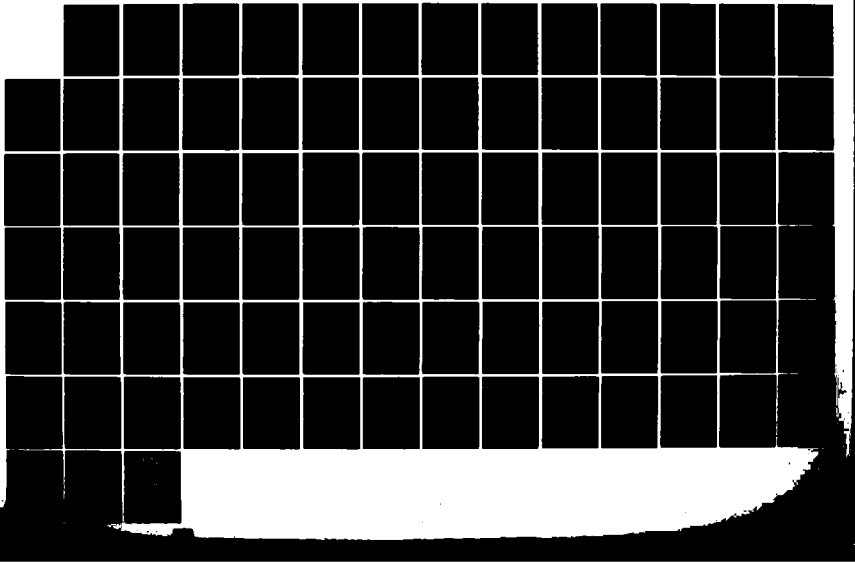
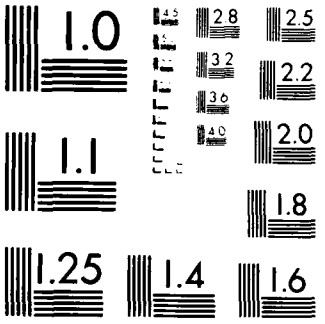
AD-A149 785 DESIGN OF A BIT-SLICED PROCESSOR ARRAY WITH BUILT-IN-SELF-TEST(U) ILLINOIS UNIV AT URBANA COMPUTER SYSTEMS GROUP P C NUI AUG 84 CSG-31 1/1

UNCLASSIFIED F/G 9/1 NL





REPORT CSG-31 JULY 1984

(2)

DESIGN OF A BIT-SLICED PROCESSOR ARRAY WITH BUILT-IN-SELF-TEST

PAUL CHUNHEI MUI

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CSG #31	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab. University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Semiconductor Research Corporation
6c. ADDRESS (City, State and ZIP Code) 1101 W. Springfield Ave. Urbana, Illinois 61801	7b. ADDRESS (City, State and ZIP Code) 300 Park Drive, Suite 215 P.O. Box 12053 Research Triangle Park, NC 27709
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Semiconductor Research Corp. 8b. OFFICE SYMBOL (If applicable) N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER SRC RSCH 83-01-014
8c. ADDRESS (City, State and ZIP Code) 300 Park Drive, Suite 215 P.O. Box 12053 Research Triangle Park, NC 27709	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: N/A PROJECT NO.: N/A TASK NO.: N/A WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) "Design of a Bit-Sliced Processor Array with Built-in Self-Test"	
12. PERSONAL AUTHOR(S) MUI, PAUL CHUNHEI	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) August 1984 15. PAGE COUNT 82
16. SUPPLEMENTARY NOTATION N/A	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Built-in Test, Self-Test, bit-sliced ALU, VLSI circuits

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

The overall objective of this report is to present an integrated approach to the design of bit-sliced processor arrays with built-in self-test. The conventional approach of making each bit-sliced processor chip self-testing is not used. Rather, a new approach of using an extra chip to test a processor array of any size and itself is used. The classical stuck-at fault model is not suitable for VLSI circuits. Rather, a functional level fault model is used. Each module of the processor array is tested exhaustively. The test responses of a fault-free processor array are made identical so that they can be easily monitored with no loss in fault coverage. The tester chip tests itself while it is testing the processor array. The fault coverages for both the tester chip and the processor array are high; the performance degradation is minimal; the area overhead is low, especially for large processor arrays; and the test length is short so tests can be performed more frequently. A VLSI design of the tester chip has been done with a 2-microns NMOS process.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE NUMBER (Include Area Code) 22c. OFFICE SYMBOL NONE

DESIGN OF A BIT-SLICED PROCESSOR ARRAY WITH BUILT-IN-SELF-TEST

BY

PAUL CHUNHEI MUI

B.S., University of Illinois, 1982

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1984

Urbana, Illinois

ACKNOWLEDGMENT

I would like to express my gratitude to Professor J. H. Patel for his assistance, technical guidance, and support in the production of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Motivation

Nowadays, in many applications, the use of computers that will not fail is a necessity. The computers used in processing transactions in the banking industry, the computers used to control the switching of telephone calls, the computers aboard the Space Shuttle, just to name a few, are the ones that cannot afford to fail. Hence, there are fault-tolerant computer systems: computers systems that will continue to function even if part of the system fails.

