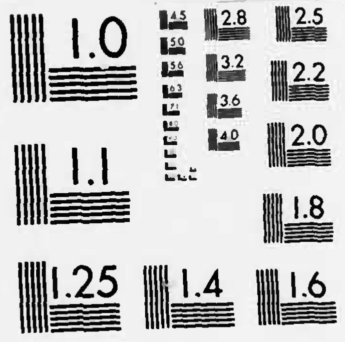
AD-A156 098 A METHODOLOGY FOR THE DESIGN OF TESTABLE CUSTOM LARGE-SCALE INTEGRATED CI..(U) AEROSPACE CORP EL SEGUNDO CA ENGINEERING GROUP 31 JAN 85 1/1

UNCLASSIFIED TR-0084A(5902-04)-1 SD-TR-85-33 F/G 9/5 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

REPORT SD-TR-85-33

AD-A156 098

A Methodology for the Design of Testable Custom Large-Scale Integrated Circuits

M. A. BREUER, CONSULTANT
Systems Support Office
Engineering Group
The Aerospace Corporation
El Segundo, California 90245

31 January 1985

Final Report

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

Prepared for

SPACE DIVISION
AIR FORCE SYSTEMS COMMAND
Los Angeles Air Force Station
P.O. Box 92960, Worldway Postal Center
Los Angeles, CA 90009-2960

DTIC
ELECTE
JUN 28 1985
S
A

85 6 17 102

This final report was submitted by The Aerospace Corporation, El Segundo, CA 90245, under Contract No. F04701-83-C-0084 with the Space Division, Deputy for Logistics and Acquisition, P.O. Box 92960, Worldway Postal Center, Los Angeles, CA 90009-2960. It was reviewed and approved for The Aerospace Corporation by C. J. Leontis, Electronics and Optics Division. The project officer was Mr. Akira Murakami, SD/ALT.

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

This technical report has been reviewed and is approved for publication. Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

FOR THE COMMANDER

[illegible]

Richard D. Benton, Major, USAF
Director, Specialty Engineering and Test

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SD-TR-85-33	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A METHODOLOGY FOR THE DESIGN OF TESTABLE CUSTOM LARGE-SCALE INTEGRATED CIRCUITS		5. TYPE OF REPORT & PERIOD COVERED Final 1 Jan 1983 - 30 July 1984 6. PERFORMING ORG. REPORT NUMBER TR-0084A(5902-04)-1
7. AUTHOR(s) M. A. Breuer, Breuer and Associates		8. CONTRACT OR GRANT NUMBER(s) F04701-83-C-0084
9. PERFORMING ORGANIZATION NAME AND ADDRESS The Aerospace Corporation El Segundo, Calif. 90245		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Space Division Air Force Systems Command Los Angeles, Calif. 90009		12. REPORT DATE 31 January 1985 13. NUMBER OF PAGES 62
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Approved for public release; distribution unlimited		
18. SUPPLEMENTARY NOTES		

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

CLSIC	Design Style
VLSI	Test Strategies
Testability	Test Strategy Measures
Test Structures	Testable Design Methodology

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

This report summarizes the main concepts in the design for testability of custom large-scale integrated circuits (CLSICs) and concepts involved in testing for physical faults in actual hardware. Important problems and issues which should be considered in designing a testable CLSIC, including test structures and design style, test strategies, test strategy measures, and testable design methodologies are introduced. A general methodology for designing a testable CLSIC is presented, which includes partitioning a chip

DD FORM 1473 (FACSIMILE)

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

19. KEY WORDS (Continued)

20. ABSTRACT (Continued)

into circuit structures, and imbedding each circuit structure into a suitable testable design structure. Measures are introduced so that different test methodologies can be quantitatively compared.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

ACKNOWLEDGMENT

The author wishes to acknowledge the many helpful discussions with M. Abadir, Xi-An Zhu, and N. Nanda in developing the concepts presented. Also acknowledged are the critical reviews and comments of C. Gwyn, L. H. Goldstein, and J. P. Hayes, the sponsorship of this work by A. Carlan, and the extensive editorial services of E. Wade and his staff.

CONTENTS

CONTENTS (Continued)

FIGURES

TABLES

1. INTRODUCTION

This report presents some of the major concepts related to the design of a testable custom large-scale integrated circuit (CLSIC). The partitioning of a CLSIC into testable circuit structures, the basic criteria and techniques used in testing, and the addition of built-in test features to facilitate testing are discussed. Built-in test features for CLSICs include the built-in test circuitry, other special built-in test structures, and the embedded firmware and software used to implement built-in testing. For example, built-in test features may include on-chip functional circuit structures, such as signature generators, comparators, parity trees, counters, encoders, and decoders; or they may be nonfunctional, such as structures used for process monitoring or to enable external testing. Nonfunctional built-in test structures are usually process-peculiar and will not be discussed in any detail.

A testable circuit structure refers to a logical organization or architecture of a CLSIC subcircuit consisting of the functional circuitry to be tested, and associated built-in test circuitry. The circuitry to be tested is called the kernel. Built-in test circuitry consists of additional circuitry, peripheral to the functional nature of the CLSIC, which is added to the chip specifically to aid in testing the functional circuitry. The built-in test circuitry may be functional in nature. Examples of testable circuit structures are level-sensitive scan designs (LSSD), built-in logic block observation designs (BILBO), and syndrome testable designs.

2. STRUCTURES AND DESIGN STYLES

Four fundamental units of logic circuitry are used to implement digital systems: busses, random access memories, registers, and combinational logic. These fundamental logic units are referred to as basic circuit structures. The simplest case of a bus is a wire, of a random access memory is a one-bit addressable storage element, of a register is a latch or flip-flop, and of a combinational logic circuit is a gate. More complicated circuitry, such as decoders and multiplexers, also are often implemented as basic structures. The interconnection of two or more of these basic structures (either different or identical units) results in a circuit structure. The difference between a basic circuit structure and a circuit structure is subtle. Arithmetic logic units, counters, and shift registers are examples of simple circuit structures. Circuit structures often have architectural styles associated with them, such as pipeline, bus-oriented, or bit-sliced.

There are numerous ways of implementing a basic structure in a single silicon chip. Circuit design considerations differ in: (a) how transistors are constructed, (b) how transistors are interconnected to form logic functions, (c) how logic functions are interconnected, and (d) what technology is used. Variations in circuit design and logic function lead to different design styles, such as read only memories (ROM), programmable logic arrays (PLA), and gate combinational networks, e.g., a NAND gate network. Hence, the use of a basic structure often defines a circuit's design style. For example, a combinational logic basic structure implementing some Boolean function, such as an arithmetic logic unit, may have as a design style ROM, PLA, or gate combinational network.

The importance of identifying design styles is that different design styles can lead to unique failure mechanisms; hence, the corresponding basic structures are often tested differently. This is not necessarily true when exhaustive testing is employed, in which case the design style is usually ignored.

As an example, consider the PLA design style. Because of the high fan-in often found in the AND array, PLAs are usually not tested very completely by random test vectors. Also, PLAs are susceptible to unique failure mechanisms, such as extra or missing crosspoint connections. Hence, a test methodology for a PLA may be quite different from that for a ROM or gate combinational network.

Often, circuit structures are specially designed to enhance testability, such as in the LSSD methodology.[1,2] In this case, a combinational logic basic structure C and a shift register structure S are interconnected to enhance the testing of C, which normally has the design style of a gate combinational network. The architecture consisting of the combination of the level-sensitive scan register connected to C is said to constitute the LSSD testable structural style; the combinational logic network C which is to be tested is the kernel of the style.

In general, a CLSIC can be partitioned into functional blocks, such as control, input/output, arithmetic logic unit, and memory. For testing purposes, a CLSIC can also be partitioned into "testable" subcircuits, each subcircuit being tested in its own unique way. These subcircuits may correspond to functional blocks. By definition, they are circuit structures. Often, one of the first steps to be taken in the design of a testable CLSIC is to partition it into subcircuits. The subcircuits, in turn, define circuit structures whose fault characteristics are well-defined and for which one or more testing strategies are known. Each such structure may be modified by the inclusion of specified built-in test circuitry in order to enhance its testability. The subcircuits so defined by the partition process need not be disjoint; in fact, they often have built-in test circuits in common.

---

[1]Eichelberger, E. B. and T. W. Williams, "A Logic Design Structure for LSI Testing," Proceedings of the 14th Design Automation Conference, June 1977, pp. 462-468.
[2]Eichelberger, E. B. and T. W. Williams, "A Logic Design Structure for LSI Testability," J. Design Automation and Fault-Tolerant Computing, Vol. 2, May 1978, pp. 165-178.

A maximal basic circuit structure is a basic structure not contained within a larger basic structure. Often, a chip is tested by identifying maximal basic circuit structures and testing them individually. If a circuit structure is not too complex, such as a counter, it can be tested as an entity. For complex circuit structures, such as a microprocessor, testing it as one entity becomes extremely complex.

3. TESTING TAXONOMY

The process of testing a circuit structure in order to detect or locate hardware faults can be carried out in one of two modes, known as external testing and self-testing. The former deals with the use of automatic test equipment to test the circuit structure; the latter relies on the chip itself to carry out the testing process. A circuit structure is often tested using precomputed test programs which are created via the process of test program generation. Two major aspects of testing, therefore, are test program generation and design for testability.

3.1 TEST PROGRAM GENERATION

The major concepts related to test program generation are: fault modeling, test generation, response evaluation, fault simulation, and fault location.

3.1.1 Fault Modeling

Fault modeling deals with the process of representing the actual physical faults in the circuit (structure) under test by some type of abstract model.[3,4] It is these modeled faults which are actually processed by most test synthesis and analysis tools. Examples of commonly used fault models are listed below:

a. Single stuck-at faults
b. Multiple stuck-at faults
c. Shorts and bridging faults

[3]Hayes, J. P., "Modeling Faults in Digital Logic Circuits," Rational Fault Analysis, R. Saeks and S. R. Liberty (eds.), Marcel Dekker, NY, 1977, pp. 78-95.
[4]Nickel, V. V., "VLSI - The Inadequacy of the Stuck at Fault Model," Proceedings IEEE International Test Conference, November 1980, pp. 378-381.

d. Functional faults
e. Coupling faults
f. Pattern-sensitive faults
g. Delay faults
h. Parametric faults
i. Nonclassical MOS faults, such as opens

3.1.2 Test Generation

Tests for a circuit can be determined in several ways.[5] The most common are listed below:

a. Manual
b. Algorithmic
c. Pseudorandom
d. Exhaustive
e. Standard test patterns

The method used to generate the test must be compatible with the level of description available for the circuit structure under consideration. For example, employing a path sensitization algorithm may require a gate level description of a circuit structure; employing a test generation algorithm for PLAs may require only the truth table of the functions being implemented; employing a functional/behavioral approach may require a high level language description of the circuit structure, such as the Instruction Set Processor (ISP) notation.

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

3.1.3 Response Evaluation

Once tests are generated, they can be translated into a test program which can then be applied either by the automatic test equipment or by built-in test features to the circuit under test. Based upon the response measured, the circuit under test can be characterized as being faulty or not. If it is faulty, diagnosis or fault location can be carried out. Methods for processing the response are listed below:

a. Direct comparison

   (1) Stored response
   (2) Gold unit (standard hardware)

b. Comparison with data compression (compact testing)

   (1) Transition counting[6]
   (2) One's counting or syndrome testing[7-9]
   (3) Signature analysis[10,11]

---

[6]Hayes, J. P., "Testing Logic Circuit by Transition Counting," Digest of Papers 5th International Symposium on Fault-Tolerant Computing, June 1975, pp. 215-222.

[7]Savir, J., "Syndrome-Testable Design of Combinational Circuits," Digest of Papers 9th International Symposium on Fault-Tolerant Computing, June 1979, pp. 137-140.

[8]Savir, J. "Syndrome-Testable Design of Combinational Circuits," IEEE Trans. on Computers, Vol. C-29, June 1980, pp. 442-451 (corrections: Nov. 1980).

