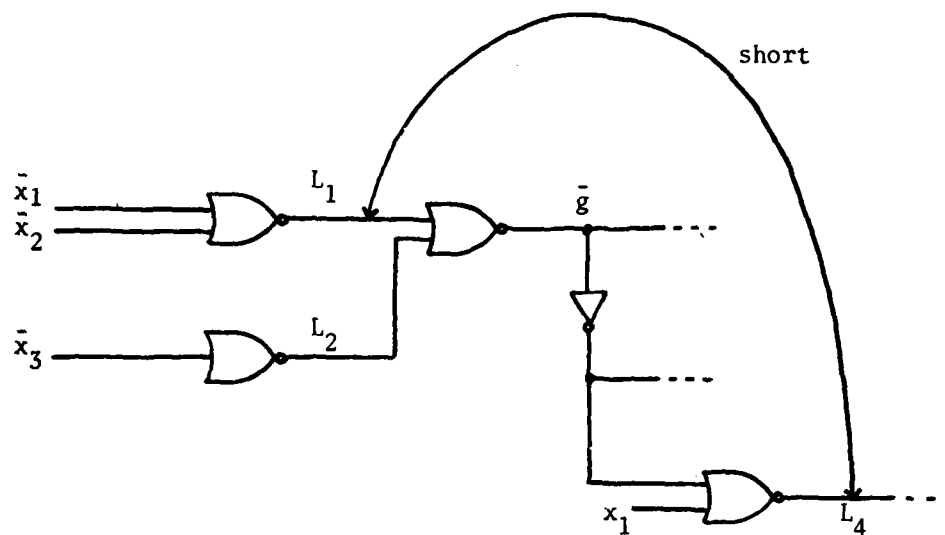


Fig. 3.11 Feedback Due to Short I

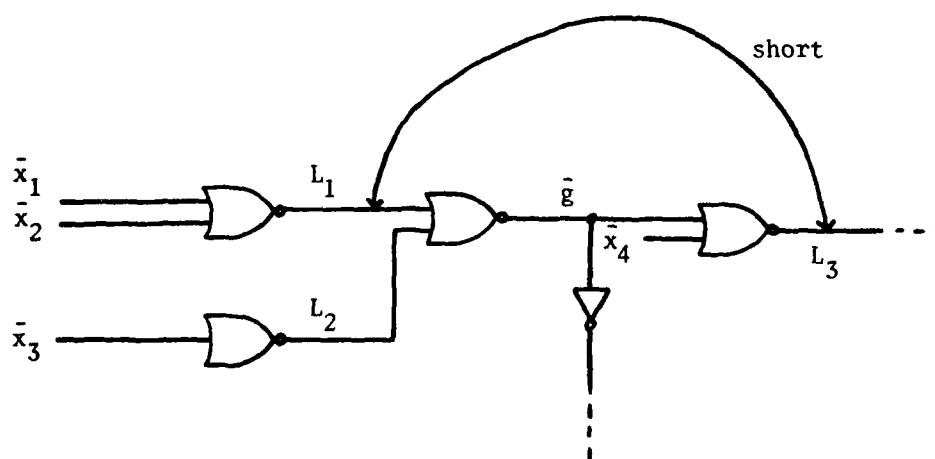


Fig. 3.12 Feedback Due to Short II

3.216 Shorts Between a Word Line and an Input Line in the AND Array

If a word line L_k is not connected to the input line x_i^* in the fault-free circuit and L_k is one of the word lines feeding the internal function g , then the product realized on L_k will become $P'_k = P_k + \bar{x}_i^*$ due to the short, because the short between lines makes the lower value dominate. This enlarges the true body of the internal function g , and also that of the extra output f_e , which is readily detected by checking C_0 of f_e . Otherwise (e.g., L_k is nominally connected to x_i^* and L_k is one of the word lines feeding the internal function g), L_k will become stuck at 0 due to the short, which connects base and collector of the transistor nominally driven by x_i^* . This will be also detected by checking C_0 of f_e .

3.217 Shorts Between a Word Line and the Collector Line of the G-array

If the shorted word line L_k is not connected to the output line g^* of the G-array and L_k does not feed the internal function g in the fault-free circuit, then the short between L_k and the collector line will result in the faulty internal function $g' = \bar{P}_k + g$, so that the true body of the extra output f_e will be enlarged. Checking C_0 of f_e will detect the fault. If L_k feeds the internal function g in the fault-free circuit, then there is a base-to-collector short and the short will result in the collector line stuck at 0. As discussed in 3.213, this is detected by checking C_0 of f .

When the word line L_k is nominally connected to the output line g of the G-array and L_k is shorted to the collector line of the G-array, then checking C_0 on the extra output f_e will detect this fault. Consider Figs. 3.5 and 3.6. If the word line L_4 and the collector line of the G-array are shorted, then the circuit will be changed to that shown in Fig. 3.13. In Fig. 3.13, since the circuit locks up with both $\bar{g}$ and L_4 stuck at 0 once $\bar{g}$ becomes low, the short will enlarge the true bodies of the internal function g and the extra output f_e . Hence, checking C_0 of f_e will detect the fault.

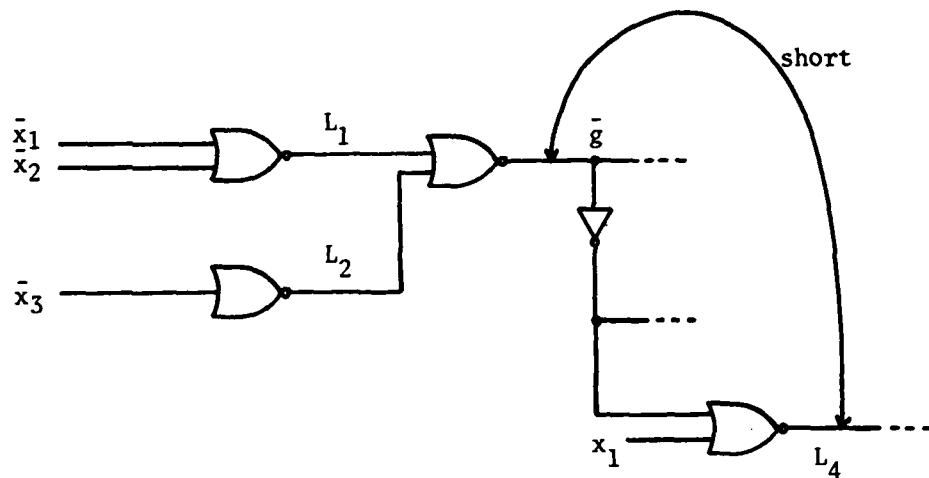


Fig. 3.13 Feedback Due to Short III

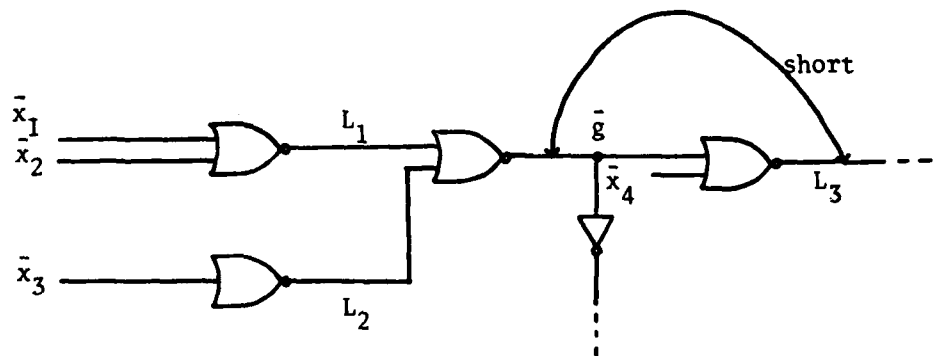


Fig. 3.14 Feedback Due to Short IV

However, if the word line L_k is nominally connected to the output line $\bar{g}$ of the G-array and it is shorted to the collector line of the G-array, then simply checking C_0 will not detect the fault. Consider Figs. 3.5 and 3.6. If the word line L_3 and the collector line of the G-array are shorted, then the circuit will be changed to that shown in Fig. 3.14. The short will make the values of $\bar{g}$ and L_3 oscillate.

3.218 Shorts Between a Word Line and an Output Line g^* of the G-array

If the word line L_k is not connected to the output line g^* and L_k does not feed the internal function g in the fault-free circuit, then the short between L_k and the output line g^* of the G-array will result in $P'_k = P_k \cdot g^*$, which will reduce the true body of the output f_i fed by the word line L_k . Checking C_0 on f_i will surely detect the fault. But if L_k is connected to the output line g^* in the fault-free circuit, then L_k will become stuck at 0 due to the fault, so that the true body of the output f_i fed by L_k will be reduced. The fault can be detected by checking C_0 of f_i .

When the word line L_k does feed the internal function g in the fault-free circuit, the short between L_k and the output line g of the G-array will be always detected by checking C_0 . Consider Figs. 3.5 and Fig. 3.6. If the word line L_1 and the output line g of the G-array are shorted, then the circuit will be changed to that shown in Fig. 3.15. As before, once L_1 becomes low, both L_1 and g will be stuck at 0. This will reduce the true body of the internal function g , so that checking C_0 of f_e will detect the fault.

However, if L_k does feed the internal function g in the fault-free circuit, the short between L_k and the output line $\bar{g}$ of the G-array will not be detected by simply checking C_0 . Consider Fig. 3.5 and Fig. 3.6. If the word line L_1 and the output line $\bar{g}$ of the G-array are shorted, then the circuit will be changed to that shown in Fig. 3.16. Since the values of L_1 and $\bar{g}$ oscillate due to the short, checking C_0 will no longer detect the fault.

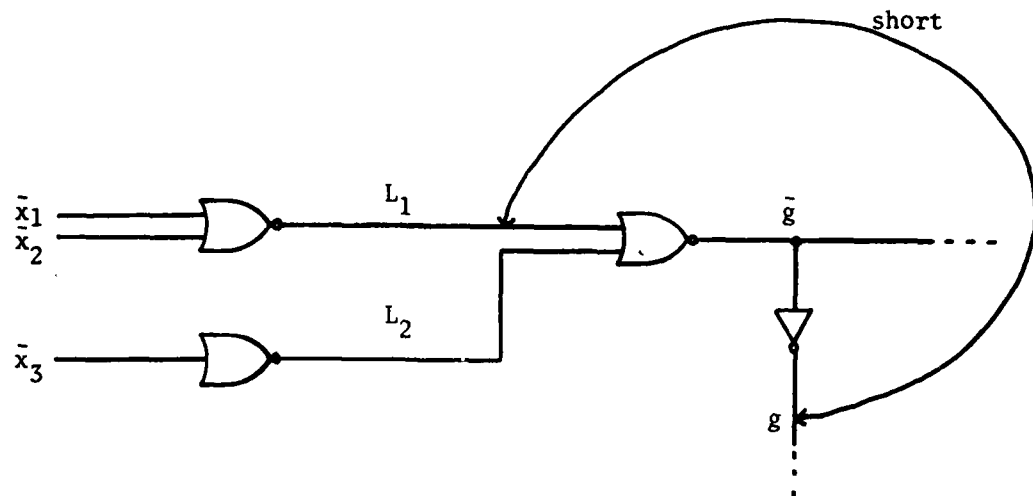


Fig. 3.15 Feedback Due to Short V

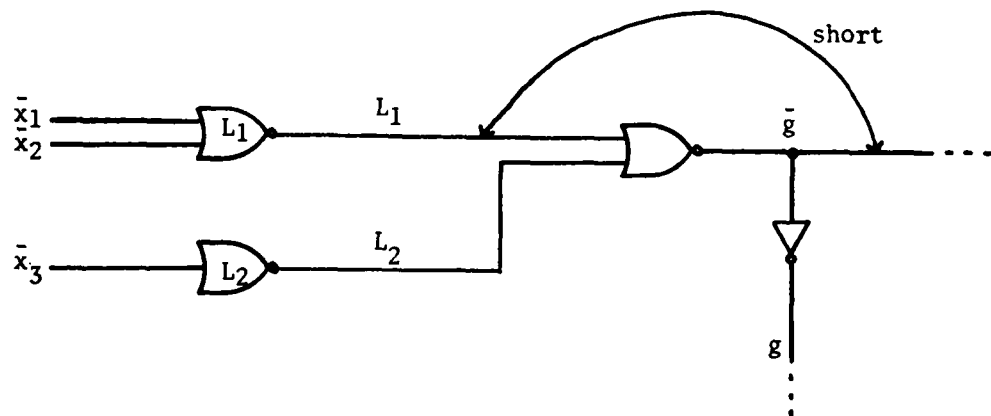


Fig. 3.16 Feedback Due to Short VI

3.22 Discussion

While the ALM, because of the addition of the G-array, is not limited to the realization of two-level combinational logic, faults in the G-array are not easily detected. In particular, some faults cause oscillation and cannot be detected by a static type of test, but only by waveform observation. Thus we conclude that testing of PLA's is easier than testing of ALM's.

3.3 Fault-Detection in Programmable Storage/Logic Arrays

Patil and Welch's programmable Storage/Logic Array (SLA) [3.10] is a form of PLA which contains flip-flops distributed throughout the array. Because in some computer designs purely combinational PLA's are difficult to use extensively due to pin limitations, some PLA's with flip-flops providing internal feedback from the outputs back to the inputs, as shown in Fig. 3.17, have been proposed [3.1], [3.2]. SLA's differ from previously described PLA's in that the AND and OR arrays are folded together so that input lines and output lines are alternated within a single array (see Fig. 3.18). As described in [3.10], "This has two important effects: (1) substantially more flip-flops can be added without the need for excess input-output routing space, and (2) rows of the array...can be subdivided into multiple independent segments which can represent independent variables over smaller portions of the array." Furthermore, the columns can also be subdivided into segments carrying independent variables with localized access by adding more flip-flops at the intervals along the columns.