In recent years, the idea of a self-testing computer has become more popular. A self-testing computer is one that can test itself without any external equipment. Wakerly [Wak78] proposed a completely self-testing computer using modulo 15 residue code to check the CPU continuously for error, using duplication and comparison to check the microprogram sequencer continuously, and using parity code to check the ROM continuously. Sridhar and Hayes [Sri81] proposed another self-testing computer which requires a test mode in order to test itself. The test patterns that are required to test the CPU and the microprogram sequencer are stored in a ROM. During test mode, they are read out and applied to the CPU and the microprogram sequencer. The ROM itself is tested by duplication and comparison. Forbes et al. [For65] at IBM designed the DX-1, an experimental self-diagnosable computer in which the CPU is partitioned into two identical slices each capable of testing the other. Recently, Konemann et al. [Kon79] proposed the use of built-in pseudorandom test pattern generators and signature analyzers to make an individual bit slice self-testing. The overhead in a self-testing computer is always less than 50 percent since

duplication is not used in every module to make it self-testable. Therefore, it is essential to design computer modules such as processor, microprogram sequencer, etc., using VLSI technology that requires low overheads for self-testing requirements. Then, the total overhead of a self-testing computer would be low. A self-testing computer module has further advantages in that the test can be performed at circuit speed and the fault coverage is higher because of better controllability and observability.

1.2. Problem Statement

The goal of this research is to develop a processor with self-testing ability under the following requirements:

(1) Low overhead compared to the size of the processor required to provide the self-testing ability.

(2) Small or no degradation in performance of processor due to the incorporation of self-testing features.

(3) Fault coverage should be high.

(4) Test time should be short compared to the interval between testings.

1.2.1. Bit-Sliced Processor Array

A device U performing a set of operations on n-bit operands is said to be bit-sliced if a system that performs the same set of operations on Nn-bit operands can be constructed by interconnecting N identical copies of U in a regular way, as shown in Figure 1.1. The basic device U is called a cell or slice. The interconnection structure usually takes the form of a cascade, i.e., a one-dimensional iterative logic array (ILA). In a bit-sliced processor, the cell U performs the functions of the arithmetic-logic unit (ALU) and the register file of a