[9]Savir, J., "Syndrome-Testing of 'Syndrome-Untestable' Combinational Circuits," IEEE Trans. on Computers, Vol. C-30, August 1981, pp. 606-608.

[10]"A Designer's Guide to Signature Analysis," Hewlett-Packard Application Note 222, Hewlett Packard, 5301 Stevens Creek Blvd., Santa Clara, CA 95050, April 1977.

[11]Nadig, H. J., "Signature Analysis-Concepts, Examples and Guidelines," Hewlett-Packard Journal, May 1977, pp. 15-21.

3.1.4 Fault Simulation

Normally, the fault coverage of a test can be determined by using a fault simulator.[5] Fault simulation can be carried out either in software or in hardware.

3.1.5 Fault Location

Fault location can be carried out by using either fault dictionaries,[5] diagnostic routines,[5] or effect-cause analysis.[12]

3.2 DESIGN FOR TESTABILITY

Design for testability is performed for several reasons; e.g., to reduce the complexity of test generation or to make the chip partially or fully self-testable. The complexity of test generation may be reduced by enhancing controllability and observability. The chip may be made partially or fully self-testable by employing built-in test structures or other built-in test features. The major concepts in this field fall into ad hoc design methods, structural built-in test methods, designing with easily testable components, and analysis tools.

3.2.1 Ad Hoc Design Methods

Numerous ad hoc designs for testability techniques have evolved over the years. Most have dealt with small-scale or medium-scale integrated circuits on printed circuit boards. Included in these techniques are concepts such as resettable flip-flops, test points to increase observability, logical cutting of feedback lines, and inhibiting internal clocks. Extensions to

[12]Abramovici, M. and M. Breuer, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis," IEEE Trans. on Computers, Vol. C-29, June 1980, pp. 451-460.

these early techniques have led to many of the built-in test methods currently used extensively in VLSI circuits.

Ad hoc design methods include:

a. Degating[13]
b. Addition of test points[14,15]
c. Bus architecture[13]
d. Partitioning[16]
e. Self-comparison[17]
f. Self-oscillation[17]

3.2.2 Structural Built-in Test Methods

Structural built-in test methods fall into two major categories, namely, semi built-in and fully built-in techniques. Semi built-in test methods employ hardware structures, such as set/scan registers, to increase controllability and observability. Off-line test generation is usually still required.

Both on-line and off-line fully built-in test techniques exist. The on-line methods are examples of concurrent testing. The off-line methods, such as built-in logic block observation, are gaining in popularity. These methods eliminate the need for off-line test generation and thus minimize the need for automatic test equipment. These techniques often require minor or no changes to the kernel structure being tested.

---

[13] Williams, T. W. and K. P. Parker, "Design for Testability--A Survey," Proceedings IEEE, Vol. 71, January 1983, pp. 98-112.
[14] Hayes, J. P. and A. D. Friedman, "Test Point Placement to Simplify Fault Detection," Digest of Papers 10th International Symposium on Fault-Tolerant Computing, June 1973, pp. 73-78.
[15] Hayes, J. P., "On Modifying Logic Networks to Improve their Diagnosability," IEEE Trans. on Computers, Vol. C-23, January 1974, pp. 56-62.
[16] Akers, S. B. "Partitioning for Testability," J. Design Automation and Fault-Tolerant Computing, Vol. 1, February 1977, pp. 133-146.
[17] Buehler, M. G. and M. W. Sievers, "Off-line, Built-in Test Techniques for VLSI Circuits," Computer, June 1982, pp. 69-82.

Some popular structural built-in test methods include:

a. Semi built-in

(1) Level-sensitive scan design[1,2]
(2) Scan path[18]
(3) Random-access scan[19]
(4) Scan/set logic[20]
(5) Partitioning[21]

b. Fully built-in

(1) On-Line

- Error detection and correction codes[22]
- Totally self-checking circuits[23]
- Self-verification[24]

(2) Off-Line

- BILBO[25,26]
- Store and generate[27]

[18] Funatsu, S., N. Wakatsuki, and T. Arima, "Test Generation Systems in Japan," Proceedings of the 12th Design Automation Conference, June 1975, pp. 114-122.

[19] Ando, H., "Testing VLSI with Random Access Scan," Digest of Papers COMPCON 80, February 1980, pp. 50-52.

[20] Stewart, J. H., "Future Testing of Large LSI Circuit Cards," Digest of Papers IEEE Semiconductor Test Symposium, October 1977, pp. 6-17.

[21] McCluskey, E. J. and S. Bozorgui-Nesbat, "Design for Autonomous Test," IEEE Trans. on Computers, Vol. C-30, November 1981, pp. 866-875.

[22] Peterson, W. W. and E. J. Weldon, Error-Correcting Codes, 2nd Edition, MIT Press, Cambridge, MA, 1972.

[23] Wakerly, J., Error Detecting Codes, Self-checking Circuits and Applications, American-Elsevier, NY, 1978.

[24] Sedmak, R. M., "Design for Self-Verification: An Approach for Dealing with Testability Problems in VLSI-Based Designs," Proceedings IEEE Test Conference, 1979, pp. 112-120.

[25] Konemann, B., J. Mucha and G. Zwiehoff, "Built-in Logic Block Observational Techniques," Digest of Papers 1979 Test Conference, October 1979, pp. 37-41.

[26] Mucha, J., "Hardware Techniques for Testing VLSI Circuits Based on Built-in Tests," Digest of Papers COMPCON 81, February 1981, pp. 366-369.

[27] Agarwal, V. K. and E. Ceruy, "Store and Generate Built-in Testing Approach," Digest of Papers 11th International Symposium on Fault-Tolerant Computing, June 1981, pp. 35-40.

- o Verification of Walsh coefficients[28]
- o Autonomous testing[29,30]
- o Syndrome testing[8,9]

3.2.3 Designing with Easily Testable Components

Designing with easily testable components is a methodology which deals primarily with the design of the kernel itself, and where the main objective is to make the kernel easy to test. A simple example would be those techniques which rely heavily on the use of exclusive-or gates. For such circuits, a single error on an input always produces an output error, making the concept of path sensitization particularly easy to achieve.

This methodology includes:

a. EOR trees[31]
b. Canonic Reed-Muller circuits[32]
c. Easily testable PLAs[33]
d. Easily testable iterative logic arrays[34]
e. Bit-slice systems[35]

---

[28]Susskind, A. K., "Testing by Verifying Walsh Coefficients," Digest of Papers 11th International Symposium on Fault-Tolerant Computing, June 1981, pp. 206-208.

[29]McCluskey, E. J. and S. Bozorgui-Nesbat, "Design for Autonomous Test," Proceedings IEEE International Test Conference, November 1980, pp. 15-21.

[30]Bozorgui-Nesbat, S. and E. J. McCluskey, "Structured Design for Testability to Eliminate Test Pattern Generation," Digest of Papers 10th International Symposium on Fault-Tolerant Computing, June 1980, pp. 158-163.

[31]Seth, S. C. and K. L. Kodandapani, "Diagnosis of Faults in Linear Tree Networks," IEEE Trans. on Computers, Vol. C-26, January 1977, pp. 29-33.

[32]Saluja, K. K. and R.M. Reddy, "Fault Detecting Test Sets for Reed-Muller Canonic Networks," IEEE Trans. on Computers, Vol. C-24, October 1975, pp. 995-998.

[33]Fujiwara, H. and K. Kinoshita, "A Design of Programmable Logic Arrays with Universal Tests," IEEE Trans. on Computers, Vol. C-30, November 1981, pp. 823-828.

[34]Friedman, A. D., "Easily Testable Iterative Systems," IEEE Trans. on Computers, Vol. C-22, December 1973, pp. 1061-1064.

[35]Sridhar, T. and J. P. Hayes, "A Functional Approach to Testing Bit-Sliced Microprocessors," IEEE Trans. on Computers, Vol. C-30, August 1981, pp. 563-571.

3.2.4 Analysis Tools

Several analysis tools have been proposed for aiding design for testability. These analysis tools usually estimate the degree of controllability and observability of the various signal lines in a circuit. Based on these results, the circuit design should be modified, if necessary, in order to enhance testability.

Several analysis tools are:

a. Measurements

(1) COMET[36]
(2) SCOAP[37,38]
(3) TMEAS[39]
(4) CAMELOT[40]

b. Design: Automatic design for testability[41]

3.3 STRUCTURE AND TESTING

Four important factors to be considered in testing a kernel are:

a. Fault modes

b. Whether or not a single vector or a sequence of vectors are required to detect a fault

[36]Chang, H. Y. and S. W. Heimbigner, "LAMP: Controllability, Observability and Maintenance Engineering Technique (COMET)," Bell System Tech. J., Vol. 53, October 1974, pp. 1505-1534.
[37]Goldstein, L. H. and E. L. Thigpen, "SCOAP: Sandia Controllability Observability Analysis Program," Proceedings of the 17th Design Automation Conference, June 1980, pp. 190-196.
[38]Goldstein, L. H., "Controllability/Observability Analysis of Digital Circuits," IEEE Trans. Circuits and Systems, Vol. CAS-26, September 1979, pp. 685-693.
[39]Grason, J., "TMEAS, A Testability Measurement Program," Proceedings of the 16th Design Automation Conference, June 1979, pp. 156-161.
[40]Bennetts, R. G., et al., "CAMELOT: A Computer-Aided Measure for Logical Testability," IEE Proceedings, Vol. 128, Part E (Comp. & Digital Techniques), London, September 1981, pp. 177-189.
[41]Chen, T-H. and M. A. Breuer, "Automatic Design for Testability via Testability Measures," IEEE Transactions on Computer-Aided Design, Vol. CAD-4, January 1985, pp. 3-11.

c. Complexity of test generation

d. Timing

These factors are primarily influenced by the structure of a kernel and its design style.

Fault modes are often a function of design style. RAMs exhibit the phenomenon of adjacent pattern interference; PLAs are susceptible to cross-point failures (extra or missing connections); and gate combinational networks are often tested for stuck-at faults, shorts, and sometimes memory retention.

For a combinational circuit, only one vector is usually required to detect a fault, while for sequential circuits a sequence of test vectors is often necessary. Faults in combinational circuits which induce memory retention may require a sequence of two vectors to detect.

The complexity of test generation is strongly related to design style as well as circuit structure. For RAMs, standard test sequences usually exist. Automatic test generation is usually a difficult if not impossible task for complex random sequential circuits. For PLAs, special algorithms exist which make test generation a fairly effective and efficient process.

Finally, timing issues related to factors such as races, hazards, and static and dynamic logic are a function of both design style and circuit structure. For example, asynchronous circuits are circuit structures and are susceptible to races. A RAM design style may be susceptible to pattern interference faults which are both timing- and data-sensitive.

In summary, different design styles and circuit structures have unique testing characteristics and are thus amenable to unique testing approaches and built-in test strategies. As an example, a PLA can be built such that the signal values on the row (product) and column (word) lines have odd parity;[33] this concept is not directly applicable to a gate combinational network implementation of the same functions. A unique logic structure for the testing of internal arrays, and the testing for pattern-sensitive faults in RAMs are discussed in the literature.[42,43]

[42]Eichelberger, E. B., et al., "A Logic Design Structure for Testing Internal Arrays," Proceedings of the 3rd USA-Japan Computer Conference, October 1973, pp. 266-272.

[43]Hayes, J. P., "Detection of Pattern Sensitive Faults in Random Access Memories," IEEE Trans. on Computers, Vol. C-24, February 1975, pp. 150-157.

4. TEST STRATEGIES

A test strategy for a kernel structure is the complete process involved in testing the structure. This includes the following three main attributes:

a. Off-line test generation

b. Run-time test hardware

  Automatic test equipment (external)
  Built-in test (internal)

c. Test accessibility

  Controllability
  Observability

4.1 OFF-LINE TEST GENERATION

Off-line test generation is the method used to derive test vectors and sequences. This process is necessary for some types of test strategies, e.g., in the LSSD methodology, but not for others, e.g., when a circuit is tested using the BILBO methodology. There are several ways to carry out off-line test generation, some of which are summarized below:

a. Manual

  Circuit-oriented, e.g., process-sensitized paths
  Functional, e.g., execute every instruction

b. Algorithmic/heuristic

  PODEM[44]
  D-algorithm[45]

[44]Goel, P., "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," Digest of Papers 10th International Symposium on Fault-Tolerant Computing, October 1980, pp. 145-150.