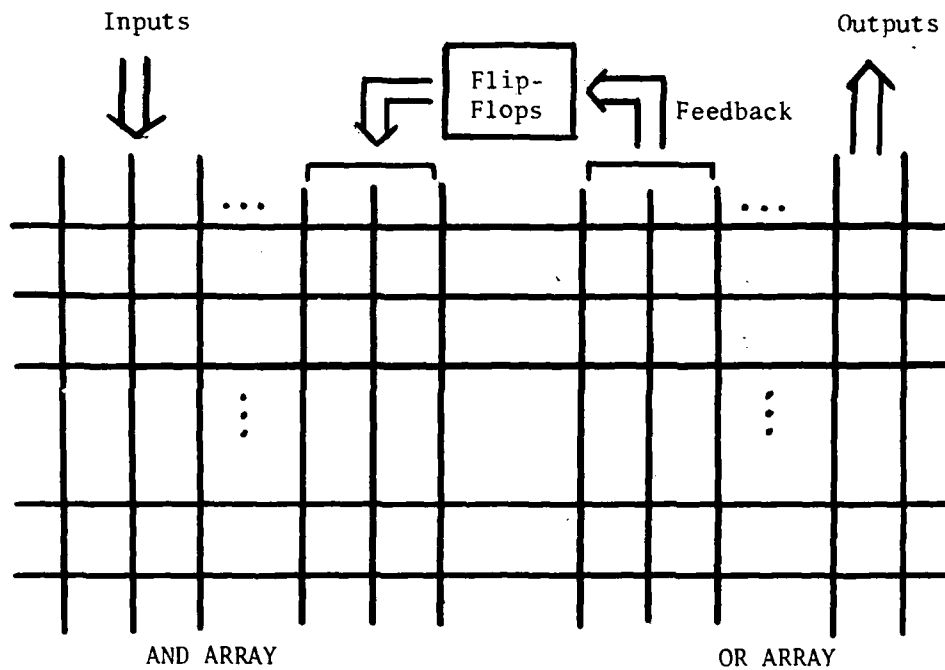


Fig. 3.17 PLA with Feedback

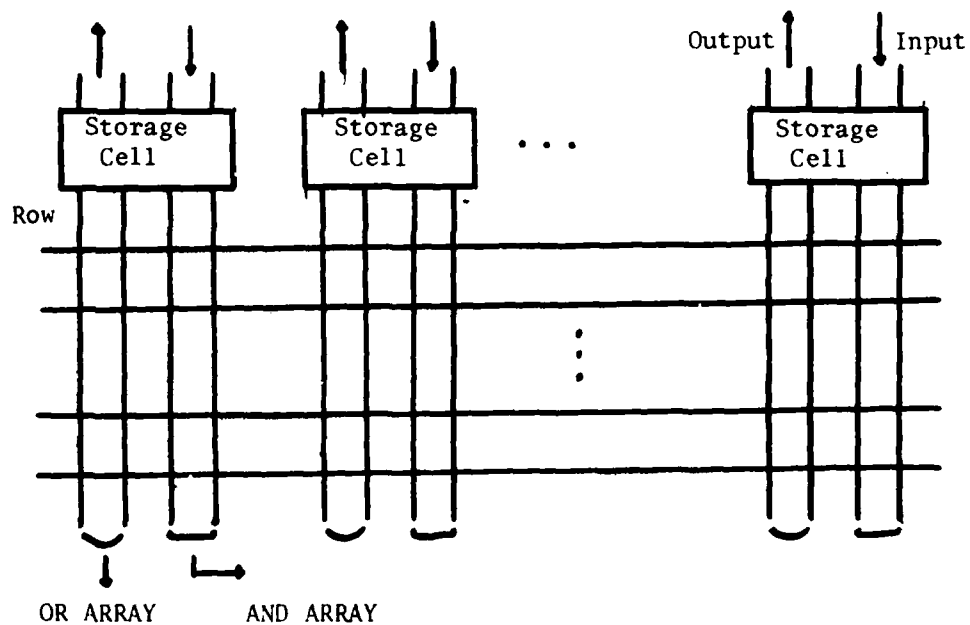


Fig. 3.18 Storage/Logic Array (SLA)

In the SLA circuit shown in Fig. 3.19, row-column connections are made by transistors with collectors that are selectively connected in a wired-NOR structure, and "storage cells" consist of cross-coupled NAND gates with complemented inputs $\bar{S}$ and $\bar{R}$ (i.e., set-reset flip-flops are used), so that the two outputs from the NAND gates, Q and $\bar{Q}$, will be 0(1) and 1(0), respectively, if the two inputs $\bar{S}$ and $\bar{R}$ are 1(0) and 0(1), respectively. Outputs Q^+ and $\bar{Q}^+$ maintain the previous values if $\bar{S}$ and $\bar{R}$ are both 1, and Q^+ and $\bar{Q}^+$ are unpredictable if $\bar{S}$ and $\bar{R}$ are both 0. (Here and later the superscript + denotes the signal shortly after set and/or reset values have been established.)

Since the leads in the storage cell contain breakpoints, it can be used, by opening the breakpoints, for purposes other than the flip-flop described above. For example, the feedback loop can be broken so that the outputs of the cell are simple combinational functions of the inputs.

As one example of SLA logic, we consider here finite-state machines (FSM's). Refer to Fig. 3.20. If the machine has m states, n bivalued inputs, and k bivalued outputs, then the total number of cells required in the SLA will be $(k + n + 3 + \lceil \log_2 m \rceil)$ where $\lceil p \rceil$ is the integer equal to or just larger than p . Of this number, n cells are used for the n inputs, one for the reset input, and one for the clock-pulse input. These cells are buffers obtained by breaking the feedback loops. A total of $\lceil \log_2 m \rceil$ flip-flop cells are used for the storage of the state variables, q_i , and k flip-flops for the storage of the k outputs, z_i . One flip-flop, F_α , is used for determining the proper time duration of the clock-pulse. This flip-flop will be called the clock-pulse modifier.

There is one row in the array for each possible state transition and the corresponding outputs. Thus if under some input state S_i can go to S_j , under another input S_i can go to S_k , and under the third input S_i stays unchanged, then there is a total of two rows involving state S_i . Given m states and p different inputs, there can be at most mp rows. The transition is made and the associated output is established

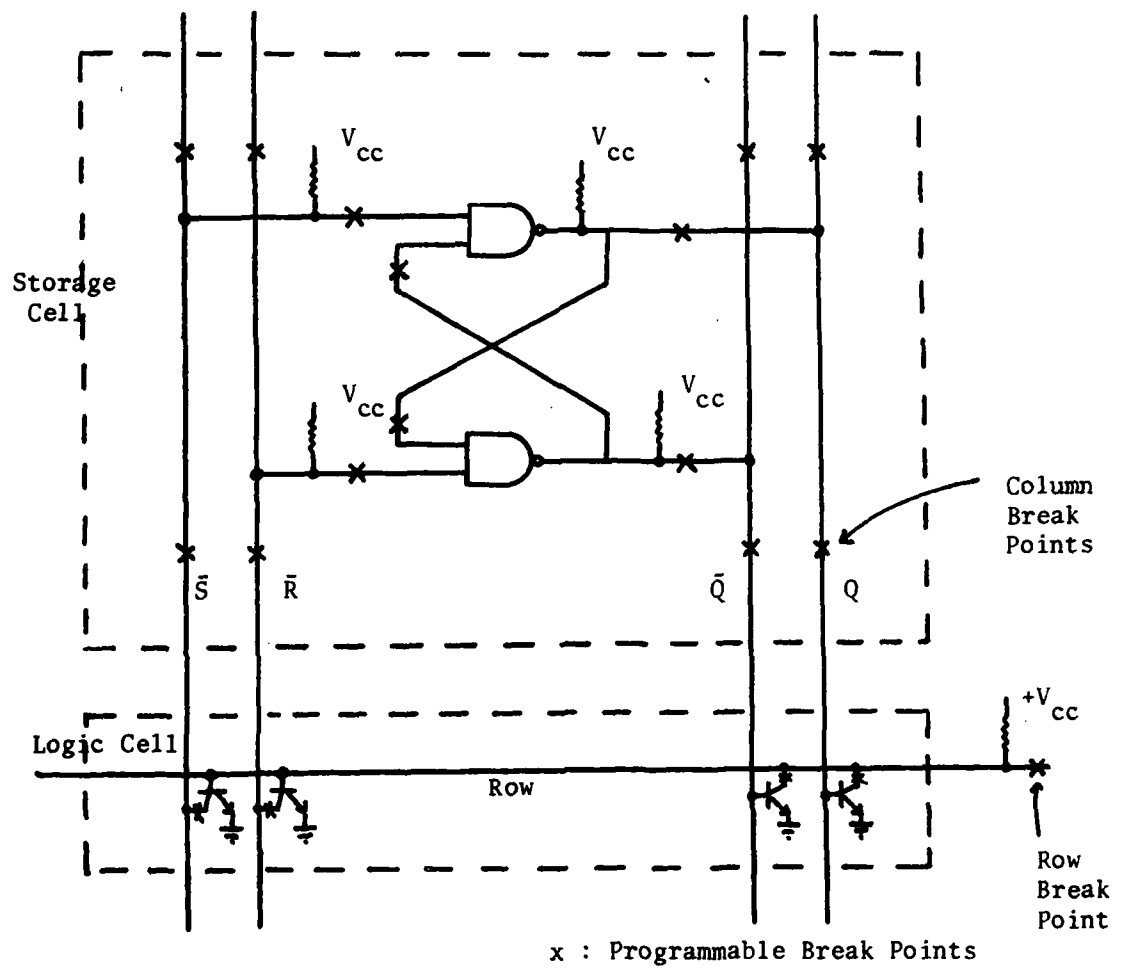


Fig. 3.19 Basic Cells of an SLA

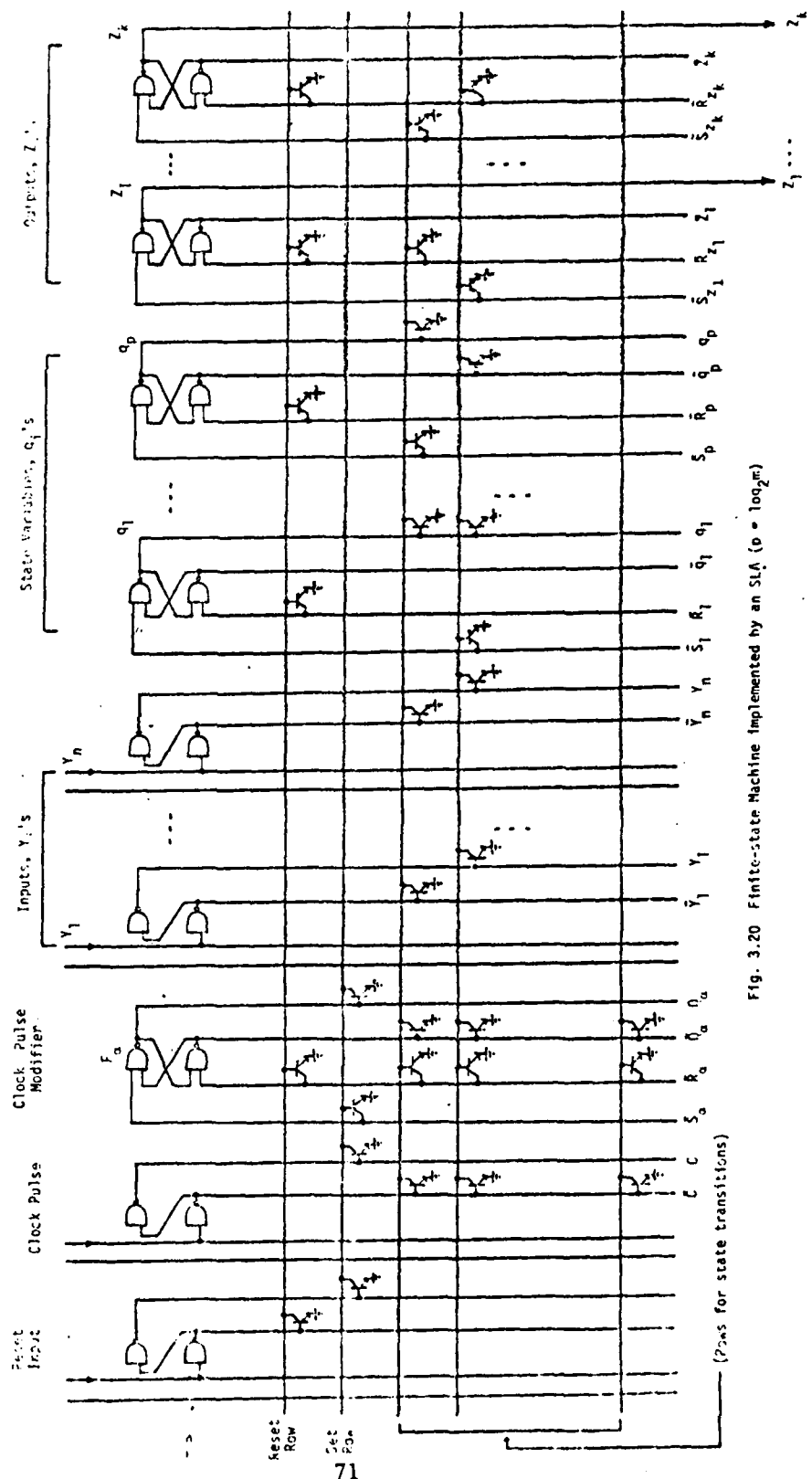


Fig. 3.20 Finite-state Machine Implemented by an SLA ($p = \log_2 m$)

when the corresponding row is activated, i.e., made high. There are also two extra rows. One will reset the machine; the other will set the machine into the initial state when activated. For proper operation the machine is first reset and then becomes set by activating the corresponding row. The row to reset the circuit, which will be called the reset row, is driven by the negated reset input, and each input line $\bar{S}_i(\bar{R}_i)$ to the state flip-flops and output flip-flops is connected to the reset row, so that the initial values of the state variables and outputs are inserted when the reset row is activated. The input line $\bar{R}_\alpha$ to the flip-flop F_α is also connected to the reset row. When the reset input is high, the reset row is activated, and because $\bar{Q}_\alpha = 1$, all other rows are made low, so that the initial conditions and outputs will be stored in the flip-flops in accordance with their connections to the reset row. (In Fig. 3.20 the initial state-variable values and outputs will all be zero.) The row to set the machine, which will be called the set row, is connected to $\bar{S}_\alpha$ and Q_α of the flip-flop F_α . The set row is also connected to the reset input and the clock-pulse input. When both reset input and clock pulse are low (note that $Q_\alpha(\bar{Q}_\alpha)$ became low (high) when the circuit was first reset), then the set row will be activated. Thus $Q_\alpha(\bar{Q}_\alpha)$ is changed to high (low), which will cause the set row to go back to low, but $Q_\alpha(\bar{Q}_\alpha)$ still remains high (low). This makes the circuit ready for state transitions, because all rows for state transitions are connected to $\bar{Q}_\alpha$.