[45]Roth, J. P., "Diagnosis of Automata Failures: A Calculus and a Method," IBM J. Research and Development, Vol. 10, July 1966, pp. 278-291.

PLA test generation[46]
Delay test generation[47]
LASAR (D-LASAR, LASAR 5.6)[48]
Functional[49]

c. Pseudo-random[50,51]

d. Exhaustive (not normally done off-line)

e. Standard test sets

GALPAT for RAMs[5,52]
Universal test sets for PLAs[33,53]

Except for exhaustive and standard test sets, tests once generated are usually processed through a fault simulator to determine fault coverage.

Note that the process of off-line test generation can involve the overhead of a complex and sophisticated suite of software modules, including design capture, testability analysis, test generation, and fault simulation routines. The resulting tests are often processed via additional software

[46]Eichelberger, E. B. and E. Lindbloom, "A Heuristic Test-Pattern Generator for Programmable Logic Array," IBM J. Research and Development, Vol. 24, January 1980, pp. 15-22.
[47]Lesser, J. D. and J. J. Shedletsky, "An Experimental Delay Test Generator for LSI Logic," IEEE Trans. on Computers, Vol. 29, March 1980, pp. 235-248.
[48]Thomas, J. J., "Automated Diagnostic Test Programs for Digital Networks," Computer Design, Vol. 10, August 1971, pp. 63-67.
[49]Thatte, S. M. and J. A. Abraham, "Test Generation for Microprocessors," IEEE Trans. on Computers, Vol. C-29, June 1980, pp. 429-441.
[50]Bastin, D., et al., "Probabilistic Test Generation Methods," Digest of Papers 3rd International Symposium on Fault-Tolerant Computing, June 1973, p. 171.
[51]Parker, K. P., "Adaptive Random Test Generation," J. Design Automation and Fault Tolerant Computing, Vol. 1, October 1976, pp. 62-83.
[52]Hnatek, E. R., A User's Handbook of Semiconductor Memories, Wiley-Interscience, NY, 1977.
[53]Hong, S. J. and D. L. Ostapko, "FITPLA: A Programmable Logic Array for Functional Independent Testing," Digest of Papers 10th International Symposium on Fault-Tolerant Computing, October 1980, pp. 131-136.

to create a fault dictionary, if required, and via a translator in order to obtain a test program that runs on a specified piece of automatic test equipment.

4.2 RUN-TIME TEST HARDWARE

Run-time test hardware is the hardware used during the actual testing process of the structure. This hardware is used to produce the test vectors required to test the circuit structure as well as process the responses obtained. Table 1 summarizes some of the hardware used in this process. Two main categories of hardware are used: external automatic test equipment and internal built-in test circuitry.

Table 1. Run-Time Test Hardware

- *Off-chip automatic test equipment*
- On-chip built-in test circuitry

    Generation of test stimuli

    BILBO register
    Linear feedback shift register
    Counter (exhaustive testing)
    ROM (stored test patterns)
    General sequential circuit
    Gray code generator

    Processing of test responses

    Signature generator
    BILBO register
    Syndrome generator/one's counter
    Transition counter
    Comparator
    RAM (store responses)
    Parity detector
    Single error correction-double error detection
    General sequential circuit

4.3 TEST ACCESSIBILITY

During the testing process, one needs a hardware mechanism in order to actually apply the test vectors to the inputs of the kernel structure under test, as well as observe the response data produced at the outputs of this structure. Since this structure is often deeply buried within a chip, built-in test features are often added to the circuit to implement these controllability and observability functions. Table 2 indicates some examples of how that accessibility is achieved.

Table 2. Test Accessibility

- o Input
  - Primary inputs
  - Scan-in registers
  - LSSD registers
  - BILBO register
  - Multiplexers
- o Output
  - Primary outputs/test points
  - Scan-out registers
  - LSSD registers
  - BILBO registers
  - Multiplexers

In some cases, such as with a BILBO register, the run-time test hardware and the test accessibility registers are one and the same. For the LSSD methodology, this is not the case. Tests are first generated off-line, usually using some type of test algorithm; external hardware (automatic test equipment) is then used at test run time to generate and process the tests. LSSD registers are then used only to achieve input and output access to the structure under test.

In summary, a test strategy involves three key concepts; namely, a means for generating input test data, the hardware required to produce the test vectors and process responses during the testing cycle, and a means for applying the input test data to the input lines and observing the response data at the output lines of the circuit structure under test. In some cases, the kernel itself is modified in order to enhance its testability.[33]

5. TEST STRATEGY MEASURES

Numerous test strategies exist. With each test strategy one can associate several measures dealing with performance criteria, constraints, and goals. An example of a performance criterion is the length of time it takes to test a circuit structure; an example of a constraint is that the input and output pin requirements for the built-in test circuitry be less than some given quantity; finally, an example of a goal is that the test strategy achieve at least 98 percent fault coverage of the single detectable stuck-at faults.

The three concepts of performance, constraints, and goals have been lumped together because they are usually highly interrelated, and often tradeoffs are made between them. For example, achieved fault coverage is often a function of the expense one is willing to incur in test generation. The incremental increase in fault coverage as a function of cost may be extremely high as one approaches 100 percent coverage. Also, for sequential circuits, the incremental increase in test length for each 1 percent additional fault coverage may become extremely large. Hence, all goals may not be feasible. Unfortunately, the quantitative prediction of performance measures is a difficult task. One cannot, for example, predict a priori the cost of test generation versus fault coverage for a given circuit.

Because of these dichotomies, the concepts of performance, constraints, and goals have been combined into the general category of measures. In Table 3, several important measures are listed which may need to be considered in selecting a test strategy for a circuit structure.

The tradeoff between more area for built-in test circuitry and decreased chip functionality leads to a classic battle between chip designers and users. Hence, the driving force for using built-in test circuitry comes from design specifications where the testability and functionality of the chip are made equally important design criteria.

Table 3. Measures Associated with a Test Strategy

M1	Yield and area effect due to built-in test circuitry	
	Example: o	LSSD often requires a 5 to 20 percent area overhead
M2	Test Application Time	
	Example: o	In LSSD, each test vector is shifted sequentially, slowing down the test process
M3	Input and output pin demand	
	Example: o	LSSD requires four additional pins
M4	Fault coverage and fault types	
	Examples: o	For LSSD, coverage of the single stuck-at fault can be arbitrarily high and can be measured via fault simulation
	o	For BILBO testing, coverage is difficult to determine
	o	For autonomous testing, coverage is essentially complete for all fault modes
M5	Test input or output storage volume (on chips)	
	Examples: o	For microdiagnostics, test volume is high.
	o	For signature generation, volume is low.
	o	For LSSD, no on-chip storage is required.
M6	Performance degradation	
M7	Preprocessing (off-line) costs	
M8	Cost of off-line automatic test equipment	
M9	Cost of accommodating engineering changes	

Test application time is usually critical when expensive automatic test equipment is employed. When a chip is part of a large system, such as a space satellite which employs off-line self-test procedures, testing time may be important because it may significantly affect the time the system is not available for normal use.

Performance degradation deals with the effect on a circuit's operating characteristics during its functional operation due to built-in test hardware. For example, using a pair of level-sensitive latches in a feedback path (as found in LSSD) instead of some other form of flip-flop may reduce the system clock rate by a small amount.

Preprocessing cost deals with the process of off-line test generation and the associated costs of acquiring and executing the required software.

Finally, the cost of processing engineering changes varies widely for different test strategies. When off-line test generation is employed, processing an engineering change can be quite costly.

6. TESTABLE DESIGN METHODOLOGY

The combination of a kernel structure S and a test strategy (test generation, run-time test hardware, and hardware for accessibility) constitutes a testable design methodology. If the structure S has a design style D, then it can be said that the testable design methodology is for design style D.

The general form for a testable design methodology is represented as follows:

A1. A kernel structure to be tested
(optional: A basic circuit structure and its design style)

A2. A test strategy

A2.1	An off-line test generation strategy
A2.2	A run-time testing environment
A2.3	Hardware for test accessibility

6.1 EXAMPLE: LEVEL-SENSITIVE SCAN DESIGN (LSSD)

As an example, an LSSD is associated with a testable design methodology having the following attributes:

A1	Gate combinational network
A2.1	Test generation algorithm/fault simulator/translator
A2.2	Automatic test equipment
A2.3	Level-sensitive scan design registers

Figure 1 indicates the major components associated with the LSSD testable design methodology. In Figure 2 a specific example of a testable circuit structure having an LSSD testable structural style is shown.

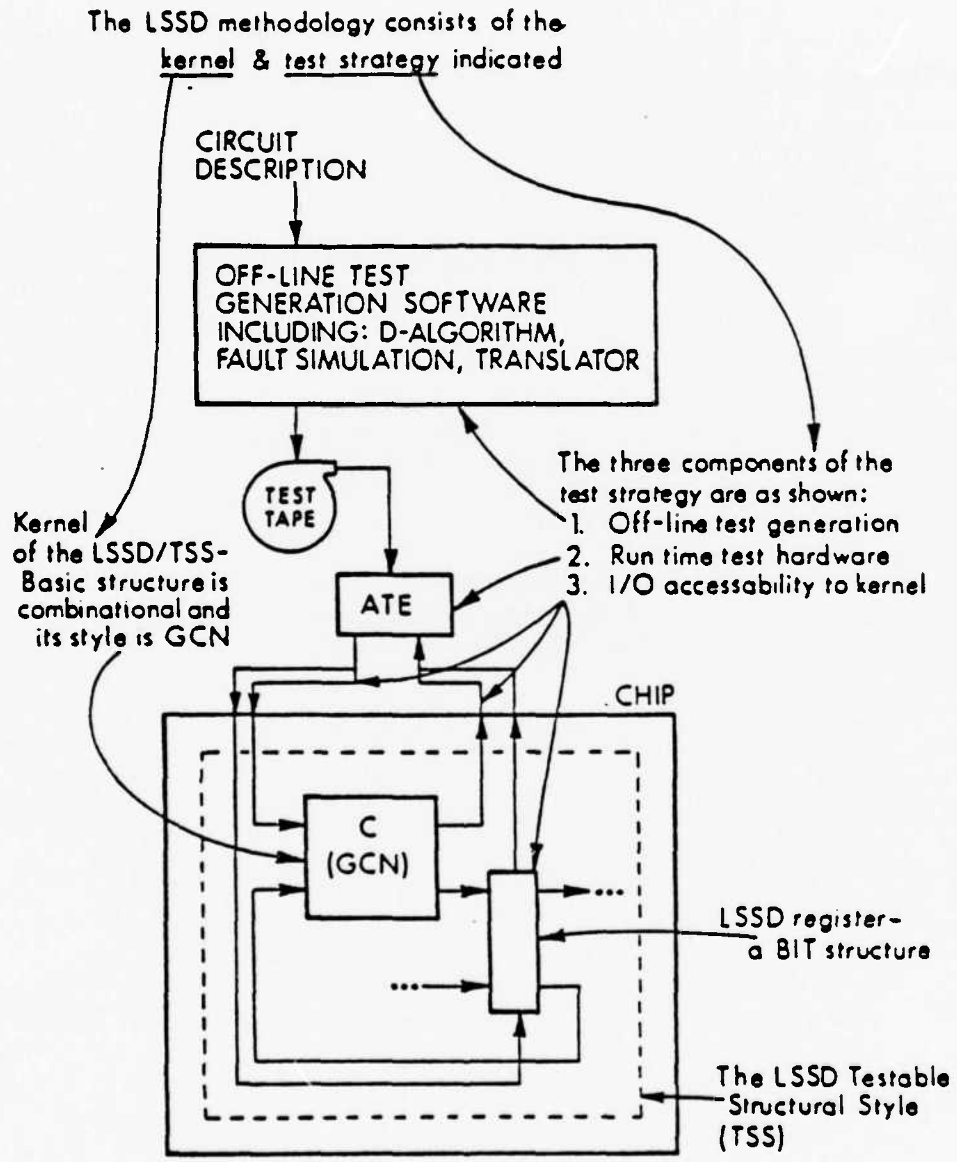

The LSSD methodology consists of the kernel & test strategy indicated
CIRCUIT DESCRIPTION
OFF-LINE TEST GENERATION SOFTWARE INCLUDING: D-ALGORITHM, FAULT SIMULATION, TRANSLATOR
TEST TAPE
ATE
The three components of the test strategy are as shown:
1. Off-line test generation
2. Run time test hardware
3. I/O accessability to kernel
Kernel of the LSSD/TSS- Basic structure is combinational and its style is GCN
CHIP
C (GCN)
LSSD register- a BIT structure
The LSSD Testable Structural Style (TSS)

Fig. 1. Level-Sensitive Scan Design Testable Design Methodology

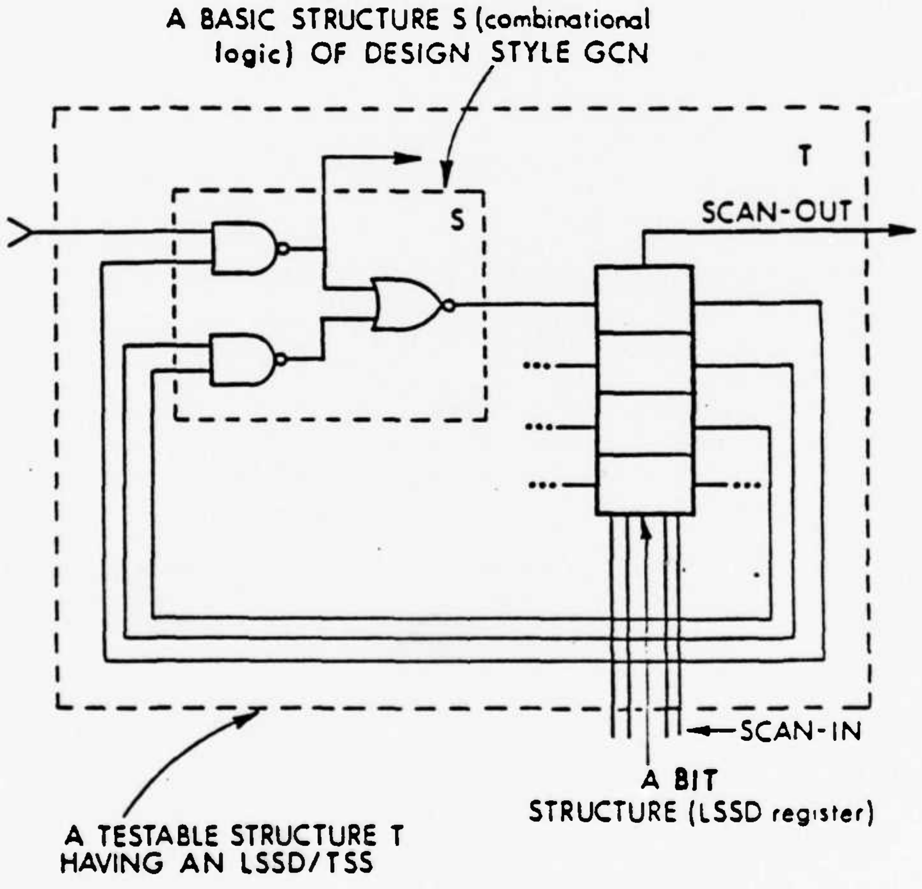

A BASIC STRUCTURE S (combinational logic) OF DESIGN STYLE GCN
T
S
SCAN-OUT
SCAN-IN
A BIT STRUCTURE (LSSD register)
A TESTABLE STRUCTURE T HAVING AN LSSD/TSS

Fig. 2. Testable Structure with a Level-Sensitive Scan Design Testable Structural Style

The space of testable design methodologies can be thought of as a multi-dimensional space having the following three main components:

a. The structure of the circuit to be tested and possibly its basic structure and design style

b. The test strategy selected to test the circuit

c. The value of the measures, such as M1 through M9, associated with the above two items

Given this space, some testable design methodologies can be judged to be good, others to be poor. For example, replacing the gate combination network by a RAM in the LSSD methodology would not lead to a useful testable design methodology.

6.2 DESIGN PROBLEM

The main tasks in designing a testable CLSIC chip can be stated as follows:

a. Partition a design into circuit structures. Depending on the testing strategy to be used, some or all of these structures may be basic circuit structures having well-defined design styles.

b. Select an appropriate test strategy for each structure.

c. Modify the design as necessary to implement the selected testable design methodologies which satisfy all measures associated with the chip.

7. CHIP BUILT-IN TEST CIRCUITRY

In making a chip testable, several standard hardware structures are often added to the chip in order to enhance its testability. Examples of such built-in test circuits are:

a. Set/scan registers, e.g., LSSD registers

b. Counters (generates 2^n test vectors)

c. BILBO registers

d. Comparators

e. Linear feedback shift registers

f. Parity generators

Over the last several years, increased levels of observability and controllability in VLSI circuits have been obtained by replacing normal flip-flops in a circuit by dual mode registers which, in normal mode, act as normal flip-flops. In the test mode, they act as shift registers, enabling test vectors to be scanned into the circuit and test responses to be scanned out. To achieve exhaustive testing, counters can be added to a circuit so that all possible test patterns can be generated. To carry out ones or transition count testing, a count register can be used. Between these two extremes, one can employ linear feedback shift registers, such as in the BILBO methodology, to either generate pseudorandom test vectors or to generate a signature. Finally, a comparator can be used to compare a generated signature with a stored correct signature. When these test circuits and others are used, powerful testable structural styles can be created.

Except for the parity generator, the test circuits listed previously are used for off-line testing. When on-line testing is used, then other built-in test circuits are employed. They are usually used to implement some coding or decoding scheme. Other examples of such test circuits are self-checking checkers.[5]

8. EXAMPLES OF TESTABLE DESIGN METHODOLOGIES

This section briefly illustrates a few popular testable design methodologies.

8.1 LEVEL-SENSITIVE SCAN DESIGN[1,2]

Probably the most well-known testable design methodology is the LSSD testable design methodology introduced by IBM. This methodology has been depicted in Figures 1 and 2.

8.2 SCAN PATH DESIGN[18]

This methodology is similar to the LSSD testable design methodology. The main differences lie in the type of flip-flops used in the registers and the clocking scheme employed.

8.3 SCAN-SET DESIGN[20]

The scan-set testable structural style is shown in Figure 3. Note that the kernel structure is now a sequential circuit; hence, the off-line test generation process for this methodology can be significantly more complex than that for the previous two methodologies. The register can either load data (observability) in parallel from test points in the kernel structure and shift these data out (scan-out), or else scan-in new data (controllability) and apply these data to test points in the kernel.

8.4 RANDOM ACCESS SCAN DESIGN [19]

The testable structural style for the random access scan testable design methodology is shown in Figure 4. Again, off-line test generation is required along with automatic test equipment, and the kernel is combinational. For this testable structural style, the flip-flops in the original sequential circuits

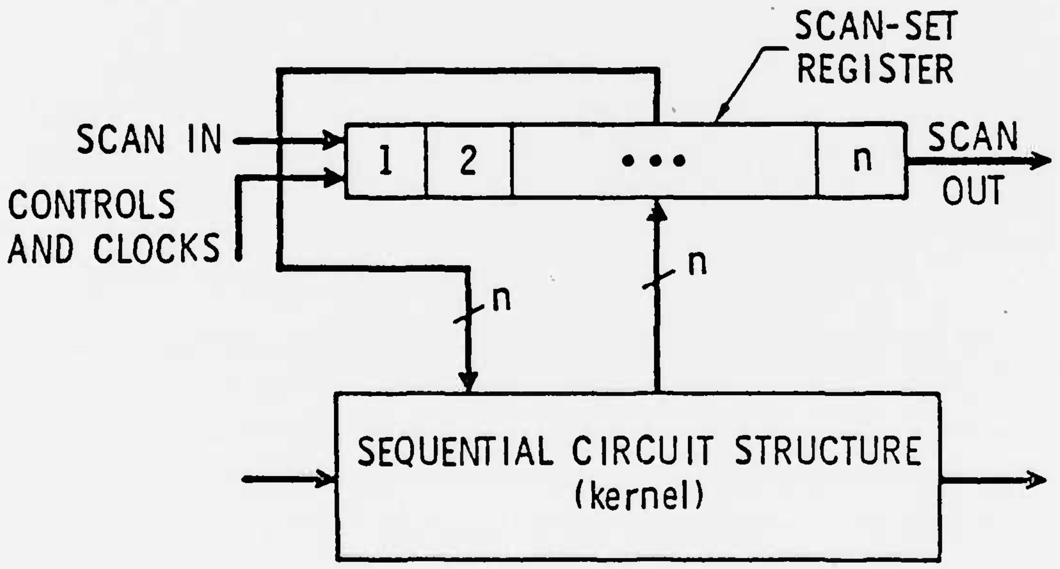


Fig. 3. Testable Structural Style Used in the Scan-Set Testable Design Methodology

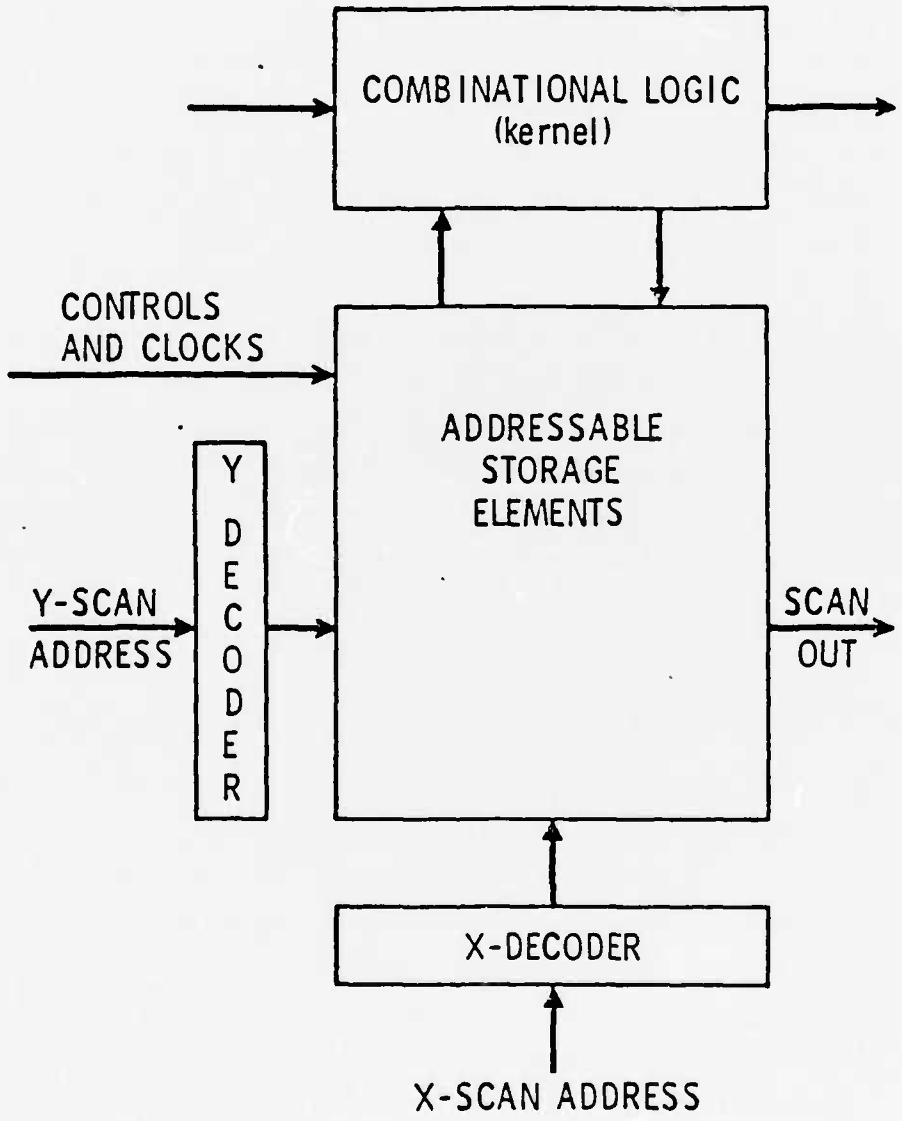


Fig. 4. Random Access Scan Testable Structural Style

are made individually addressable during the testing mode, and their contents are set and read via the automatic test equipment. During the normal mode of operation, the kernel and flip-flops in the addressable storage array operate as a normal sequential circuit.