Each row for implementing a state transition is connected to the state variables q_i^* at the outputs of the state flip-flops and the input variables Y_i^* applied from outside the array. The input lines $\bar{S}_i, \bar{R}_i$ to the state flip-flops and the input lines to the output flip-flops are connected to the appropriate rows. All rows for state transitions are connected to the negated clock-pulse input from the outside, so that a row (no row if the present state is expected to be unchanged) is activated for a state transition only when the present state and inputs are appropriate and the clock pulse becomes high.

All rows for state transitions are also connected to the output line $\bar{Q}_\alpha$ from flip-flop F_α , which serves to protect the circuit from improper clock-pulse length. The input line $\bar{R}_\alpha$ to flip-flop F_α is also connected to each of these rows. If one of the rows for state transitions is then activated when the clock pulse is high, the corresponding next state variables q_i 's and outputs Z_i 's will be stored in the state and output flip-flops, and at the same time $Q_\alpha(\bar{Q}_\alpha)$ from the flip-flop F_α will become low (high). This value of $Q_\alpha(\bar{Q}_\alpha)$, which is not changed unless the clock pulse becomes low, will make all rows for state transitions low. In other words, more than one state transition for one clock pulse is not allowed, even if the length of a clock pulse is excessive. (This scheme achieves the effect of edge triggering.)

As a simple example, consider machine M_1 with the flow table shown in Fig. 3.21. It has four states, one output, one input, and seven state transitions, so that there will be seven cells ($1 + 1 + 3 + \log_2 4 = 7$). The cell for the input, the one for the reset input, and the one for the clock-pulse input do not have feedback loops. Nine rows (a reset row, a set row, and seven rows for state transitions) in the SLA are shown in Fig. 3.22. When the reset input is made high, row r_1 is activated, so that $Q_\alpha(\bar{Q}_\alpha)$ will become low (high), and both state variables q_1 and q_2 will become low. This represents the initial state A, and the output Z of the initial state A will be low. If next the reset input is changed to low and the clock pulse is made low, then row r_2 will be activated, and so $Q_\alpha(\bar{Q}_\alpha)$ will be changed to high (low), but state variables q_1 , q_2 and the output Z are unchanged. There are two rows, r_3 and r_4 , which recognize the initial value of the state variables ($q_1 q_2 = 00$), so that if the input Y is made low and the clock pulse goes high, then row r_3 will be activated and so q_1 , q_2 , and Z become, respectively, low, high, and low, which represents the next state B ($q_1 q_2 = 01$) and its output ($Z = 0$). Similarly, when the next clock pulse occurs, one of the two rows r_5 and r_6 will be activated for the corresponding state transition and output.

q_1q_2		$Y = 0$	$Y = 1$	Output
00	A	B	C	0
01	B	C	D	0
10	C	D	C	1
11	D	A	B	0

Fig. 3.21 Flow Table of M_1

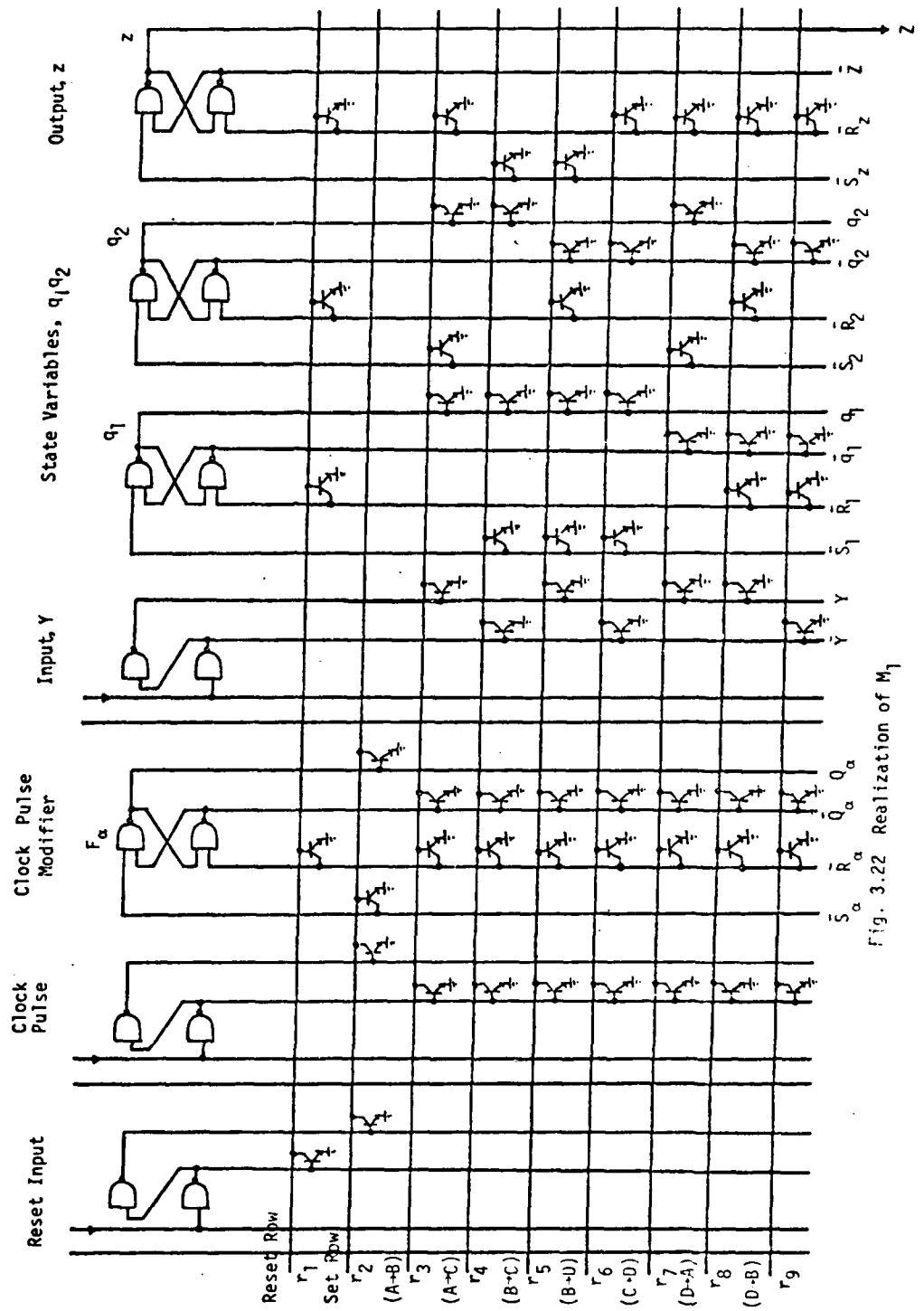


Fig. 3.22 Realization of M_1

3.31 Effects of Faults

3.311 Stuck Lines

Refer to Fig. 3.20. Input line $\bar{S}$ to the flip-flop stuck at 1(0) will cause the output Q to be reset (or it becomes unpredictable, as explained later in case C) if the other, $\bar{R}$, is nominally low. Otherwise (i.e., $\bar{R}$ nominally high), the output Q will be unchanged (set) due to the fault. Similarly, input line $\bar{R}$ stuck at 1(0) will cause the output Q to be set (unpredictable) if $\bar{S}$ is nominally low. Otherwise, the fault will cause Q to be unchanged (reset). Thus the fault may result in the incorrect next state, but it will leave the outputs correct if the flip-flop F is one of the state flip-flops. The possible malfunction will be called an incorrect state transition (IST). If the flip-flop involved is one of the output flip-flops, an incorrect output, denoted I_O , may occur. Except for case A, if the input line $\bar{R}_\alpha$ to the flip-flop F_α for the clock-pulse modifier is stuck at 1(0), then the circuit will no longer be synchronous, unless the fault is redundant, so that the next state and the outputs may be incorrect, which will also be called an incorrect state transition (IST). The input line $\bar{S}_\alpha$ stuck at 1(0) (except for case C) will make all the rows connected to $\bar{Q}_\alpha$ low, so that no state transitions (NST) will occur.

The output line $Q_j^*(Q_\alpha^*)$ from a state flip-flop F_j (the flip-flop F_α) stuck-at-1 keeps all the rows connected to $Q_j^*(Q_\alpha^*)$ from being activated, so that the corresponding state transitions cannot occur. This will result in NST. If the output line $Q_j^*(Q_\alpha^*)$ is stuck at 0, then a row connected to $Q_j^*(Q_\alpha^*)$ may be activated for a state transition when it should not be, so that the next state and the output may be incorrect. Hence we have an IST. If the output line Q_k^* from the output flip-flop F_k is stuck at 1(0), then it will clearly cause the I_O .

An input line Y_i^* stuck-at-1 keeps all the rows connected to Y_i^* from being activated, i.e., it results in NST. If Y_i^* is stuck at 0,

then one of the rows connected to Y_i^* may be activated when it should not be, so that it will result in an IST.

The reset-input line stuck at 1(0) will keep the circuit from being set (reset), so that this will result in NST (IST).

The clock-pulse input line stuck-at-1 keeps all rows connected to it from being activated, which results in NST, while the clock-pulse input line stuck-at-0 causes the circuit to be no longer synchronous, so that it could result in an IST. It is easy to see that row r_k stuck at 1(0) will cause an IST (NST).

Case C

If both inputs $\bar{S}$ and $\bar{R}$ to the flip-flop F become low due to the fault, then the outputs Q^+ and $\bar{Q}^+$ from F will be unpredictable.

3.312 Single or Multiple Missing (Extra) Devices at the Crosspoints

If a missing device disconnects the input line $\bar{S}(\bar{R})$ from the row r_k , then the output Q^+ from the flip-flop F remains unchanged when it should be changed. Thus the fault will cause an IST, but the outputs will be correct, if the flip-flop F is one of the state flip-flops, or an IO will result if the flip-flop F is one of the output flip-flops. A missing device in the output line Q_i^* causes the corresponding row to be activated when it should not be activated, so that the next state and the outputs may be incorrect; hence there results in an IST. Similarly, multiple missing devices will cause IST's and/or IO's.

An extra device in the input line $\bar{S}(\bar{R})$, except for case C, may cause the output Q to be different from the nominal value, so that the next state may be incorrect, hence the result is an IST. If an extra device connects the output line Q_i^* to row r_k , then row r_k will not be activated when it should be, so that the present state and the outputs will be unchanged, hence the result is NST. Similarly, multiple extra devices will cause IST's and/or NST's.

A missing device in the input line Y_i^* which is nominally connected to row r_k causes the row r_k to be activated when row r_k should not be,

which results in an IST. If an extra device connects the input line Y_i^* to row r_k , then row r_k will not be activated due to the fault when it should be, so the result is NST. It is not difficult to see that multiple missing (extra) devices in the input lines will cause IST's and/or NST's.

Similarly, one or more missing (extra) devices in the reset-input line and the clock-pulse line will cause IST's and/or NST's.

3.313 Shorts

Here, as before, we assume that a short between lines makes the lower value dominate. In the SLA, if two rows, r_m and r_n , are shorted, then they will always remain low (i.e., be never activated) because at least one of any pair of rows is always low. The result of the short is NST.

Except for case C explained previously, consider a short between the input line $\bar{S}_i(\bar{R}_i)$ and the output line Q_j^* from flip-flop F_j . The input line $\bar{S}_i(\bar{R}_i)$ becomes high when the clock pulse is low. If the output line Q_j^* is nominally low, then the short between Q_j^* and $\bar{S}_i(\bar{R}_i)$ will make $\bar{S}_i(\bar{R}_i)$ low, so that the output Q_i^* from the flip-flop F_i will become 1(0). This may cause an IST if F_i is one of the state flip-flops, an IO if F_i is one of the output flip-flops, and an IST or NST if F_i is the clock-pulse modifier.

A short between $\bar{S}_i$ and $\bar{Q}_i$ (or $\bar{R}_i$ and Q_i) will not affect the normal values of Q_i and $\bar{Q}_i$. If there is a short between $\bar{S}_i$ and Q_i (or $\bar{R}_i$ and $\bar{Q}_i$), then the value of $Q_i(\bar{Q}_i)$ may oscillate between 1 and 0, so that the fault is not easily modelled.

Given the input Y_j^* low, the short between $\bar{S}_i(\bar{R}_i)$ and Y_j^* will make $\bar{S}_i(\bar{R}_i)$ low when $\bar{S}_i(\bar{R}_i)$ should be high. Similarly to the short between Q_j^* and $\bar{S}_i(\bar{R}_i)$, this will cause one of the following: an IST, an IO, or NST, depending on what the flip-flop F_i is used for. It is not difficult to realize that a short between two columns other than those considered above will cause either an IST, an IO, or NST.

Next we consider shorts between a row and a column. When the clock pulse is low, all rows in the array become low, and so all input lines $\bar{S}_i(\bar{R}_i)$ become high. The short between row r_k and the input line $\bar{S}_i(\bar{R}_i)$ will cause $\bar{S}_i(\bar{R}_i)$ to be low, so that, as explained in the previous case, this will cause either an IST, an IO, or NST. Consider a short between row r_k and the output line Q_i^* from the flip-flop. If r_k and Q_i^* are nominally high and low, respectively, then the short will cause r_k to be low. In other words, r_k will not be activated, hence NST. If r_k and Q_i^* are nominally low and high, respectively, then Q_i^* will become low due to the short, so that it will cause an IST (an IO) if the flip-flop F_i is one of the state (output) flip-flops. Similarly, if the input line Y_i^* and the row r_k are nominally low (high) and high (low), respectively, then the short between Y_i^* and r_k will cause NST (an IST). It is easy to see that a short between a row and the reset-input line (the clock-pulse input line) will cause either NST or an IST.

To summarize, our exhaustive examination has shown that the effect of faults in the SLA will be one of the following:

1. No State Transition (NST)
2. Incorrect State Transition (IST)
3. Incorrect Output (IO)

3.32 Change in the Number of States Due to Faults

In preparation for later discussion of the proper method of fault-detection in an SLA, we examine first the effect of faults on the number of states.

3.321 NST's

Since the fault does not allow the present state S_i to be changed to the next state S_j , the total number of states in the machine may be decreased. In other words, once the machine reaches the state S_i , it cannot reach the state S_j , so that the state S_j will no longer occur if S_i is the only state through which the machine can reach S_j .

For example, assume that q_1 is stuck-at-1 in Fig. 3.22. Then rows r_3 , r_4 , r_5 , and r_6 will never be activated, which will keep state transitions $(A \rightarrow B)$, $(A \rightarrow C)$, $(B \rightarrow C)$, and $(B \rightarrow D)$ from occurring. Thus, if the initial state is A, then the machine will always remain in state A, so that it ceases being an FSM.

3.322 IO's

In the SLA as described, when a clock pulse occurs, the next state and the new output are uniquely determined by the present state and the inputs. If there are faults in one or more of the output flip-flops, then these faults may cause incorrect outputs, and furthermore, some of the faults may make the outputs of the next state no longer depend only on the present state and the inputs. The fault that makes an output constant reduces the number of states. On the other hand, if a fault makes the outputs of the next state depend on more than the present state and the inputs, as will be explained below, then the number of states will be increased over the nominal number.

Theorem 3.1. In a Moore-machine implemented by means of an SLA, the number of states will be increased due to some faults if the machine has at least two distinct states, S_i and S_j , the outputs of which are different, and one or more of the next states (successors) of one state, S_i , is the same as one or more of the next states of the other state, S_j .

Proof of Theorem 3.1

For simplicity, assume that only one output flip-flop, F_z , is used in the SLA. Either $\bar{S}_z$ or $\bar{R}_z$ is nominally connected to each row for implementing a state transition, so that, when the row is activated, the proper output will be stored at the output line Q_z . Consider three rows, r_i , r_j , and r_k which implement the state transitions, respectively, $(S_i \rightarrow S_k)$, $(S_j \rightarrow S_k)$, and $(S_k \rightarrow S_n)$, and let the outputs of states S_i and S_j be respectively, 0 and 1. If the fault (due to missing devices) disconnects the three rows from the input lines $\bar{S}_z$ and $\bar{R}_z$, then the output

of state S_k will be the output of the previous state S_i when row r_i is activated, but it will be the output of the previous state S_j when row r_j is activated. Thus the state S_k will have two different outputs, 0 and 1, associated with it. When the row r_k is activated, the output of state S_n will remain unchanged and so it will be the output of the previous state S_k , which has two different outputs depending on the state previous to S_k . Since the output of the next state is not uniquely determined by the present state and the inputs, one or more extra states have been effectively added. Q.E.D.

For example, consider machine M_1 , which is strongly connected. Refer to Figs. 3.21 and 3.22. Since there are two states B and C satisfying the condition assumed in Theorem 3.1, the number of states could be increased due to some faults. If the two devices at the cross-points between r_7 and $\bar{R}_2$ and between r_8 and $\bar{R}_2$ are missing, then the output of state D will be 1 when row r_7 is activated, but it will be 0 when row r_6 is activated, so that the output of state A when row r_8 is activated will not be uniquely determined, because the output of state A is the output of the previous state D. This is shown in the flow-table of Fig. 3.23. The state D has two different outputs, so that by adding one extra state, E, we can complete the flow table of the faulty machine, which is shown in Fig. 3.24. (This is a Mealy-machine)

As a second example, consider machine M_2 which is not strongly connected, but satisfies the condition assumed in Theorem 3.1. Its flow table is shown in Fig. 3.25. States A and B have different outputs and the same successor D. Assuming that the row for the state-transition ($A \rightarrow B$) is represented by r_{AB} , if all devices between $\bar{S}_2(\bar{R}_2)$ and rows r_{AD} , r_{BD} , and r_{DA} are missing, then the flow table will be changed to that shown in Figs. 3.26 through 3.28. The two different possible output values of state D will cause extra states A_1 , D_1 to be added as shown in Fig. 3.27. The resulting Mealy machine, M_2^e , can be transformed to its Moore equivalent, M_2^o , as shown in Fig. 3.28.

$q_1 q_2$		$Y = 0$	$Y = 1$
00	A	B, 0	C, 1
01	B	C, 1	D, 0
10	C	D, 1	C, 1
11	D	A, 	B, 0

the output of the state D

Fig. 3.23 Faulty Version of M_1

	$Y = 0$	$Y = 1$
A	B, 0	C, 1
B	C, 1	D, 0
C	E, 1	C, 1
D	A, 0	B, 0
E	A, 1	B, 0

Fig. 3.24 Result of Fault in M_1

	Y = 0	Y = 1	Output
A	B	D	0
B	D	C	1
C	C	C	0
D	D	A	0

Fig. 3.25 Machine M_2

	Y = 0	Y = 1	
A	B, 1	D, 0	← the output of state A
B	D, 1	C, 0	← the output of state B
C	C, 0	C, 0	
D	D, 0	A, 0	

the output of the previous entry leading to D

Fig. 3.26 Machine M_2 with Faults

	Y = 0	Y = 1
A	B,1	D,0
B	D ₁ ,1	C,0
C	C,0	C,0
D	D,0	A,0
D ₁	D ₁ ,1	A ₁ ,1
A ₁	B,1	D ₁ ,1

Fig. 3.27 Resulting Mealy Machine M_2^e

	Y = 0	Y = 1	Output
A	B	D	0
B	E	C	1
C	C	C	0
D	D	A	0
E	E	F	1
F	B	E	1

Fig. 3.28 Resulting Moore Machine M_2^o

3.323 IST's

If there is an IST, the number of states will never be increased over the nominal number. This is so because once the state, whether correct or not, and the inputs are given, the output flip-flops will generate a unique output pattern based on that state and the inputs. For example, if the input line Y is stuck at 0 in Fig. 3.22, then rows r_3 , r_5 , r_7 , and r_8 become independent of the Y-value, so that both rows r_3 and r_4 will be activated when only row r_4 should be, and similarly both r_5 and r_6 rather than only r_6 will be activated, both r_8 and r_9 instead of only r_9 , and row r_7 will be activated when it should not be. The flow table will be changed to that shown in Fig. 3.29. Note that the number of states is unchanged.

3.33 Modification of the Circuits

As explained in Section 3.322, some faults in the output flip-flops may cause the number of states to be increased over the nominal number. This effect makes it difficult to apply conventional testing methods for sequential circuits [3.11], [3.12] to an SLA because these methods are successful in circuits where faults cannot increase the number of states. We propose here that SLA circuits be modified so that faults cannot increase the number of states. It will be shown that this can be achieved by generating the outputs by means of combinational logic circuits rather than output flip-flops.

The added combinational logic circuits consist of extra rows and cells with feedback loops that are all broken. See Fig. 3.31, where there is one output cell at the right. (The number of output cells will be $\lceil \frac{k}{2} \rceil$ if the number of output flip-flops that they replace in the original SLA is k.) Each extra row is connected to the proper state variables q_i^* to satisfy the outputs of all the states of the Moore machine and then each input line to the output cells is connected to one or more extra rows to generate the outputs of the machine. (See

	Y = 0	Y = 1
A	B, 0	D, x
B	C, 1	C, x
C	D, 0	D, 0
D	A, 0	A, 0

x : Unpredictable output

Fig. 3.29 Machine M_1 with Faults

$q_1 q_2$		Y = 0	Y = 1	Outputs ($Z_1 Z_2$)
00	A	B	C	00
01	B	C	A	10
10	C	A	D	11
11	D	D	B	01

Fig. 3.30 Machine M_3 .

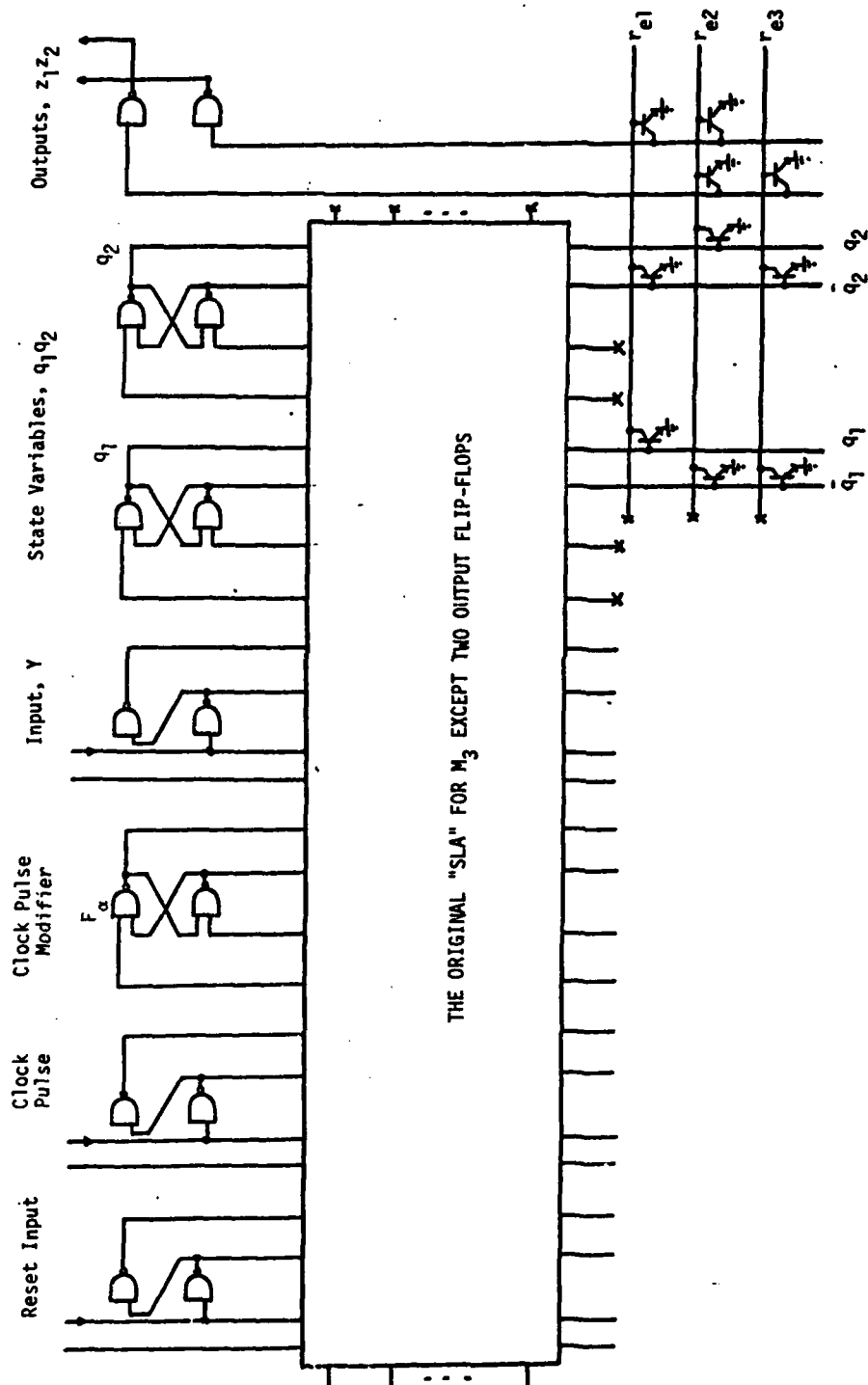


Fig. 3.31 Modified SLA for M_3

the example below.) It is easy to see that the structure of the combinational circuits described above is exactly that of the two-level PLA, where the inputs to the AND array are the state variables q_i^* and the outputs from the OR array are the outputs from the cells. Since no fault in the PLA can cause the number of states to be increased (i.e., a PLA is not changed to a sequential circuit due to faults as discussed in Section 3.1), no fault in the modified SLA will cause the number of states to be increased over the nominal number. Thus conventional testing methods can be applied to the modified SLA.

For example, consider machine M_3 with the flow table shown in Fig. 3.30. The combinational circuits for the outputs ($Z_1 Z_2$) in the modified SLA consist of three extra rows (r_{e1} , r_{e2} , and r_{e3}) and one cell. The latter has all feedback loops broken, so that the two output lines from that cell will carry outputs Z_1 and Z_2 . (Note that two flip-flop cells are required for two outputs Z_1 and Z_2 in the original SLA.) As shown in Fig. 3.31, the products realized on the extra rows r_{e1} , r_{e2} , and r_{e3} are, respectively, $P_{e1} = \bar{q}_1 q_2$, $P_{e2} = q_1 \bar{q}_2$, and $P_{e3} = q_1 q_2$. The outputs Z_1 and Z_2 from the cell will be $Z_1 = P_{e1} + P_{e2}$ and $Z_2 = P_{e2} + P_{e3}$.

3.34 Testing Methods for Sequential Circuits

We will say that a FSM is diagnosible if it has a distinguishing sequence. Techniques for making FSM's diagnosible will first be briefly reviewed and then a novel technique, more appropriate for SLA application, will be given.

3.341 C. R. Kime's Technique [3.12]

Consider the flow-table of a Mealy machine. We shall say that the machine M_1 contains the machine M_2 if deleting some of M_1 's columns creates M_2 , which has no equivalent states. If M_2 is diagnosible, then M_1 will also be diagnosible. If M_1 does not contain such a machine, Kime suggested appending to it a single column which is an irreducible machine M_2 that has a distinguishing sequence. Adding this column by

means of one extra input symbol will make any machine diagnosable. As Ref. [3.12] summarizes it "The column Kime adds is a "divide-by-two column." In other words, state S_i , with binary assignment b_i , maps to the state with assignment $\left\lfloor \frac{1}{2} b_i \right\rfloor$ ($\lfloor \cdot \rfloor$ signifies integer part of $\cdot$). The output for the state S_i is the rightmost bit of its state assignment (e.g., for the assignment 01, the output would be 1)."

For an example of this procedure, Ref. [3.12] gives the machine M_4 shown in Fig. 3.32. The divide-by-two column is added, resulting in the machine shown in Fig. 3.33.