8.5 BUILT-IN LOGIC BLOCK OBSERVATION DESIGN[25,26]

This testable structural style is an example of a fully built-in test approach; hence, no off-line test generation is used, and only minimal automatic test equipment is required. The BILBO registers carry out four functions for testing: controllability, test vector generation, observability, and test response processing (signature generation).

Figure 5 shows the testable structural style used in the BILBO testable design methodology. The kernel is again combinational logic and usually of the gate combination network design style. Since this approach is based upon pseudorandom test patterns, a ROM or PLA design style is not suitable. The circuit C_1 is tested by configuring the BILBC register on the left as a pseudorandom pattern generator and the BILBO on the right as a signature generator.

8.6 SYNDROME DESIGN[7,8,9]

The testable structural style for the syndrome testable design methodology is shown in Figure 6. Again, the kernel is combinational, but this approach is applicable to gate combinational network, PLA, or ROM design styles. Only a single output is indicated. Testing is accomplished by having the counter produce all 2^n input vectors, while the count register counts the number of 1's on the output. The correct number of 1's is the number of minterms in the function realized by C and is denoted by K. Then $S = K/2^n$ is said to be the syndrome. Fault detection is achieved by comparing the final state of the count register with S. In this built-in self-test methodology, no off-line test generation is required, and the automatic test equipment requirements are minimal. Often, the design of the circuit C (for gate combinational network and PLA design styles) is modified to enhance testability;

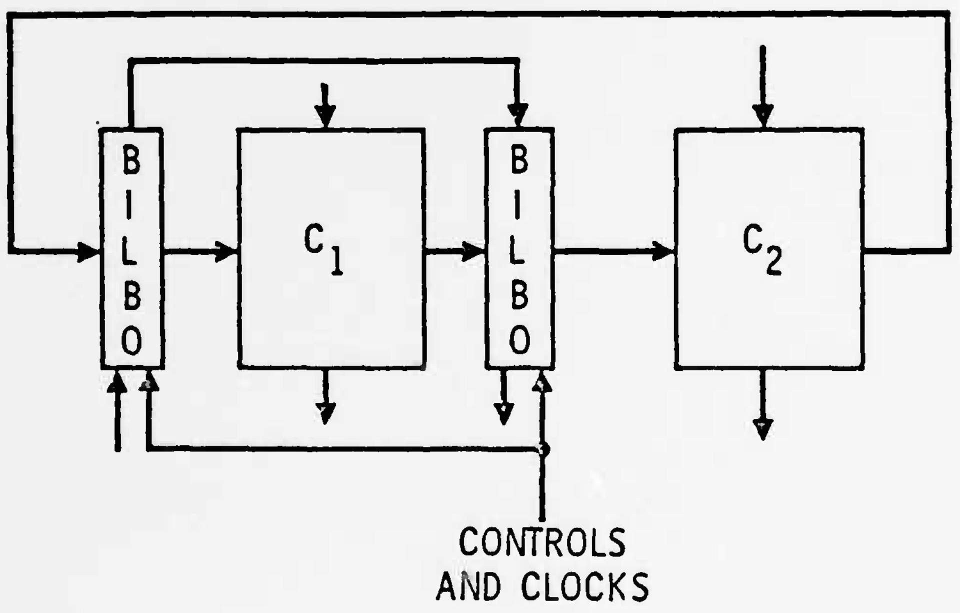


Fig. 5. Built-in Logic Block Observation Testable Structural Style

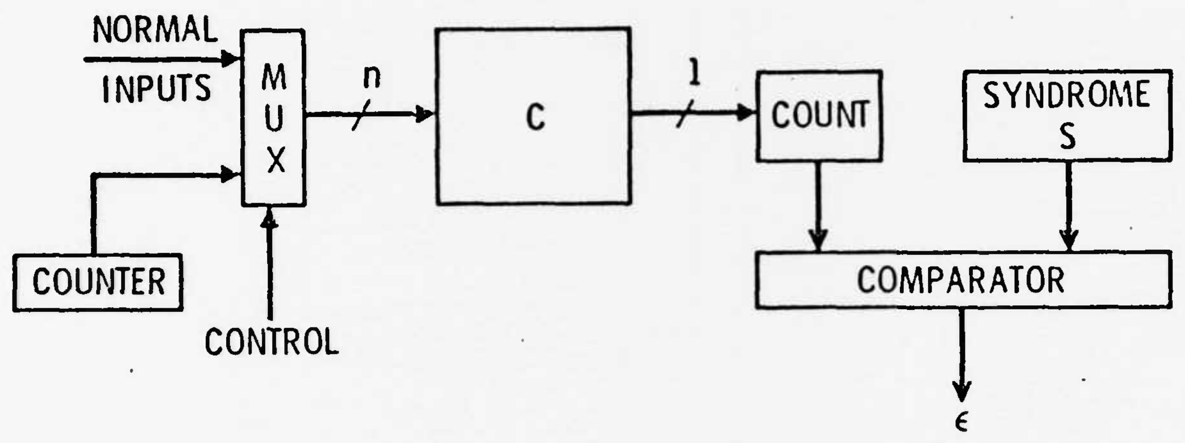


Fig. 6. Syndrome Testable Structural Style

e.g., a syndrome testable circuit is one for which every single stuck-at fault is detectable by this testing approach.

There are several variations to this form of testing. For autonomous testing,[21,29,30] the output of the kernel is directly observed by an automatic test equipment, rather than compacted into a signature (syndrome). This form of testing thus guarantees detection of all faults which are not sequential in nature. Alternatively, the response can be processed via a linear feedback shift register, and again a signature can be generated.

8.7 EASILY TESTABLE BIT-SLICED DESIGN[35]

While bit-sliced architectures are usually implemented via interconnecting chips, as the level of integration increases these architectural styles will be used more extensively at the chip level. One reason for this is regularity in layout and testing. A testable structural style ideal for bit-sliced architectures has been developed. One version of this architecture is for CI-testable arrays. To introduce this concept, a few definitions are needed. An iterative logic array is a one-dimensional cascade of identical cells (see Fig. 7). The cells can be either combinational or sequential circuits. An iterative logic array is said to be C-testable if it can be tested with a constant number of test patterns, independent of the array size N. Let T be a test set that tests an iterative logic array D completely under the assumption that only one cell in the array is faulty. Then D is I-testable with respect to T if the expected responses to T appearing at the vertical outputs of every cell L_i of D are identical. A CI-testable iterative logic array is both C-testable and I-testable with respect to some test set T. The necessary and sufficient conditions for an iterative logic array to be CI-testable have been determined.[34]

In Figure 7, L_1, L_2,....L_N represents the CI-testable iterative logic array to be tested. The normal inputs and outputs are shown. The test T can be stored off-chip and applied via automatic test equipment or on-chip and stored in a ROM. The equality checker determines if the responses from

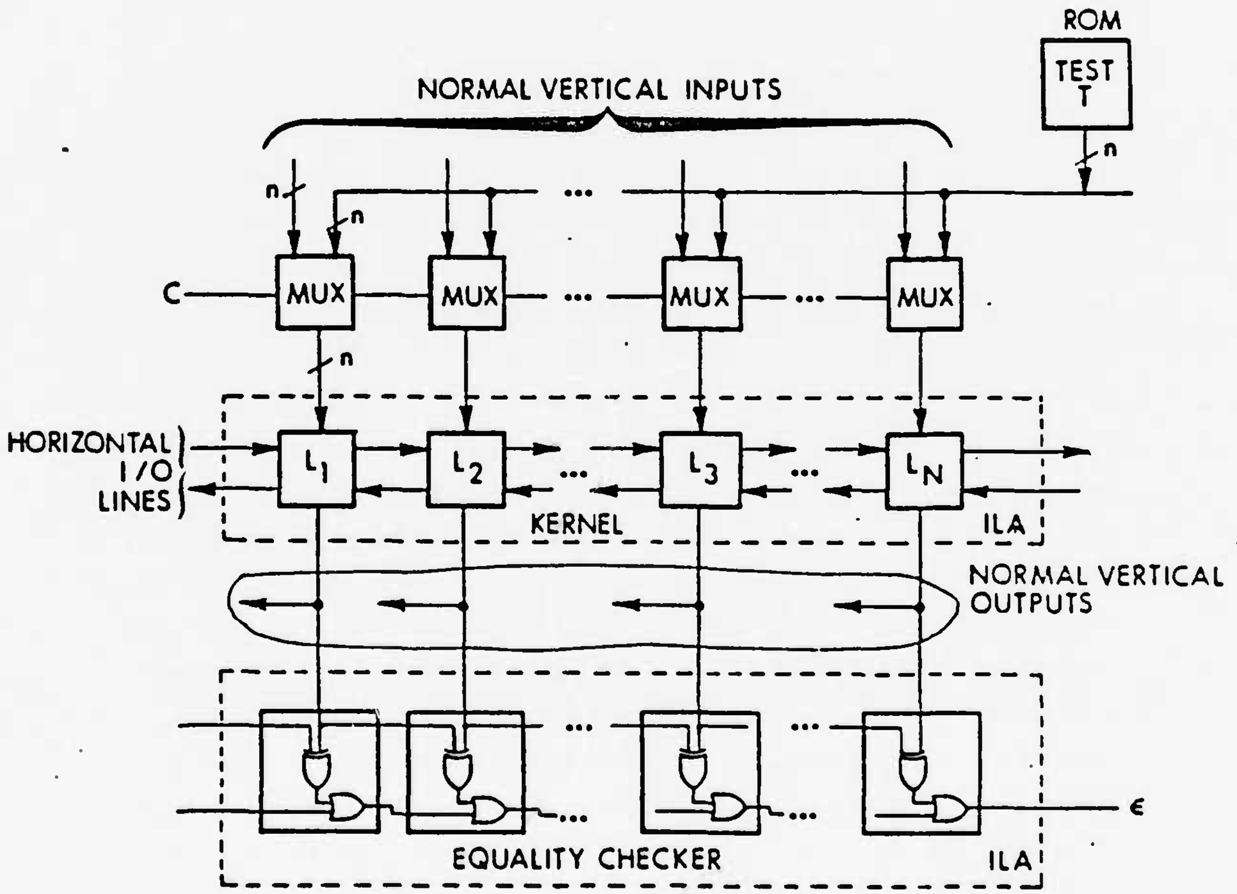


Fig. 7. Bit-Sliced Testable Structural Style

each L_i are identical. The case of a single output line from each L_i is shown, but the concept can be easily generalized to the case of multiple output lines.

Off-line test generation is required for this methodology; for complex cells, this process may be quite difficult and require the use of checking sequences.[54] Real-time test hardware can be either on-line or off-line. Test application to the kernel is achieved via the multiplexers, while observability of the responses is not required due to the equality checker and the concept of I-testability.

8.8 SUMMARY

In summary, fully built-in testing deals with those test strategies where the role of the external test equipment is minimal. BILBO and syndrome testing are examples of methodologies which employ fully built-in testable structural styles. The general architecture for such a style is shown in Figure 8. Table 4 summarizes the various options for each block in Figure 8. When built-in test structures are added to a circuit, care must be taken to ensure that the test structures are themselves tested, either implicitly or explicitly. Also, when several different testable structures exist on a chip, some additional hardware overhead may be required to control the test process.