The effect of the added column is to shift the state assignment one digit to the right and introduce a zero as the new leftmost digit. It follows that the added column has a distinguishing sequence of length k when there are k bits in the state assignment. In the four-state example, the distinguishing sequence will generate an output sequence which consists of the state variables q_2 and q_1 of each state.

3.342 R. L. Martin's Technique [3.12]

Martin's technique for making FSM's diagnosable is based on "feedback-shift-register (FSR) realization" of the machine. Fig. 3.34 shows the typical FSR circuit, where the state variable q_i of the next state in the machine is the variable q_{i-1} of the present state. If a machine is modified for FSR realization, then it has a distinguishing sequence of length k when there are k bits in the state assignment. As described in [3.12], "We propose adding a cycle of length 2^k column with outputs (added) so that any sequence of k inputs of this added column generates an output sequence which is the state assignment of the initial state... Further, since adding the cycle column to any SM makes it strongly connected, the rather unfortunate constraint of strongly connectedness usually assumed in diagnosing techniques can be discarded." For example, consider machine M_4 given previously in 3.341. As shown in Fig. 3.35, a distinguishing sequence of length 2 exists due to the cycle column under input 2, and the output sequence will be the state variables q_1 and q_2 of each state.

q_1q_2		0	1
00	A	A/0	C/0
01	B	A/0	D/1
10	C	B/1	A/1
11	D	B/1	C/0

Fig. 3.32 Machine M_4

q_1q_2		0	1	2
00	A	A/0	C/0	A/0
01	B	A/0	D/1	A/1
10	C	B/1	A/1	B/0
11	D	B/1	C/0	B/1

Fig. 3.33 Kime's Augmentation of M_4

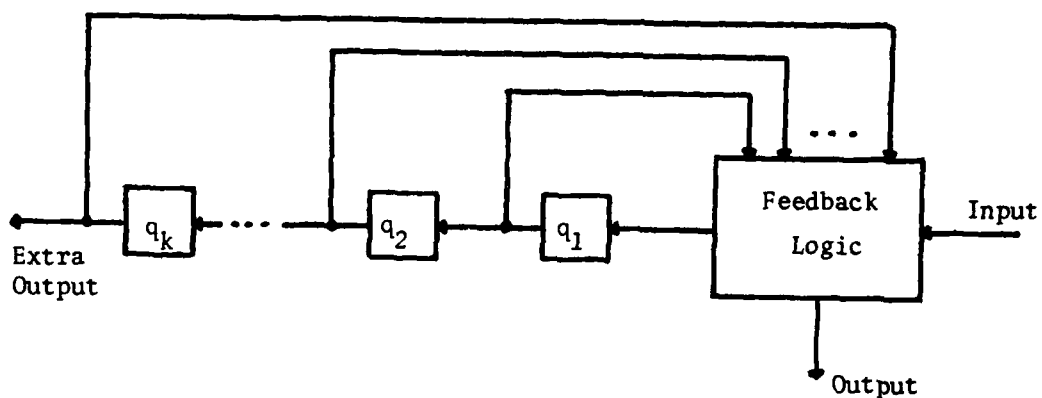


Fig. 3.34 FSR Realization, where " $q_k q_{k-1} \dots q_2 q_1$ " is the Binary State Assignment.

$q_1 q_2$		0	1	2
00	A	A/0	C/0	B/0
01	B	A/0	D/1	D/0
10	C	B/1	A/1	A/1
11	D	B/1	C/0	C/1

Fig. 3.35 Machine M_4 Modified in Accordance with Martin's Scheme

AD-A096 310

LEHIGH UNIV BETHLEHEM PA
TESTABILITY AND RELIABILITY OF LSI.(U)
JAN 81 A K SUSSKIND

F/G 9/1

F30602-78-C-0292

UNCLASSIFIED

RADC-TR-80-384

NL

2 of 2

AD A

000000



END

DATE

FILED

8-81

DTIC

3.35 Application of Testing Methods to the FSM in the SLA

Because Kime's technique for making FSM's diagnosable fails to make the machine strongly connected, it leads to a realization that it is not necessarily simple to check. Hence we prefer Martin's technique for achieving testability.

If the Moore machine contained in an SLA is directly modified in accordance with Martin's scheme, then in most cases it will become a Mealy machine due to the output requirements in the added column. Thus the number of states is likely to be increased over the original number, which is an undesirable result, when the modified machine (Mealy machine) is converted to the Moore equivalent for SLA-implementation.

An alternative approach for applying Martin's technique would be to apply conversion twice. First, the original machine is converted to its Mealy equivalent with a minimum number of states. Then after modification in accordance with Martin's scheme the machine is converted to its Moore equivalent. But still the modified Moore machine may have more states than the original Moore machine, as is illustrated in the following example.

Consider Moore machine M_5 with the flow table shown in Fig. 3.36. When this machine is converted to the Mealy machine shown in Fig. 3.37, the number of states is not changed. As a next step, we add the extra column under new input 2. If the modified Mealy machine of Fig. 3.38 is converted back to a Moore machine, then the number of states is increased by one over that of the original machine M_5 . This is shown in Fig. 3.39.

We suggest here an alternative way of adding a column which is directly applicable to Moore machines and does not increase the number of states. Assume a machine in which m states have the output 0 and n states have the output 1. If the flow table of the machine is arranged so that states having the output 0 are placed in the first m rows and the others, having the output 1, are placed in n rows following the first m rows, we get a flow table like that shown in Fig. 3.40. In the

	0	1	Output
A	B	C	0
B	C	A	0
C	D	B	1
D	A	D	0

Fig. 3.36 Machine M_5 .

	0	1
A	B/0	C/1
B	C/1	A/0
C	D/0	B/0
D	A/0	D/0

Fig. 3.37 Mealy Equivalent of M_5

q_1q_2		0	1	2
00	A	B/0	C/1	B/C
01	B	C/1	A/0	D/0
10	C	D/0	B/0	A/1
11	D	A/0	D/0	C/1

Fig. 3.38 Mealy Equivalent Modified

	0	1	2	Output
A	B	C	B	0
B	C	A	D	0
C	D	B	E	1
D	A	D	C	0
E	B	C	B	1

Fig. 3.39 Moore Equivalent Converted

		0	1	Output
m rows	S_1	.	.	0
	S_2	.	.	0
	.	.	.	.
	.	.	.	.
	.	.	.	.
	S_m	.	.	0
n rows	S_{m+1}			1
	.			.
	.			.
	.			.
	S_{m+n}			1

Fig. 3.40

	0	1	2	Output
S_1	.	.	S_2	0
S_2	.	.	S_3	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
S_m	.	.	S_{m+1}	0
S_{m+1}			S_{m+2}	1
.			.	.
.			.	.
S_{m+n}			S_1	1

Fig. 3.41

	0	1	2	Output
A	B	C	B	0
B	C	A	D	0
D	A	D	C	0
C	D	B	A	1

Fig. 3.42 Moore Machine M_5 Directly Modified

added column, the successor state to state S_i will be S_{i+1} for $i = 1, 2, \dots, m + n - 1$, and for present state S_{m+n} , the next state will be S_1 , as shown in Fig. 3.41. The modified machine will be diagnosable because the added column has a distinguishing sequence of length $m-1$ ($n-1$) if $m(n)$ is bigger than $n(m)$. The property of strongly connectedness is assured since all states are in a single cycle. As an example, we return to machine M_5 of Fig. 3.36. By adding the one column shown in Fig. 3.42, the machine has a distinguishing sequence of length 2, and the number of states is not changed.

References

- [3.1] H. Fleisher and L. I. Mason, "A contribution to array logic," IBM J. Res. Develop., pp. 98-109, Mar. 1975.
- [3.2] J. E. Lagner, et al., "Hardware implementation of a small system in programmable logic arrays," IBM J. Res. Develop., pp. 110-119, Mar. 1975.
- [3.3] E. B. Eichelberger and E. Lindbloom, "A heuristic test-pattern generator for programmable logic arrays," IBM J. Res. Develop., vol. 24, pp. 15-22, Jan. 1980.
- [3.4] C. W. Cha, "A testing strategy for PLA's," in 15th Design Auto. Conf. Proc., 1978, pp. 326-331.

- [3.5] D. L. Ostapko and S. J. Hong, "Fault analysis and test generation for programmable logic arrays," IEEE Trans. Comput., pp. 617-627, Sept. 1979.
- [3.6] J. E. Smith, "Detection of faults in programmable logic arrays," IEEE Trans. Comput., pp. 845-853, Nov. 1979.
- [3.7] D. L. Greer, "Logic design of programmable logic arrays," IEEE Trans. Comput., vol. C-28, pp. 609-617, Sept. 1979.
- [3.8] E. L. Muehldorf and T. W. Williams, "Optimized stuck fault test pattern generation for PLA macros," in Dig. Semiconductor Test Symp., Cherry Hill, N.J., Oct. 25-27, 1977, pp. 88-101, IEEE catalog no. 77ch-12f-7c.
- [3.9] D. L. Greer, "An associative logic matrix," IEEE J. Solid-State Circuits, vol. SC-11, pp. 679-691, Oct. 1976.
- [3.10] S. S. Patil and T. A. Welch, "A programmable logic approach for VSLI," IEEE Trans. Comput., vol. C-28, pp. 594-601, Sept. 1979.
- [3.11] C. R. Kime, A Failure Detection Method of Sequential Circuits, Department of Electrical Engineering, University of Iowa Technical Report, 66-130 (January 1966).
- [3.12] R. L. Martin, Studies in Feedback-Shift-Register Synthesis of Sequential Machines, Research Monograph No. 50, The M.I.T. Press, Cambridge, Mass.
- [3.13] F. C. Hennie, Finite-State Models for Logical Machines, John Wiley & Sons, Inc., New York, 1968.
- [3.14] Z. Kohavi, Switching and Finite Automata Theory, Second Edition, McGraw-Hill Book Co., New York 1970.

4.0 A TESTABILITY MEASURE

A testability measure (TM) must satisfy two major requirements: (1) it must indicate the difficulty encountered in testing and (2) the computational complexity of evaluating the TM must be significantly lower than that of actually finding the tests.

The appropriate response to the first requirement is far from clear, because the requirement is not. What do we mean by "difficulty encountered in testing"? Is it the effort required to generate (specify) tests for all stuck-at faults? for all bridging faults? Is it required that the size of the test set must not exceed some limit, so that test duration is modest and the data stored in the tester is manageable? Do we assume scan-in/scan-out (also called LSSD; see Ref. 4.1) design, so that we need to test only combinational circuits and shift registers?

If these questions are hard to answer, our quandary is made even greater when we weigh the appropriateness of the use of universal network forms for which universal tests of very modest size are known and can be written down without significant effort (see Ref. 4.2). We realize that the easily-testable network form of Ref. 4.2 has not found acceptance. On the other hand, we know that PLA's are finding wide application. We showed in Chapter 3 that simple PLA's can be very thoroughly tested by means of checking the Walsh coefficients. But the difficulty in testing PLA's with n input-signal pins by verifying Walsh coefficients is independent of the particular function on hand, since (a) the test set always consists of the set of all binary n -tuples, and (b) the test preparation consists only of calculating the appropriate Walsh coefficient of the particular (combinational) logic function on hand. So we would conclude that all PLA's with an equal number of input pins are equally testable, as far as Walsh-coefficient verification is concerned. To add to our perplexity, one might ask how he

would compare the testability of PLA's with that of j-level "random" logic.

When we considered all of these matters, we had to conclude that it would be unrealistic to expect that one can formulate a testability measure that is applicable to the entire broad range of issues raised above. Rather, we restricted ourselves to conventional testing in which the test equipment generates a specific set of input vectors (to be either stored in memory or derived by some algorithm, but not consisting of all 2^n n-tuples) and the response is compared to the nominal (fault-free) case. We were not able to come up with a means of assessing the size of the test set (i.e., its length) and therefore based our testability measure on the degree of difficulty in finding (specifying) tests for stuck-at faults.

In order to keep the task of computing the TM modest, and so satisfy the second requirement listed above, we developed approximations to the computation of the component calculations. We can compare these with precise calculations and thus we have been able to assess how well our approach does in comparison to the precise formulation. We believe that the "testability" of our TM against a rigorous measure is a valuable feature of our approach.

A second feature of our approach is that it lends itself to the "building-block" approach in those cases where the blocks do not share inputs.

4.1 Formulation of the Testability Measure

Our testability measure is derived from the number of solutions to the Boolean equation

$$x_L^* \frac{df}{dx_L} = 1 \quad (4.1)$$

Solutions to (4.1) simultaneously satisfy the pair of Boolean equations

$$x_L^* = 1 \quad (4.2)$$

$$\frac{df}{dx_L} = 1 \quad (4.3)$$

In (4.2), x_L^* denotes the signal on lead L which is to be tested for "sticking". If the test is for L-stuck-at-1(0), $x_L^* = \bar{x}_L(x_L)$. In other words, (4.2) sets the signal on lead L to logic value 0 when the lead is to be tested for stuck-at-1, and to logic value 1 when the lead is to be tested for stuck-at-0. The number of solutions to (4.2) is called the controllability of lead L.

Equation (4.3) sets the Boolean difference (BD) to logic value 1 and its solutions have the following meaning. Recall that given a function f of $x_1, x_2, \dots, x_L, \dots, x_n$ one defines (see, e.g., Ref. 4.3)

$$\frac{df}{dx_L} = f(x_1, x_2, \dots, 0, \dots, x_n) \oplus f(x_1, x_2, \dots, 1, \dots, x_n)$$

Thus the BD is the exclusive-OR of the function with $x_L = 0$ and the function with $x_L = 1$. To satisfy (4.3), the two terms on the right in the defining relation must have opposite logic values, which can happen only when the value of f with $x_L = 0$ is different from the value of f with $x_L = 1$. In other words, when (4.3) is satisfied, then the values of the independent variables x_i are such that a change in the value of the signal on lead L results in a change in the value of the output. Put still another way, satisfying (4.3) assures that conditions in the circuit are such that a change in x_L is observed at the output, and so we call the number of solutions to (4.3) the observability of lead L.

For example, let

$$f = AB \oplus \bar{A}C$$

be realized by the circuit of Fig. 4.1. Then, since

$$\frac{df}{dA} = C \oplus B = B\bar{C} + \bar{B}C = 1$$

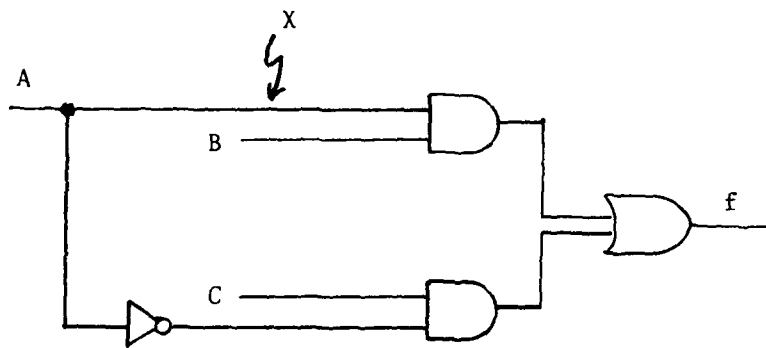


Fig. 4.1 First Example

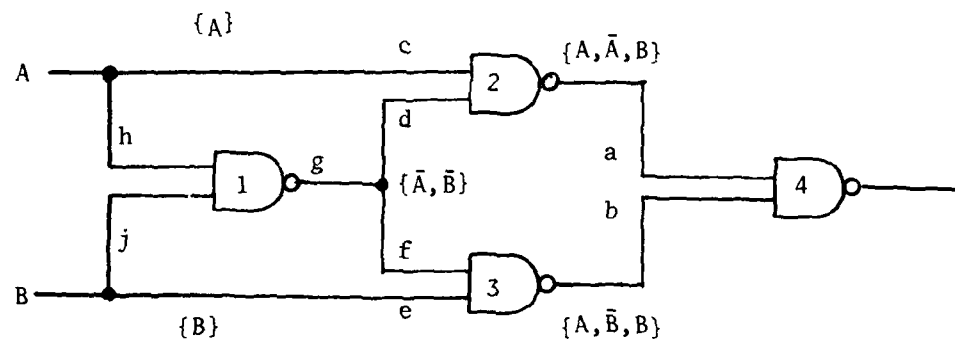


Fig. 4.2 Second Example

has two solutions ($B = 1, C = 0$ and $B = 0, C = 1$), it follows that the observability of A is 4 because there are four solutions in the unit 3-cube.

To make our observability measure independent of the number of variables, we normalize our results by dividing the number of solutions by 2^n . Thus for our example the observability of lead A is $\frac{4}{8} = \frac{1}{2}$. To assess the observability of lead X, the top fan-out of input A, we write

$$f = XB + \bar{A}C$$

from which we get

$$\frac{df}{dX} = \bar{A}C \oplus (B + \bar{A}C)$$

There are three input vertices for which $\frac{df}{dX} = 1$: $B = 1, C = 0$, and $A = B = C = 1$. Thus the observability of lead X is $\frac{3}{8}$.

The calculation of the BD, while not difficult, is tedious and if it were to be used for calculating the observability of every lead in the network, lengthy computations would be required. Moreover, getting the solutions to (4.3) is half the work of finding tests; the other half consists of finding solutions to (4.2). But keep in mind that our observability measure does not require the input conditions under which a "wobble" in lead L is observed at the output; our measure only seeks to determine the number of these conditions, and for that purpose, we develop approximations, which will be described in Section 4.4.

Just as we are not willing to go to the process of finding all the solutions to (4.3) and then counting them, so we are also not prepared to find all the solutions to (4.2) and then count them. Rather, our controllability measure $C_L^1(C_L^0)$, which measures the number of (primary) input combinations that set x_L to 1(0), is also based on approximations and these will be presented and discussed in Section 4.3. Our controllability measure is also normalized by dividing the number of solutions to (4.2) by 2^n .

Some elementary examples may help. If the function is an n -input AND of independent inputs, i.e., $f_A = x_1 x_2 \cdots x_n$, then $C_A^1 = \frac{1}{2^n}$ (because only one input combination of 2^n sets f_A to 1) and $C_A^0 = \frac{2^n - 1}{2^n} = 1 - C_A^1$. Note that it must always be true that $C^1 + C^0 = 1$.

As a second example, consider $f_B = x_1 \oplus x_2 \oplus \cdots \oplus x_n$. This is a function which has as many 1's as 0's in its truth table, so

$$C_B^1 = C_B^0 = \frac{2^{n-1}}{2^n} = \frac{1}{2}.$$

The parity function has an interesting observability feature.

$$\text{Since } \frac{df_B}{dx_i} = (x_1 \oplus x_2 \oplus \cdots \oplus x_{i-1} \oplus x_{i+1} \oplus \cdots \oplus x_n) \oplus (x_1 \oplus x_2 \oplus \cdots \oplus x_{i-1} \oplus 1 \oplus x_{i+1} \oplus \cdots \oplus x_n)$$

and $A \oplus A = 0$, it follows that $\frac{df_B}{dx_i} = 1$, no matter what the other

input values are. In other words, a change in x_i can be observed at the output no matter what. That is reflected in our observability measure, which takes on the value $\frac{2^n}{2^n} = 1$, which is the largest value that it could assume. It is gratifying to know that our observability measure reflects properly the ease with which signal variations are transmitted through an exclusive-OR box.

4.2 The Basic Strategies

To calculate the observability as well as the controllability we will not work with Boolean functions, as we did in the example of Fig. 4.1, but rather with a gate model of the network. This is certainly desirable, since we know that testability depends not only on the function realized, but also on the particular way in which it is mechanized. (The classical illustration of the effect of network make-up is the parity function. If realized as a tree (pyramid) of two-input exclusive-OR gates, it can be tested in four steps for all

single faults, no matter what the number of inputs, n , is. If realized as a two-level NAND/NAND (AND/OR) network, however, all of the 2^n possible inputs are needed for single-fault detection.)

We will work our way through the network twice. The first time, we proceed from input to output, in order to calculate the controllability values of each lead. To do this, we proceed level by level. In the first level are all those gates that have only independent (primary) inputs connected to them. In the second level are all those gates that have at least one input from gates in level 1 and possibly primary inputs. Similarly, in the i -th level are all those gates that have at least one input from gates in level $(i-1)$ and possibly inputs from gates on lower levels and perhaps primary inputs. By calculating in increasing order of level, we are able to express the controllability values of all the leads in terms of the controllabilities of the primary inputs only.

When the network is processed for the second time, we proceed from output to input, i.e., in decreasing order of the levels. This will allow us to express the observability of each lead in terms of the observability of the output and the controllability of the various leads which, as was pointed out above, has previously been expressed in terms of the controllabilities of the primary inputs. Hence observability is expressed in terms of the observability of the output and the controllability of the inputs.

As we work our way through the network from input to output, we will set up lead variable lists. These are sets which have as elements the independent variables. The lead variable list of lead L contains independent input variable x_i^* if the logic function that expresses the signal on L depends on x_i^* . If the inversion parity on the path between the primary input x_i and lead L is even (i.e., there is an even number of inverters on the path), $x_i^* = x_i$; if the inversion parity is odd, $x_i^* = \bar{x}_i$. When x_i has at least one path to L with even inversion parity and another with odd inversion parity, then the lead list contains both x_i and $\bar{x}_i$.

4.3 Controllability Calculations

4.31 AND Gates

Consider first a two-input AND gate with inputs 1 and 2 that have associated lead-variable lists L_1 and L_2 , respectively.

A. If L_1 and L_2 have no common elements,

$$C_{AND}^1 = C_1^1 \cdot C_2^1 \quad (4.4)$$

i.e., the 1-controllability of the output of the AND gate is the product of the 1-controllabilities of each input. This is certainly a very reasonable, in fact precise, rule. Because of our normalization, it says that if inputs 1 and 2 are independent, the probability of a 1 output is the product of the probability of a one on input 1 and the probability of a one on input 2. Thus (4.4) corresponds to the joint probability of two disjoint events.

B. If the lead-variable set of one input is a subset, not necessarily proper, of the other lead-variable set,

$$C_{AND}^1 = \min (C_1^1, C_2^1) \quad (4.5)$$

When the containment condition holds, it is quite likely that a 1 on one lead is accompanied by a 1 on the other, but not conversely. But for a 1 output both inputs must be 1. Hence our rule.

C. If the elements in list L_1 are the complements of those in L_2 ,

$$C_{AND}^0 = C_1^0 + C_2^0 \quad (4.6)$$

Under the given condition, if one input is 0, the other is likely to be one, and so the number of situations under which there is at least one 0 input is approximated by the sum of the number of 0 inputs from each input. Here, as elsewhere, when sums are formed they are not allowed to exceed 1. Thus in our calculations $0.6 + 0.7 = 1.3 \Rightarrow 1$.

D. If the case on hand does not nearly meet Cases A, B, or C, then each of the partially applicable formulas is used and the results are averaged. We emphasize the word "nearly" and are not able to quantify it at this time. We have worked a number of examples to date where we used "nearly" to mean about two out of three, and the results of the controllability calculations have been very good. However, our experience has not been sufficiently extensive to draw definitive conclusions.

Now consider an AND gate with more than two inputs. Because of associativity, we can decompose the AND gate into a pyramid (tree) of AND gates, each with fan-in of two, and then apply repeatedly the three formulas given above. We arrange the decomposition in such a manner that as many gates as possible have their inputs such that one of the above three conditions is fully met. For example, suppose

$$L_1 = \{a, b, c\}, L_2 = \{b, \bar{c}\}, L_3 = \{\bar{a}, \bar{b}, \bar{c}\}.$$

Then we arrange the decomposition so that leads 1 and 3 are combined first, using (4.6), and then we combine $L_{1,3} = \{a, \bar{a}, b, \bar{b}, c, \bar{c}\}$ with L_2 according to (4.5). As a second example, suppose

$$L_1 = \{a, b\}, L_2 = \{a, \bar{c}\}, L_3 = \{\bar{b}, c\}$$

Here we cannot form a "pure" pair, and so we combine leads 1 and 2 according to the average of (4.4) and (4.5) to obtain the lead variable list $L_{1,2} = \{a, b, \bar{c}\}$, which then gets combined with L_3 . Two of the three elements in $L_{1,2}$ satisfy the condition of (4.6), and so we use it.

It should be clear that where the conditions given above are simultaneously met by more than one lead pair, the above can be extended in the obvious manner. Thus for L_1 and L_2 that have no elements in common, we say immediately

$$L_{1,2} = \frac{L_1 + L_2}{2} \quad (4.4A)$$

and similarly L_1 and L_3 satisfy the containment relationship

$$C_{AND}^1 = \min (C_1^1, C_2^1, C_3^1) \quad (4.5A)$$

4.32 OR Gates

The rules for calculating the output controllability of two-input OR gates immediately follow from the rules for AND gates by applying duality.

- A. If L_1 and L_2 have no common elements,

$$C_{OR}^0 = C_1^0 \cdot C_2^0 \quad (4.7)$$

- B. If one lead set contains the other,

$$C_{OR}^0 = \min(C_1^0, C_2^0)$$

- C. If the elements in list L_1 are the complements of those in L_2 ,

$$C_{OR}^1 = C_1^1 + C_2^1 \quad (4.9)$$

Calculations for gates with greater fan-in are also made by an appropriate composition in a pyramid (tree). Again, we try to pair that the conditions are met as nearly as possible.

4.33 Other Gates

We assume that the only other gates involved are inverters (for which $C_{INV OUT}^1 = C_{INV IN}^0$), NAND gates (for which $C_{NAND}^1 = C_{AND}^0$) and NOR gates (for which $C_{NOR}^1 = C_{OR}^0$). Thus (4.4) through (4.9) are a complete set of rules for calculating the controllability transfer through the gates.

4.34 Fan-Out Free Networks

In a fan-out free network, (4.4) or (4.4A) is precisely met at every AND gate, and (4.7) or its extension to larger fan-in is precisely met at every OR gate. Thus our calculations are precise, no

matter how large the network. This is at least the easy case is handled perfectly by the controllability measure.

4.4 Example of Controllability Calculation

To illustrate our method of calculating controllability values for the difficult case, we analyze the three-level network of Fig. 4.2 that mechanizes $A \oplus B$. The lead-variable sets are indicated in the figure. Assume the controllabilities of the inputs are each $\frac{1}{2}$. Then, according to (4.4),

$$C_g^0 = C_d^0 = C_f^0 = C_A^1 \cdot C_B^1 = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

In calculating the controllability of lead a, sets $\{A\}$ and $\{\bar{A}, \bar{B}\}$ are involved. Because these sets partially satisfy the conditions that make (4.4) and (4.6) applicable, we take the average of the values determined by these. The first gives

$$C_a^0 = C_c^1 \cdot C_d^1 = \frac{1}{2} \times (1 - \frac{1}{4}) = \frac{3}{8}$$

and the second

$$C_a^1 = C_c^0 + C_d^0 = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$$

so the average is

$$C_a^1 = \frac{1}{2} \left[(1 - \frac{3}{8}) + \frac{3}{4} \right] = \frac{11}{16}$$

Because the circuit is symmetrical, $C_b^1 = C_a^1$. To compute the output controllability, we take the average of the results obtained from applying (4.5) and (4.6). The former is partially applicable, because the two inputs to gate 4 have A and B in common in their lead-variable sets. Equation (4.6) is partially applicable because the complemented variable in one set appears uncomplemented in the other. According to (4.5),

$$C_{out}^0 = \min (C_a^1, C_b^1) = C_a^1 = \frac{11}{16}$$

According to (4.6),

$$C_{out}^1 = C_a^0 + C_b^0 = 2C_a^0 = 2(1 - \frac{11}{16}) = \frac{10}{16}$$

The average is $\frac{1}{2} \left[(1 - \frac{11}{16}) + \frac{10}{16} \right] = \frac{15}{32}$ and this compares favorably with the correct value of $\frac{1}{2}$.