[54]Friedman, A. D., and P. Menon, Fault Detection in Digital Circuits, Prentice Hall, NJ, 1971.

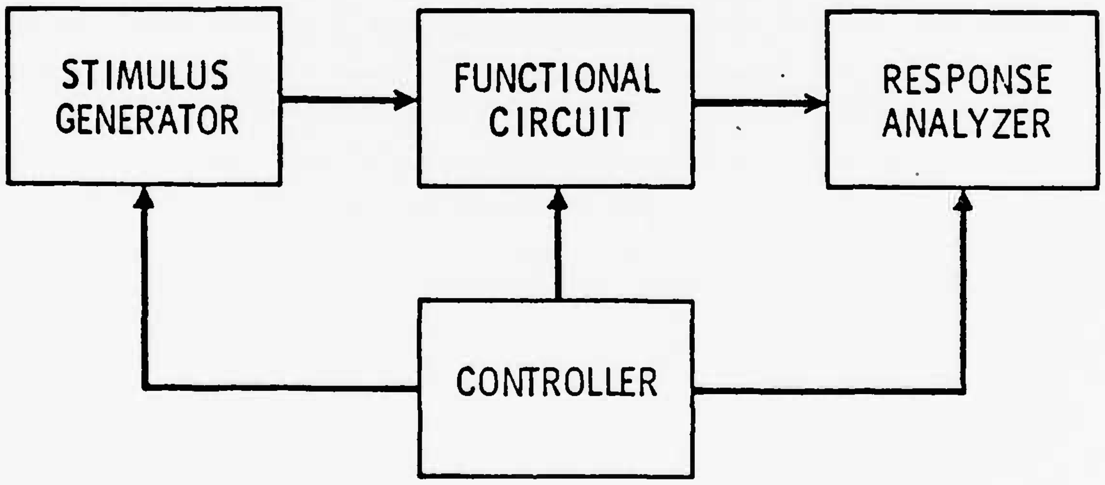


Fig. 8. General Form for a Fully Built-in Testable Structural Style

Table 4. Some Options in the Design of a Fully Built-in Testable Structural Style

- Stimulus Generator
  - Hardware test generation
    - random patterns using a linear feedback shift register
    - all input combinations using a linear feedback shift register or a counter (exhaustive)
    - some specified patterns using a nonlinear feedback shift register
  - Stored test patterns
  - Store and generate--store some pre-calculated patterns as initial values for a linear feedback shift register
- Functional circuit
  - Sequential circuit--can be partitioned into combinational parts using set/scan registers
  - Combinational circuit--partition into manageable subcircuits
- Response analyzer
  - Use compressed responses
    - syndrome (one's counting)
    - signature using linear feedback shift register
    - transition counting
  - Store the correct responses
  - Generate the correct response
  - Compare responses with correct ones and generate go or no-go signal
- Controller
  - Control transition between test mode and normal mode
  - Control testing process

9. TESTABLE DESIGN METHODOLOGIES FOR PROGRAMMABLE LOGIC ARRAYS

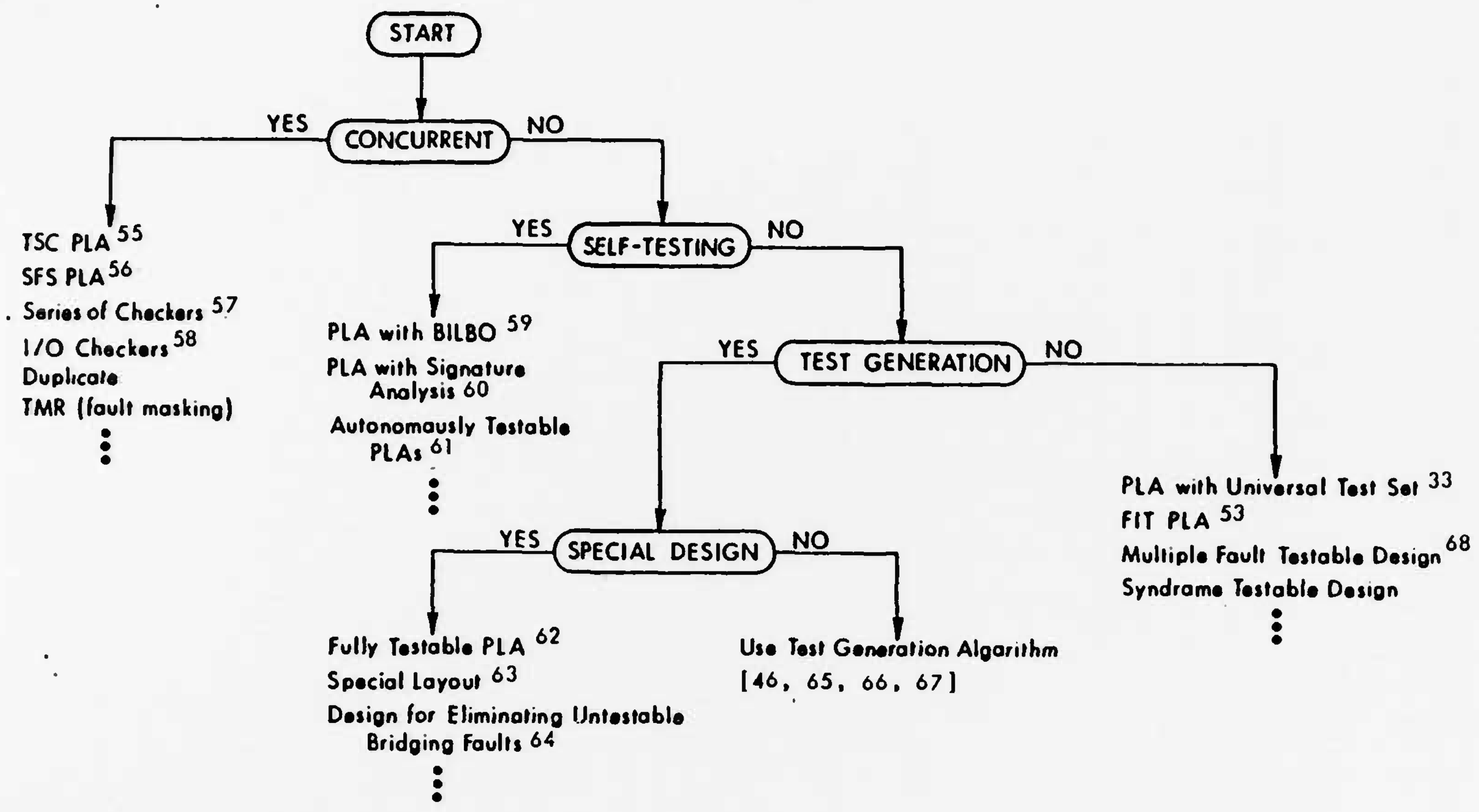
Numerous techniques for testing the PLA design style have been suggested. Figure 9 indicates several testable design methodologies for PLAs according to certain attributes, such as whether or not they support concurrent testing, produce a self-testing PLA, require off-line test generation, and are based upon a special design approach. Naturally, these techniques could have been classified and grouped differently, such as by fault coverage area overhead.

Figures 10 through 12 indicate the testable structural styles corresponding to just three of the techniques listed in Figure 9.

9.1 PROGRAMMABLE LOGIC ARRAY WITH UNIVERSAL TEST SET

Figure 10 indicates a testable structural style for a PLA which employs a universal test set, hence no test pattern generation is required.[33] The normal design of the PLA is shown in heavy lines. The medium lines indicate added built-in test structures, and the thin lines indicate wires. The product term selector is a shift register; the data in this register enable and disable the product lines in the array. The AND array is extended by one product line such that each input row has an odd parity; a word parity line is also added to the OR array. The inputs y_0, y_1, y_2 are used to control the circuit during the normal and test modes. An error is indicated by testing the two lines (Z_1, Z_2). This test can be done on-chip or off-chip. The D_{in} is a new input used to supply data to the product term selector register. Normally, the universal test set is stored off-chip and is applied via the automatic test equipment.

If the PLA has n inputs and m product lines, then the number of tests in the universal test set is 2n + 3m. These tests detect all single stuck-at faults in the decoder blocks f and the PLA, all crosspoint faults in the PLA, and all stuck-at faults in the parity chain #1.



Note: References listed on page 53.

Fig. 9. Testable Design Methodologies for Programmable Logic Arrays

References from Figure 9:

[55]Wang, S. L. and A. Avizienis, "The Design of Totally Self-Checking Circuits Using Programmable Logic Arrays," Digest of Papers 9th Internati nal Symposium on Fault-Tolerant Computing, June 1979, pp. 173-180.

[56]Mak, G. P., J. A. Abraham, and E. S. Davidson, "The Design of PLAs with Concurrent Error Detection," Digest of Papers 12th International Symposium on Fault-Tolerant Computing, June 1982, pp. 303-310.

[57]Khakbaz, J. and E. J. McCluskey, "Concurrent Error Detection and Testing for Large Programmable Logic Arrays," CRC Report No. 81-14, Stanford University, Stanford, CA, September 1981.

[58]Dong, H. and E. J. McCluskey, "Concurrent Testing of Programmable Logic Arrays," CRC Report No. 82-11, Stanford University, Stanford, CA, June 1982.

[59]Daehn, W. and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays," IEEE Trans. on Computers, Vol. C-30, No. 11, November 1981, pp. 829-833.

[60]Hassan, S. Z., "Testing PLAs Using Multiple Parallel Signature Analyzers," CRC Report No. 82-9, Stanford University, Stanford, CA, June 1982.

[61]Yajima, S. and T. Aramaki, "Autonomously Testable Programmable Logic Arrays," Digest of Papers 11th International Symposium on Fault-Tolerant Computing, June 1981, pp. 41-43.

[62]Dong, H. and E. J. McCluskey, "Design of Fully Testable Programmable Logic Arrays," CRC Report No. 82-20, Stanford University, Stanford, CA, June 1982.

[63]Son, K. and D. K. Pradhan, "Design of Programmable Logic Arrays for Testability," Proceedings IEEE International Test Conference, November 1980, pp. 163-166.

[64]Pradhan, D. K. and K. Son, "The Effect of Untestable Faults in PLAs and a Design for Testability," Proceedings IEEE International Test Conference, November 1980, pp. 359-367.

[65]Cha, C. W., "A Testing Strategy for PLAs," Proceedings 15th Design Automation Conference, June 1978, pp. 326-331.

[66]Ostapko, D. L. and S. J. Hong, "Fault Analysis and Test Generation for Programmable Logic Arrays," Digest of Papers 8th International Symposium on Fault-Tolerant Computing, June 1978, pp. 83-89.

[67]Smith, J. E., "Detection of Faults in Programmable Logic Arrays," IEEE Trans. on Computers, Vol. C-28, No. 11, November 1979, pp. 848-853.

[68]Saluji, K. K., K. Kinoshita, and H. Fujiwara, "A Multiple Fault Testable Design of Programmable Logic Arrays," Digest of Papers 11th International Symposium on Fault-Tolerant Computing, June 1981, pp. 44-46.