4.5 Observability Calculation

To calculate the observability of input lead 1 of an AND (OR) gate with a fan-in of r , we consider the gate equivalent to a gate with fan-in of two in which one input is lead 1 and the other the output of a second AND (OR) gate which has a fan-in of $r-1$. We then compute the 1-controllability (0-controllability) of the output of the gate with fan-in of $r-1$ in accordance with Section 4.3. This is depicted in Fig. 4.3. Because a NAND (NOR) gate with fan-in of r is equivalent to an AND (OR) gate with fan-in of r followed by a single inverter and the observability of the input of an inverter equals that of its output, our approach to calculating the observability of an input to a NAND (NOR) gate is identical to that of an AND (OR) gate. This is also shown in Fig. 4.3.

From the simplifications depicted in Fig. 4.3, it follows that it is sufficient to describe our method of calculating the observability for AND (OR) gates with a fan-in of two. For AND gates, we calculate according to

$$OBS_{input\ 1} = (C_{input\ 2}^1)(OBS_{output}) \quad (4.10)$$

and for OR gates according to

$$OBS_{input\ 1} = (C_{input\ 2}^0)(OBS_{output}) \quad (4.11)$$

These rules are easy to justify. The "wiggle" on an input of an AND (OR) gate shows up at the output only when the other input is 1 (0).

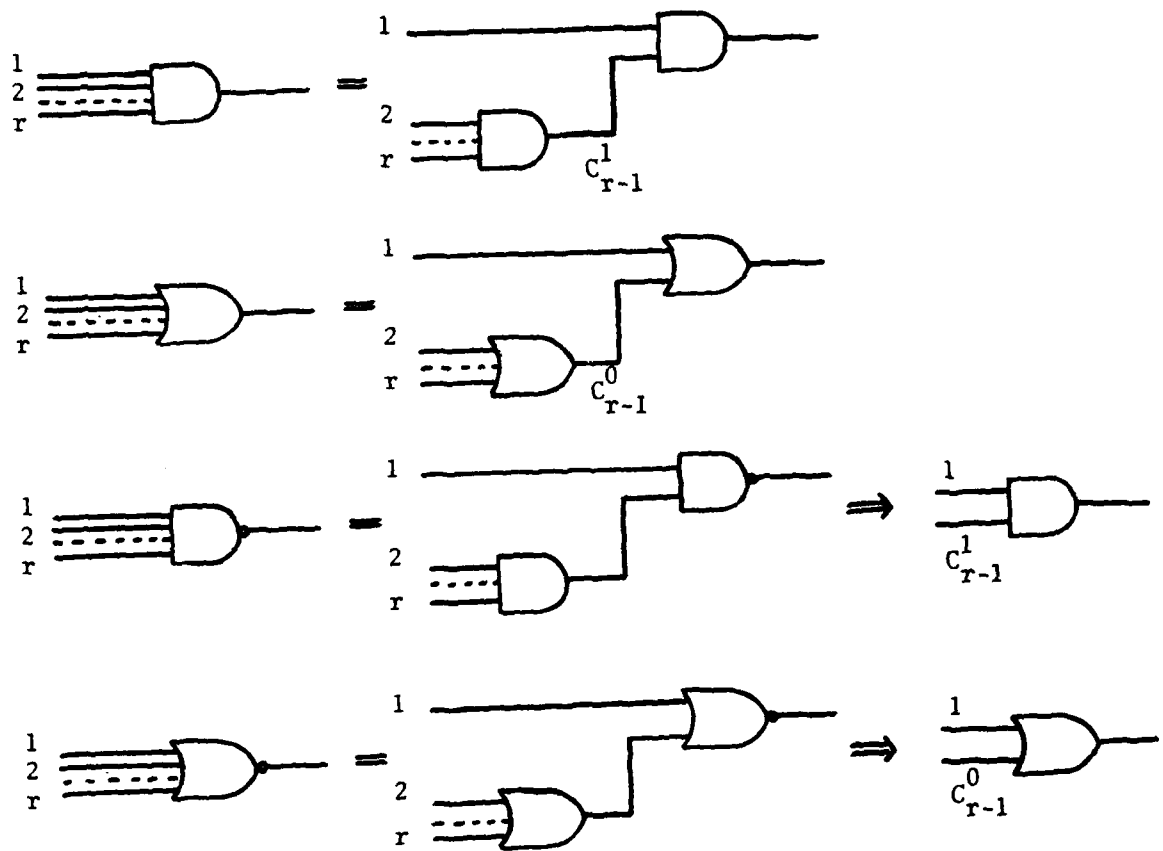


Fig. 4.3 Gate Equivalents for Observability Calculations

The observability of a network output is 1, and since we calculate observability from output toward input, (4.10) and (4.11) let us jump from gate outputs to their inputs. It is not difficult to see that if a network is fan-out free, our method of calculating observability is now completely described and leads to precise results, just as it did in the case of controllability.

If the network does have fan-out, care must be exercised in calculating the observability of the stem of the fan-out, OBS_S . Say S fans out into A and B. Our gate calculations permit us to calculate OBS_A and OBS_B . These two measures are then combined to obtain OBS_S according to which of three cases holds. Before we can treat these, however, we must define the path variable list.

The path-variable list of lead i, P_i , is the union of the lead-variable lists L_j that are associated with all the gates that lie on the path from lead i to the output. For example, in Fig. 4.2 the path-variable list P_d of lead d is $L_c \cup L_b = \{A\} \cup \{A, \bar{B}, B\} = \{A, \bar{B}, B\}$ and similarly

$$P_f = L_e \cup L_a = \{B\} \cup \{A, \bar{A}, B\} = \{A, \bar{A}, B\}$$

$$P_c = L_d \cup L_b = \{\bar{A}, \bar{B}\} \cup \{A, \bar{B}, B\} = \{A, \bar{A}, B, \bar{B}\}$$

If a lead has more than one path to the output, then its path list is the union of the path list of each path. In our example, lead g has fan-out, so

$$P_g = P_d \cup P_f = \{A, \bar{A}, B, \bar{B}\}.$$

$$P_h = L_j \cup P_g = \{A, \bar{A}, B, \bar{B}\}, \text{ so}$$

$$P_A = P_h \cup P_c = \{A, \bar{A}, B, \bar{B}\}.$$

We can now state the rules for calculating the observability of the stem of a fan-out point, OBS_S , in terms of the observabilities OBS_A and OBS_B of the two leads A and B connected to S.

A. If the inversion parity of the signal on lead A is the same as the inversion parity of the signal on lead B,

$$\text{OBS}_S = \text{OBS}_A + \text{OBS}_B \quad (4.12)$$

B. If the inversion parity of the signals does not satisfy Case A, two subcases are considered:

1. If the elements in P_A that appear in one form, x_j^* , do not appear in the opposite form, $\overline{x_j^*}$, in P_B

$$\text{OBS}_S = |\text{OBS}_A - \text{OBS}_B| \quad (4.13)$$

2. If the condition under (1) does not hold or P_A and P_B have no common elements,

$$\text{OBS}_S = \text{OBS}_A + \text{OBS}_B \quad (4.14)$$

As in the computation of the controllability measure, when the stated condition is not fully met, we take the average. This can only involve (4.13) and (4.14), since the primary condition of either A or B must be fully met. Without loss of generality, suppose $\text{OBS}_A > \text{OBS}_B$. Then (4.13) gives $\text{OBS}_A - \text{OBS}_B$, while (4.14) gives their sum, so that the average will be

$$\text{OBS}_S = \frac{\text{OBS}_A - \text{OBS}_B + \text{OBS}_A + \text{OBS}_B}{2} = \text{OBS}_A$$

Thus we use the larger of the two observability measures when Case B applies, but subcases 1 and 2 are each only partially met. This is a rather appealing feature of our computation scheme.

Keep in mind that when forming sums, we do not permit results larger than unity. Moreover, we treat a multiple fan-out as a pyramid (tree) of two-output cases, and again arrange these so that the stated conditions are fully met as often as possible.

We illustrate our method of calculating the lead observabilities by means of Fig. 4.2.

$$\text{OBS}_a = C_b^1 \cdot \text{OBS}_{\text{out}} = \frac{11}{16} \times 1 = \frac{11}{16}$$

$$\text{OBS}_c = C_d^1 \cdot \text{OBS}_a = \frac{3}{4} \times \frac{11}{16} = \frac{33}{64}$$

$$\text{OBS}_d = C_c^1 \cdot \text{OBS}_a = \frac{1}{2} \times \frac{11}{16} = \frac{11}{32}$$

$\text{OBS}_f = \text{OBS}_d$, by symmetry, and the two fan-outs from g satisfy Case A. So

$$\text{OBS}_g = \text{OBS}_d + \text{OBS}_f = 2 \times \frac{11}{32} = \frac{11}{16}$$

$$\text{OBS}_h = C_j^1 \cdot \text{OBS}_g = \frac{1}{2} \times \frac{11}{16} = \frac{11}{32} = \text{OBS}_j$$

The two fan-outs from input A satisfy CASE B.

$$P_c = \{A, \bar{A}, B, \bar{B}\}$$

$$P_h = \{A, \bar{A}, B, \bar{B}\}$$

Because $P_c = P_h$, only (4.14) is applicable

$$\text{OBS}_A = \text{OBS}_c + \text{OBS}_h = \frac{33}{64} + \frac{11}{32} = \frac{55}{64}$$

Thus our result differs from the precise answer, which was previously shown to be 1, by $\frac{9}{64}$. This is a larger error than one might like. But keep in mind that the "granularity" of our computations is 2^{-n} , where n is the number of independent variables. Here $n = 2$, so our granularity is $\frac{1}{4} = \frac{16}{64}$.

The major source of error lies in the value used for C_b^1 . Using the correct value $C_b^1 = \frac{3}{4}$, we get $\text{OBS}_a = \frac{3}{4}$, $\text{OBS}_c = \frac{3}{4} \times \frac{3}{4} = \frac{9}{16}$,

$$\text{OBS}_d = \frac{1}{2} \times \frac{3}{4} = \frac{3}{8}, \text{OBS}_g = 2 \times \frac{3}{8} = \frac{3}{4}, \text{OBS}_h = \frac{1}{2} \times \frac{3}{4} = \frac{3}{8}, \text{ and}$$

$$\text{OBS}_A = \frac{9}{16} + \frac{3}{8} = \frac{15}{16}, \text{ which misses the precise answer by only } \frac{4}{64}.$$

4.6 Some Examples

We report here on some typical results obtained in applying our testability and observability measures.

1. Two-level, two-input exclusive-OR (NAND/NAND) circuit.

Our results were error-free except for the input observability values that were off by $\frac{1}{4}$.

2. Two-level, three-input exclusive-OR (NAND/NAND) circuit.

All controllability values were precise, except for the output controllabilities, which were each calculated with an error of $\frac{1}{8}$.

Four of the observability values were off by $\frac{23}{512}$ and three by $\frac{169}{512} = 0.33$. The latter figure was obtained for the observabilities of the three independent variables.

3. Two-level, four-input exclusive-OR (NAND/NAND) circuit.

The results were similar to the three-variable case. The output controllability was off by $\frac{1}{8}$ and the input observability values were off by 36%.

4. The analysis of the circuit of Fig. 4.4 gave no error greater than nine percent. Note that this circuit has redundancy as well as fan-out. (The bottom terminal of the AND gate can be set to 1 without affecting the function realized, which is $w\bar{x} + \bar{x}\bar{y}\bar{z}$.)

In general, we found that our controllability results were very good; they were rarely above the granularity of the numbers used. The observability values, however, did at times have substantial errors, but only when there was a great deal of fan-out in the circuit. In fact, in one pathological example where there was a great deal of logic redundancy as well, the maximum error in observability was 44%. The two-level exclusive-OR circuit is also unfavorable to our approximations. A certain amount of this difficulty is to be expected, but the current rules may well need some refinement, and this should be considered in the future.

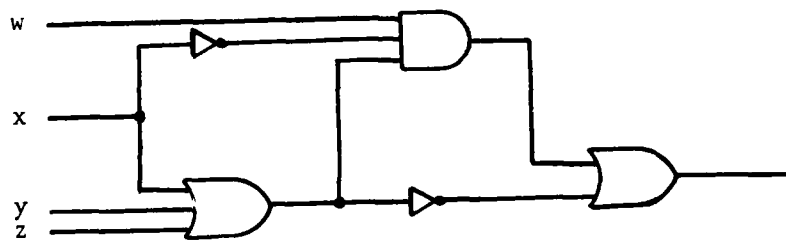


Fig. 4.4 Third Example

4.7 Testability of Sequential Circuits

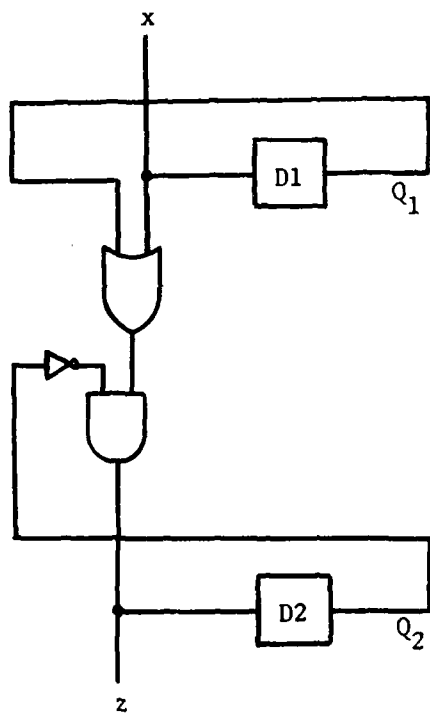
To compute the testability of a sequential circuit, we will first convert it to its iterative equivalent through the following sequence of four steps:

(1) The output leads from all the flip-flops are broken and the breaks are labeled with the initial conditions, i.e., with the initial values of the state variables. (2) All of the loop-free gate structure (hence not the flip-flops) is drawn as the typical "cell". (3) The flip-flops are placed to the right of the typical cell and their inputs (excitations) are properly connected to the typical cell. (4) The typical cell is repeated and those leads that in (2) had initial conditions on them are connected to the appropriate outputs of the appropriate flip-flops. Steps (3) and (4) are repeated until there is at least one signal path from an independent (primary) input to an observable output through each flip-flop.

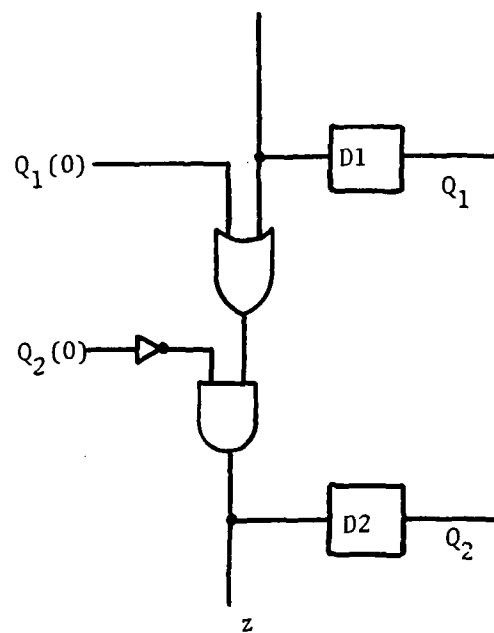
We illustrate the procedure in Fig. 4.5. The boxes marked D are clocked D flip-flops and here, as everywhere, we have omitted clock connections from the drawing. The conventionally drawn circuit is shown in (a). The result of steps (1), (2) and (3) is indicated in (b) and the completed iterative circuit is shown in (c).

As is well known, the iterative circuit is the space-sequential equivalent (without closed loops) of the time-sequential circuit (with closed loops). In the time sequential circuit, each input choice is independent of the previous one, and so in the iterative circuit the set of primary inputs to one cell is independent of that to any other cell. Also, the observability of each of the primary outputs of each cell is 1.

Now consider the 4-stage shift register of Fig. 4.6 (a), where four D-flip-flops are shown in cascade. (Implicit in our representation is that z and the contents of flip-flop 4 are always the same; they cannot differ.) This circuit can be redrawn as in (b) to emphasize the fact that the "logic" here consists of nothing but four wires (identity functions). Changes in D_1 have no opportunity to



(a)



(b)

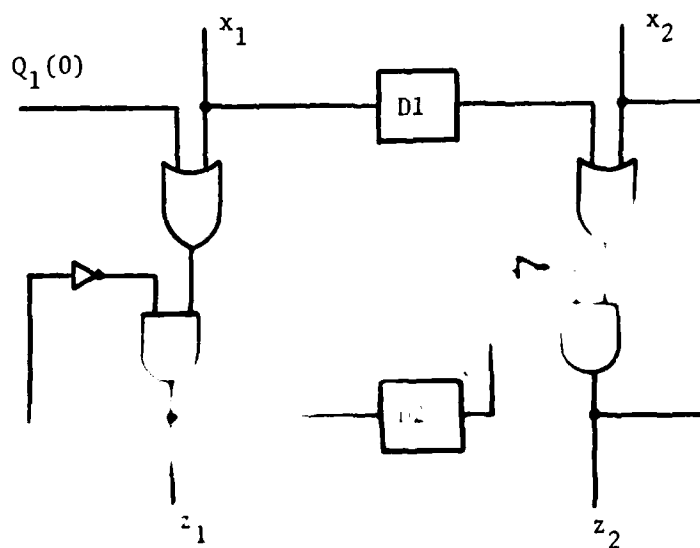


Fig. 4.5 Iterative Circuit Representation of Sequential Circuit

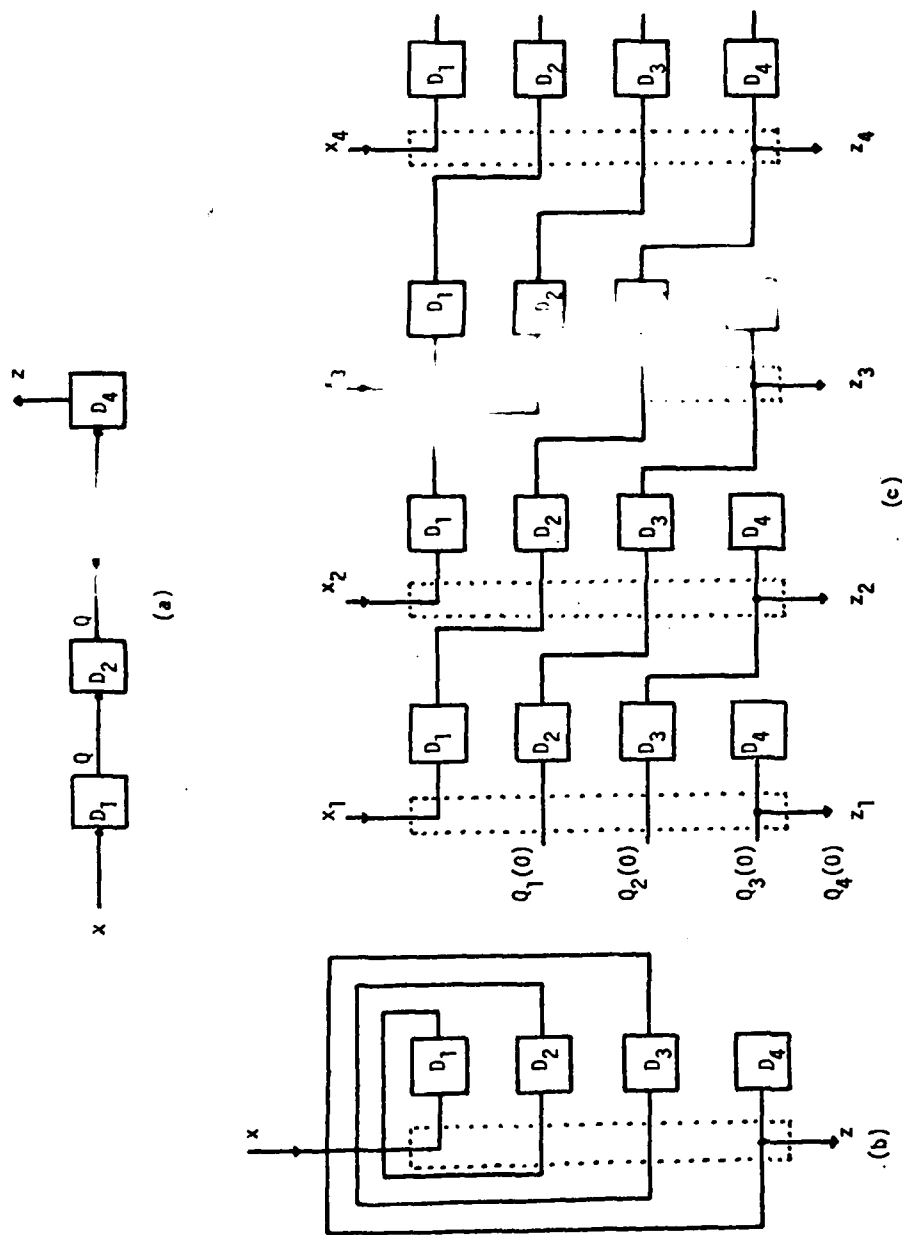


Fig. 4.6 Shift Register and its Iterative Circuit Representation

affect any output for the first three inputs and so four cells are needed. We will call the number of cells the length of the iterative circuit and will denote it by L .

The only difference between the iterative circuit and the circuits which we have analyzed previously in this chapter lies in the existence of the D flip-flops. But D flip-flops simply set the output equal to the input; they are nothing more than a delay (hence the name), or a synchronization means. When working properly, they do not affect the transfer of information, and so we will treat them as short circuits when calculating controllability and observability.

With this in mind, one finds that for every lead in the 4-bit shift register, $C^1 = C^0 = \frac{1}{2}$ (if the inputs have controllability $\frac{1}{2}$) and the observability of leads everywhere is 1. Moreover, it is readily seen that with these assumptions the observability as well as the controllability values are independent of how many cells are used, i.e., how long the shift register is. However, it is well known that the testing of a shift register depends greatly on its length and, in the general case, the testing difficulty encountered with sequential circuits depends on how deeply "buried" the flip-flops are. (A notorious case of long test sequences results from a chain of divide-by-two circuits, i.e., long counter circuits.)

The parameter L is a measure of how accessible the flip-flops are to the independent (primary) inputs and how readily they communicate with the primary (observable) outputs. At this time, we have not yet formulated an appropriate factor, dependent on L , to modify the combinational controllability and observability. Perhaps it should take the form 2^{-L} . But we feel confident that some function of L will be useful in reflecting the degree to which flip-flops are buried.

In cases where flip-flops other than type D are used, the links between the cells can indeed modify the controllability and observability values. For J-K flip-flops, for example, the controllability of the output (Q) can be calculated as follows. Consider the transition

table for a J-K flip-flop shown below:

	JK			
	00	01	11	10
0	0	0	1	1
1	1	0	0	1

The row coordinates (0,1) are the state of the flip-flop (often called Q^-) prior to the arrival of a clock pulse; the column coordinates are the values of the J and K signals, respectively, during the clock pulse; the interior entries show the resulting state of the flip-flop (often called Q^+) after the clock pulse. If P denotes the probability that $Q^- = 1$, and J^1 and K^1 the 1-controllability of the J and K terminals, respectively, then

$$\begin{aligned} Q_+^1 &= \text{1-controllability of the J-K flip-flop} \\ &= (1-P)J^1 + P(1-K^1) \end{aligned} \quad (4.15)$$

When this flip-flop occurs after Cell i, then the value of Q_+^1 of the corresponding flip-flop after Cell i-1 is used for P.

With respect to the transfer of observability through the J-K flip-flop, we suggest using

$$\text{OBS}(J) = (1-P) [\text{OBS}(Q)] \quad (4.16)$$

because when $\bar{Q} = 0$ a change in the J input is reflected in a change in the output, Q^+ , no matter what the state of K. Similarly, we propose

$$\text{OBS}(K) = P [\text{OBS}(Q)] \quad (4.17)$$

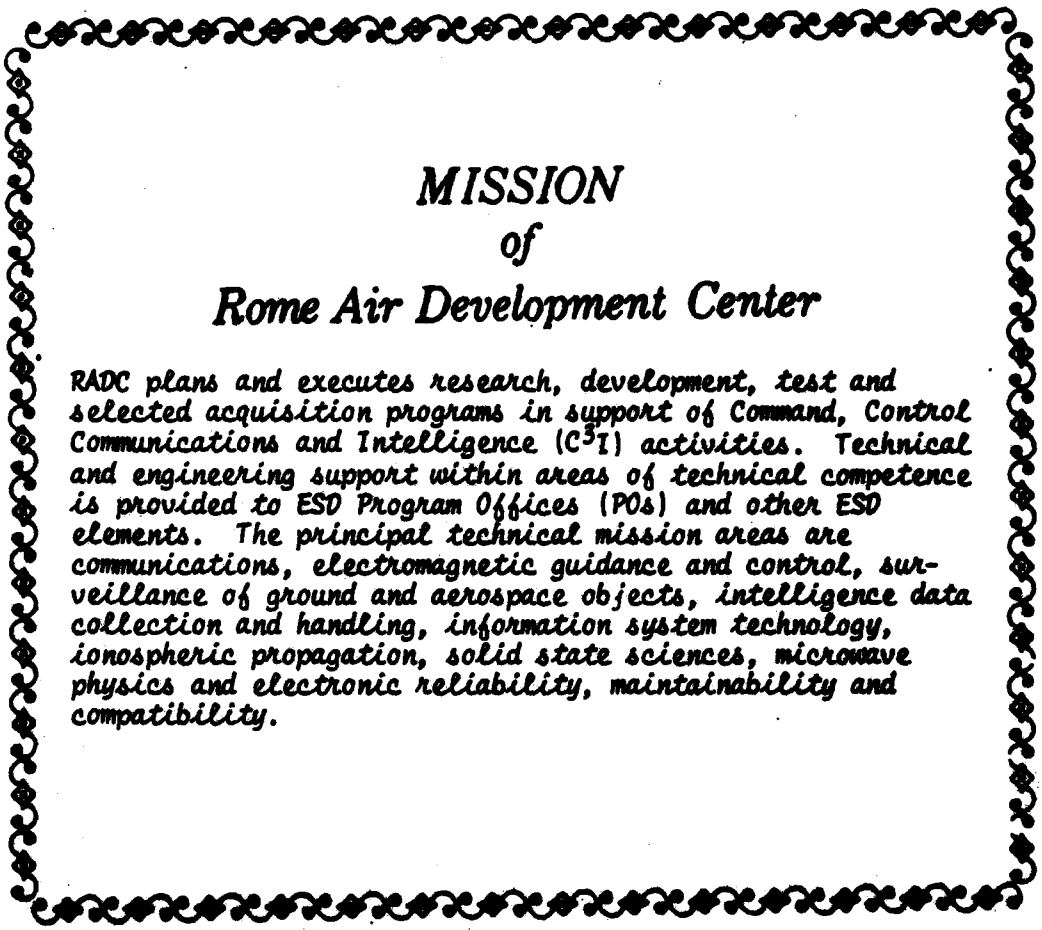
These seem reasonable formulations at this time, but we must point out that we have not yet had time to assess their usefulness.

Finally, a word of caution. Formula (4.15) assumes that the signals on leads J and K are derived from independent sets of inputs, i.e., that the expressions for J^1 and K^1 share no literals. If they do, the expression for Q_+^1 must be modified. One way to do this is based on drawing

the gate model for the J-K flip-flop, which has feedback loops, as an iterative circuit, as was done by M. Flomenhoft in Ref. 4.4. Then the analysis techniques given earlier in this chapter can be used on the iterative circuit model of the flip-flop and the solutions obtained can then be inserted in the links between the cells. This does increase the computational labor, and so one should seek some simplification, possibly through the use of a less precise flip-flop model.

References

- 4.1 E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testing," Proc. 14th Design Automation Conf., New Orleans, June 1977, pp. 462-468.
- 4.2 S. M. Reddy, "Easily Testable Realizations for Logic Functions," IEEE Trans. Comp. C-21, 11, pp. 1183-1188, Nov. 1972.
- 4.3 A. C. L. Chiang, I. S. Reed, and A. V. Banes, "Path Sensitization, Partial Boolean Difference, and Automated Fault Diagnosis," IEEE Trans. Comp., C-21, 2, pp. 189-195.
- 4.4 M. J. Flomenhoft, "Algebraic Techniques for Finding Tests for Logical Faults in Digital Circuits," Ph.D. Dissertation in Electrical Engineering, Lehigh University, 1973.



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 (C³I) 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.