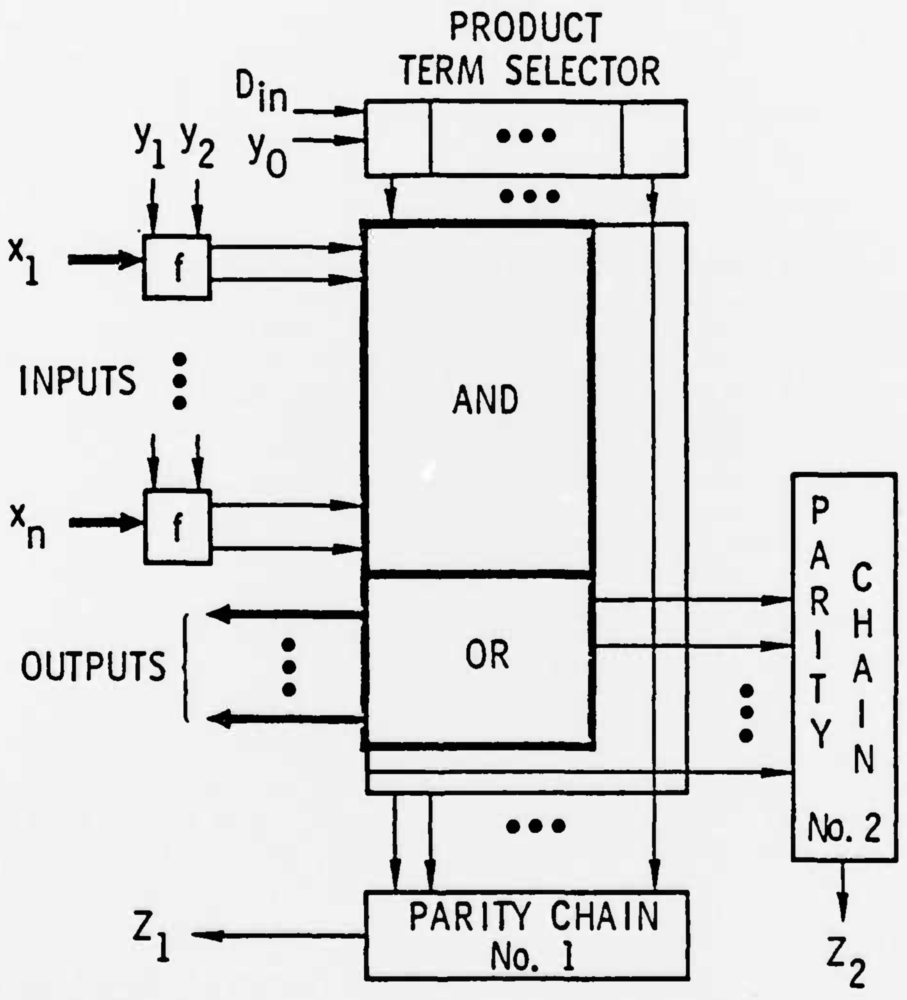


Fig. 10. Programmable Logic Array with Universal Test Set

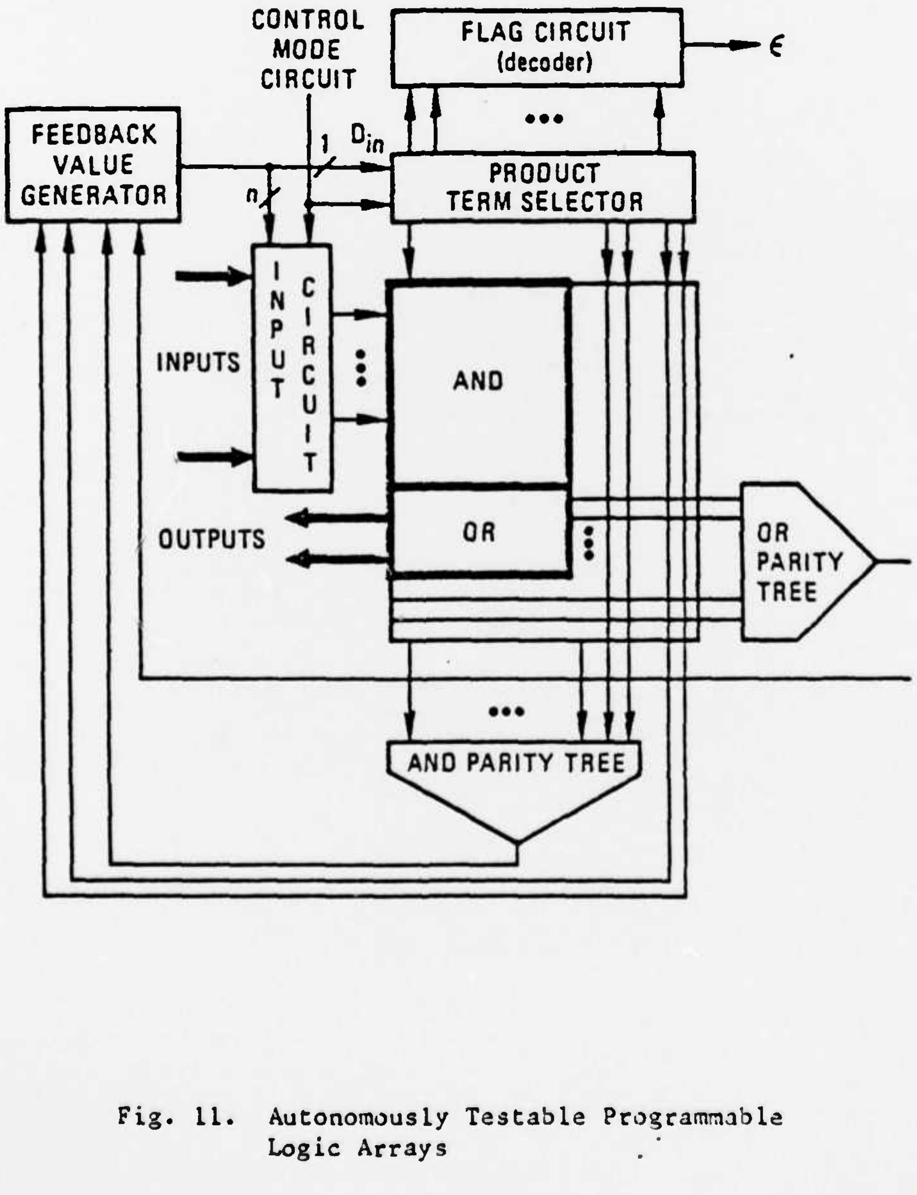


Fig. 11. Autonomously Testable Programmable Logic Arrays

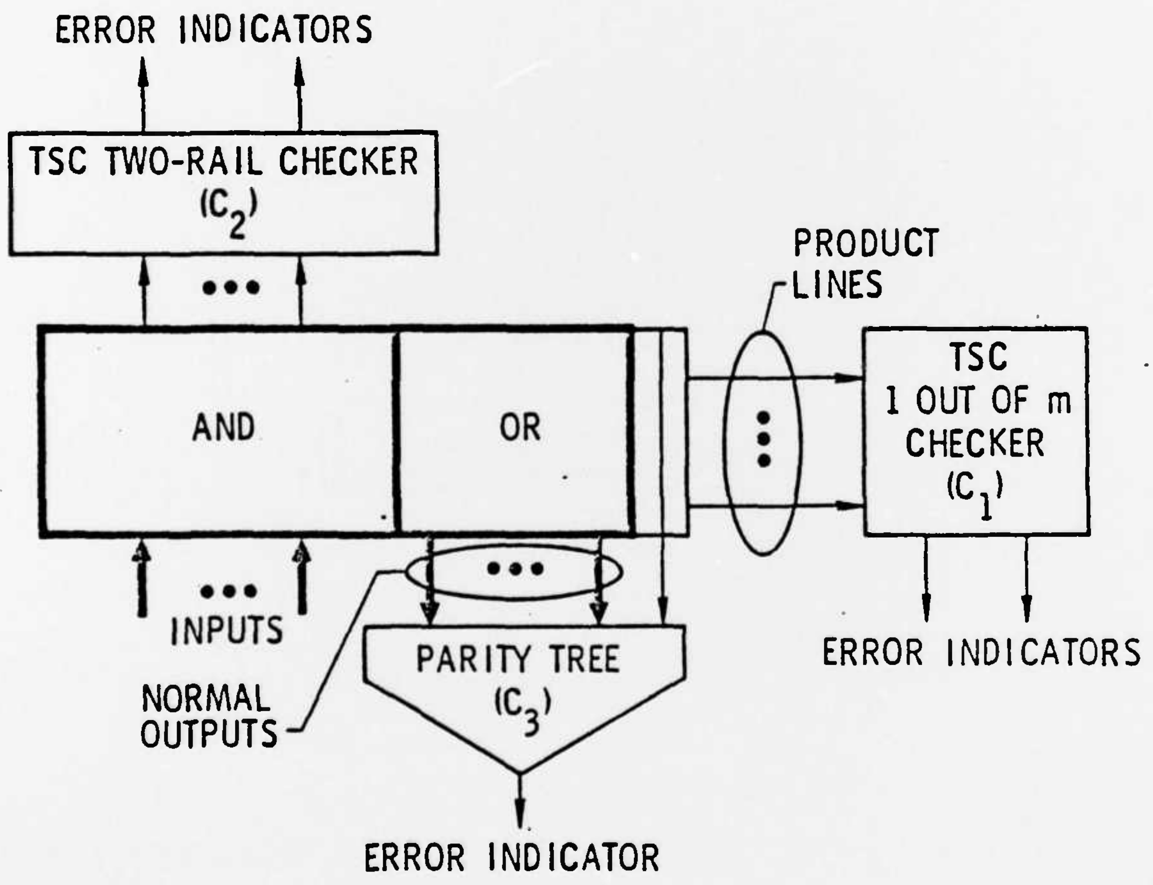


Fig. 12. Programmable Logic Array with Concurrent Error Detection and Testing

9.2 AUTONOMOUSLY TESTABLE PROGRAMMABLE LOGIC ARRAYS

Figure 11 indicates what is referred to as an autonomously testable structural PLA style.[61] This form of testing is very similar to the universal test set approach, except that rather than store the universal test set and have them applied via an automatic test equipment, the autonomous test approach generates the test patterns on-chip.

For this design, a product term selector register, several additional parity word and product term lines, and the parity chains have again been replaced by parity trees to enhance their testability.

The control for normal and test modes may still be external; however, the input test data and the data for D_{in} are now all generated on-chip by the feedback value generator which is a simple sequential circuit. At the end of the test process, the product term selector register contains a signature; it is decoded by the flag circuit which produces an error flag if a fault has been detected.

This approach employs $n + 2m + 8$ tests and detects all cross-point faults in the PLA as well as all single stuck-at faults in the entire circuit, except for parts of the feedback value generator and flag circuit. These can be duplicated if necessary.

9.3 PROGRAMMABLE LOGIC ARRAYS WITH CONCURRENT ERROR DETECTION AND TESTING

Figure 12 indicates a PLA testable structural style which supports concurrent error detection.[57] The PLA must be designed so that it has concurrent product lines, i.e., exactly one product term is true for every input vector. This condition usually increases the size of the PLA. Since the PLA inputs exist as a two-rail circuit (x_i, x'_i), a totally self-checking two-rail checker C_2 is used to detect stuck-at faults on input lines. A parity output word is added to the OR array, and a parity tree C_3 is used to detect errors on the outputs. Since concurrent testing is employed, a totally self-checking 1 out of m checker C_1 can be used to detect errors on the product lines.

During normal operation, this testable structural style will detect any of the following faults which produce output errors: single stuck-at faults, shorts between adjacent lines, and crosspoint faults. Most transient faults are also detected. Since it is possible that the normal inputs may not completely test C_1 and C_3, it may be necessary to carry out off-line testing so that these circuits can be completely tested.

10. SUMMARY

This report has presented a survey of some of the important concepts related to the design of a testable CLSIC. Both testing and design for testability have been discussed. Several design for testability concepts have been presented, with emphasis on structures for semi and fully built-in testing.

In addition, an approach to achieve testable designs has been suggested. In this approach, it is necessary to first partition a CLSIC into structures to be tested as separate entities. Some of these structures may be basic structures and have design styles. Often the characteristics inherent in a structure or its design style dictate a testing approach. The concept of a test strategy, consisting of off-line test generation, run-time test hardware, and built-in test structures for input and output accessibility, was introduced. Given a selected test strategy for a structure to be tested, a testable structural style is created. A testable chip thus consists of instances of testable structures, each of which corresponds to some testable structural style. The result of using these concepts in an orderly and effective way, satisfying the goals and constraints imposed by the design specifications, constitutes a testable design methodology.

REFERENCES

1. Eichelberger, E. B. and T. W. Williams, "A Logic Design Structure for LSI Testing," Proceedings of the 14th Design Automation Conference, June 1977, pp. 462-468.

2. Eichelberger, E. B. and T. W. Williams, "A Logic Design Structure for LSI Testability," J. Design Automation and Fault-Tolerant Computing, Vol. 2, May 1978, pp. 165-178.

3. Hayes, J. P., "Modeling Faults in Digital Logic Circuits," Rational Fault Analysis, R. Saeks and S. R. Liberty (eds.), Marcel Dekker, NY, 1977, pp. 78-95.

4. Nickel, V. V., "VLSI - The Inadequacy of the Stuck at Fault Model," Proceedings IEEE International Test Conference, November 1980, pp. 378-381.

5. Breuer, M. A. and A. D. Friedman, Diagnosis and Reliable Design of Digital Systems, Computer Science Press, Rockville, MD, 1976.

6. Hayes, J. P., Testing Logic Circuit by Transition Counting," Digest of Papers 5th International Symposium on Fault-Tolerant Computing, June 1975, pp. 215-222.

7. Savir, J., "Syndrome-Testable Design of Combinational Circuits," Digest of Papers 9th International Symposium on Fault-Tolerant Computing, June 1979, pp. 137-140.

8. Savir, J. "Syndrome-Testable Design of Combinational Circuits," IEEE Trans. on Computers, Vol. C-29, June 1980, pp. 442-451 (corrections: Nov. 1980).

9. Savir, J., "Syndrome-Testing of 'Syndrome-Untestable' Combinational Circuits," IEEE Trans. on Computers, Vol. C-30, August 1981, pp. 606-608.

10. "A Designer's Guide to Signature Analysis," Hewlett-Packard Application Note 222, Hewlett Packard, 5301 Stevens Creek Blvd., Santa Clara, CA 95050, April 1977.

11. Nadig, H. J., "Signature Analysis-Concepts, Examples and Guidelines," Hewlett-Packard Journal, May 1977, pp. 15-21.

12. Abramovici, M. and M. Breuer, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis," IEEE Trans. on Computers, Vol. C-29, June 1980, pp. 451-460.

13. Williams, T. W. and K. P. Parker, "Design for Testability--A Survey," Proceedings IEEE, Vol. 71, January 1983, pp. 98-112.

14. Hayes, J. P. and A. D. Friedman, "Test Point Placement to Simplify Fault Detection," Digest of Papers 10th International Symposium on Fault-Tolerant Computing, June 1973, pp. 73-78.

15. Hayes, J. P., "On Modifying Logic Networks to Improve their Diagnosability," IEEE Trans. on Computers, Vol. C-23, January 1974, pp. 56-62.

16. Akers, S. B., "Partitioning for Testability," J. Design Automation and Fault-Tolerant Computing, Vol. 1, February 1977, pp. 133-146.

17. Buehler, M. G. and M. W. Sievers, "Off-line, Built-in Test Techniques for VLSI Circuits," Computer, June 1982, pp. 69-82.

18. Funatsu S., N. Wakatsuki, and T. Arima, "Test Generation Systems in Japan," Proceedings of the 12th Design Automation Conference, June 1975, pp. 114-122.

19. Ando, H., "Testing VLSI with Random Access Scan," Digest of Papers COMPCON 80, February 1980, pp. 50-52.

20. Stewart, J. H., "Future Testing of Large LSI Circuit Cards," Digest of Papers IEEE Semiconductor Test Symposium, October 1977, pp. 6-17.

21. McCluskey, E. J. and S. Bozorgui-Nesbat, "Design for Autonomous Test," IEEE Trans. on Computers, Vol. C-30, November 1981, pp. 866-875.

22. Peterson, W. W. and E. J. Weldon, Error-Correcting Codes, 2nd Edition, MIT Press, Cambridge, MA, 1972.

23. Wakerly, J., Error Detecting Codes, Self-checking Circuits and Applications, American-Elsevier, NY, 1978.

24. Sedmak, R. M., "Design for Self-Verification: An Approach for Dealing with Testability Problems in VLSI-Based Designs," Proceedings IEEE Test Conference, 1979, pp. 112-120.

25. Konemann, B., J. Mucha, and G. Zwiehoff, "Built-in Logic Block Observational Techniques," Digest of Papers 1979 Test Conference, October 1979, pp. 37-41.

26. Mucha, J., "Hardware Techniques for Testing VLSI Circuits Based on Built-in Tests," Digest of Papers COMPCON 81, February 1981, pp. 366-369.

27. Agarwal, V. K. and E. Ceruy, "Store and Generate Built-in Testing Approach," Digest of Papers 11th International Symposium on Fault-Tolerant Computing, June 1981, pp. 35-40.

28. Susskind, A. K., "Testing by Verifying Walsh Coefficients," Digest of Papers 11th International Symposium on Fault-Tolerant Computing, June 1981, pp. 206-208.

29. McCluskey, E. J. and S. Bozorgui-Nesbat, "Design for Autonomous Test," Proceedings IEEE International Test Conference, November 1980, pp. 15-21.

30. Bozorgui-Nesbat, S. and E. J. McCluskey, "Structured Design for Testability to Eliminate Test Pattern Generation," Digest of Papers 10th International Symposium on Fault-Tolerant Computing, June 1980, pp. 158-163.

31. Seth, S. C. and K. L. Kodandapani, "Diagnosis of Faults in Linear Tree Networks," IEEE Trans. on Computers, Vol. C-26, January 1977, pp. 29-33.

32. Saluja, K. K. and R.M. Reddy, "Fault Detecting Test Sets for Reed-Muller Canonic Networks," IEEE Trans. on Computers, Vol. C-24, October 1975, pp. 995-998.

33. Fujiwara, H. and K. Kinoshita, "A Design of Programmable Logic Arrays with Universal Tests," IEEE Trans. on Computers, Vol. C-30, November 1981, pp. 823-828.

34. Friedman, A. D., "Easily Testable Iterative Systems," IEEE Trans. on Computers, Vol. C-22, December 1973, pp. 1061-1064.

35. Sridhar, T. and J. P. Hayes, "A Functional Approach to Testing Bit-Sliced Microprocessors," IEEE Trans. on Computers, Vol. C-30, August 1981, pp. 563-571.

36. Chang, H. Y. and S. W. Heimbigner, "LAMP: Controllability, Observability and Maintenance Engineering Technique (COMET)," Bell System Tech. J., Vol. 53, October 1974, pp. 1505-1534.

37. Goldstein, L. H. and E. L. Thigpen, "SCOAP: Sandia Controllability Observability Analysis Program," Proceedings of the 17th Design Automation Conference, June 1980, pp. 190-196.

38. Goldstein, L. H., "Controllability/Observability Analysis of Digital Circuits," IEEE Trans. Circuits and Systems, Vol. CAS-26, September 1979, pp. 685-693.

39. Grason, J., "TMEAS, A Testability Measurement Program," Proceedings of the 16th Design Automation Conference, June 1979, pp. 156-161.

40. Bennetts, R. G., et al., "CAMELOT: A Computer-Aided Measure for Logical Testability," IEE Proceedings, Vol. 128, Part E (Comp. & Digital Techniques), London, September 1981, pp. 177-189.

41. Chen, T-H. and M. A. Breuer, "Automatic Design for Testability via Testability Measures," IEEE Transactions on Computer-Aided Design, Vol. CAD-4, January 1985, pp. 3-11.

42. Eichelberger, E. B., et al., "A Logic Design Structure for Testing Internal Arrays," Proceedings of the 3rd USA-Japan Computer Conference, October 1978, pp. 266-272.

43. Hayes, J. P., "Detection of Pattern Sensitive Faults in Random Access Memories," IEEE Trans. on Computers, Vol. C-24, February 1975, pp. 150-157.

44. Goel, P., "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," Digest of Papers 10th International Symposium on Fault-Tolerant Computing, October 1980, pp. 145-150.

45. Roth, J. P., "Diagnosis of Automata Failures: A Calculus and a Method," IBM J. Research and Development, Vol. 10, July 1966, pp. 278-291.

46. Eichelberger, E. B. and E. Lindbloom, "A Heuristic Test-Pattern Generator for Programmable Logic Array," IBM J. Research and Development, Vol. 24, January 1980, pp. 15-22.

47. Lesser, J. D. and J. J. Shedletsky, "An Experimental Delay Test Generator for LSI Logic," IEEE Trans. on Computers, Vol. 29, March 1980, pp. 235-248.

48. Thomas, J. J., "Automated Diagnostic Test Programs for Digital Networks," Computer Design, Vol. 10, August 1971, pp. 63-67.

49. Thatte, S. M. and J. A. Abraham, "Test Generation for Microprocessors," IEEE Trans. on Computers, Vol. C-29, June 1980, pp. 429-441.

50. Bastin, D., et al., "Probabilistic Test Generation Methods," Digest of Papers 3rd International Symposium on Fault-Tolerant Computing, June 1973, p. 171.

51. Parker, K. P., "Adaptive Random Test Generation," J. Design Automation and Fault Tolerant Computing, Vol. 1, October 1976, pp. 62-83.

52. Hnatek, E. R., A User's Handbook of Semiconductor Memories, Wiley-Interscience, NY, 1977.

53. Hong, S. J. and D. L. Ostapko, "FITPLA: A Programmable Logic Array for Functional Independent Testing," Digest of Papers 10th International Symposium on Fault-Tolerant Computing, October 1980, pp. 131-136.

54. Friedman, A. D. and P. Menon, Fault Detection in Digital Circuits, Prentice Hall, NJ, 1971.

55. Wang, S. L. and A. Avizienis, "The Design of Totally Self-Checking Circuits Using Programmable Logic Arrays," Digest of Papers 9th International Symposium on Fault-Tolerant Computing, June 1979, pp. 173-180.

56. Mak, G. P., J. A. Abraham, and E. S. Davidson, "The Design of PLAs with Concurrent Error Detection," Digest of Papers 12th International Symposium on Fault-Tolerant Computing, June 1982, pp. 303-310.

57. Khakbaz, J. and E. J. McCluskey, "Concurrent Error Detection and Testing for Large Programmable Logic Arrays," CRC Report No. 81-14, Stanford University, Stanford, CA, September 1981.

58. Dong, H. and E. J. McCluskey, "Concurrent Testing of Programmable Logic Arrays," CRC Report No. 82-11, Stanford University, Stanford, CA, June 1982.

59. Daehn, W. and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays," IEEE Trans. on Computers, Vol. C-30, No. 11, November 1981, pp. 829-833.

60. Hassan, S. Z., "Testing PLAs Using Multiple Parallel Signature Analyzers," CRC Report No. 82-9, Stanford University, Stanford, CA, June 1982.

61. Yajima, S. and T. Aramaki, "Autonomously Testable Programmable Logic Arrays," Digest of Papers 11th International Symposium on Fault-Tolerant Computing, June 1981, pp. 41-43.

62. Dong, H. and E. J. McCluskey, "Design of Fully Testable Programmable Logic Arrays," CRC Report No. 82-20, Stanford University, Stanford, CA, June 1982.

63. Son, K. and D. K. Pradhan, "Design of Programmable Logic Arrays for Testability," Proceedings IEEE International Test Conference, November 1980, pp. 163-166.

64. Pradhan, D. K. and K. Son, "The Effect of Untestable Faults in PLAs and a Design for Testability," Proceedings IEEE International Test Conference, November 1980, pp. 359-367.

65. Cha, C. W., "A Testing Strategy for PLAs," Proceedings 15th Design Automation Conference, June 1978, pp. 326-331.

REFERENCES (Concluded)

66. Ostapko, D. L. and S. J. Hong, "Fault Analysis and Test Generation for Programmable Logic Arrays," Digest of Papers 8th International Symposium on Fault-Tolerant Computing, June 1978, pp. 83-89.

67. Smith, J. E., "Detection of Faults in Programmable Logic Arrays," IEEE Trans. on Computers, Vol. C-28, No. 11, November 1979, pp. 848-853.

68. Saluji, K. K., K. Kinoshita, and H. Fujiwara, "A Multiple Fault Testable Design of Programmable Logic Arrays," Digest of Papers 11th International Symposium on Fault-Tolerant Computing, June 1981, pp. 44-46.

END

FILMED

8-85

DTIC