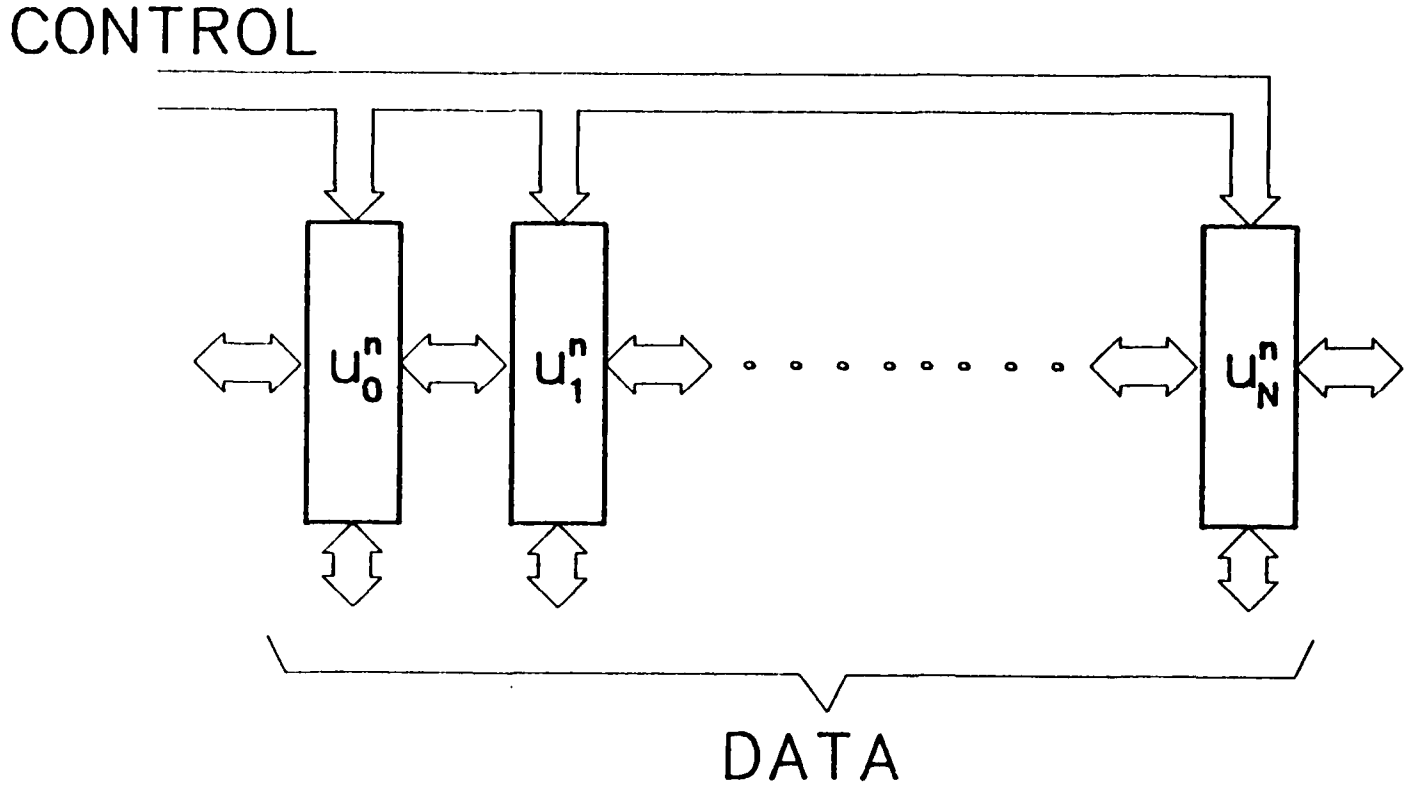


Figure 1.1 General Structure of ILA

computer [Fri73], [Kau67], [Par81].

The advantages of using a bit-sliced processor array are:

(1) Processors implemented with the same basic bit slice have a common set of control signals or microinstructions. Thus, bit-slicing helps in designing families of computers of different word sizes, but use the same basic software.

(2) The use of bit-sliced processors introduces structural simplicity and regularity in the interconnections between IC chips.

(3) The short word size of an individual processor slice permits the use of high-level circuit models and powerful functional fault models. Using these models it is possible to construct test sets of near-minimal size that are guaranteed to detect all faults of interest. Moreover, the regular interconnection structure that characterizes bit slicing simplifies testing of the entire bit-sliced array. The test length (time) is much shorter for testing a regularly interconnected bit-sliced processor array than for testing a single large processor cell in order to obtain the same percentage of fault coverage whether random or exhaustive test patterns are being used.

Therefore, the goal of this research is to develop a bit-sliced processor with self-testing ability which satisfies the above requirements.

1.3. Background

In this section a brief survey of testing techniques and previous work in this area of self-testing bit-sliced processor are presented.

1.3.1. Testing Techniques

1.3.1.1. Built-In-Test

Usage of integrated circuits requires periodic testing to ensure that they are functioning properly. Testing of integrated circuits involves the generation of specific test patterns for the particular circuit concerned and the monitoring of test responses. The traditional approach to this testing problem is to design a tester which can apply the test patterns to the integrated circuit under test (which can be a chip, a card, a board, etc.) and monitor the test responses. Test patterns can be generated by using Automatic Test Generation (ATG), which can give a certain percentage of fault coverage. A high percentage of fault coverage can be obtained by supplementing the automatically generated test patterns with manual test patterns. The problems with this approach are:

(1) Since the tester is used to test integrated circuits, the correct operation of the tester is necessary which means that the tester itself has to be tested by some other means.

(2) Automatic Test Generation works well for combinational circuits, but as the proportion of sequential circuits increases, Automatic Test Generation breaks down and the percentage of faults covered by automatically generated test patterns can be very low which means that manual test patterns have to be used in order to bring the fault coverage high. This will cause a long delay in the production cycle, and the work of writing manual test patterns can be very tedious.

Current development in the area of design for testability made the testing of integrated circuits easier. Testing techniques such as LSSD, Scan Path and Scan/Set Logic reduce the test pattern generation problem of a design which consists of both combinational and sequential circuits down to the test pattern generation of the combinational part only [Wil82]. But a tester is still needed to shift in the test patterns and monitor the test

responses. The test time can be long because of the time required to shift in and shift out the test patterns.

In recent years, a new testing technique called built-in-test (BIT) has become very popular. By adding extra circuits and logic to the original design, the modified design can now test itself without the use of an external tester. The functions of the added circuit are to generate test patterns to test the original circuit and possibly itself, and to monitor the test responses [Wil82], [McC81]. The attractiveness of this approach is that no external tester is needed, the circuit can test itself. The emerging of this testing technique comes from the fact that in today's technology, VLSI circuits are becoming cheaper and cheaper, so the addition of extra circuits to provide self-testing ability seems to give more advantages than disadvantages.

Most of the work in this area of built-in-test is on the chip level. Extra circuits are added to chips so that each chip can test itself. When the chip is in test mode, it can generate test patterns necessary to test itself and monitor the test responses. The success of the test is usually indicated by an output pin or pins at the end of the test. Not much work has been done in this area of built-in-test for higher levels of the hierarchy (levels above the chip level), i.e., designing an extra chip which can generate test patterns to test a certain number of other chips and itself, and, at the same time, monitor the test responses. This is exactly the area upon which this research is focused.

1.3.1.2. Faults in VLSI Circuits

Circuit level studies of physical failures of NMOS and CMOS circuits, which are two of the dominant technologies in VLSI today, reveal the existence of many non-stuck-at faults. A short or open on a line could have many effects that cannot be explained by a stuck-at fault model, and a physical failure in the circuit could affect a small area of the

chip and thus affect several gates. Circuit simulation of faulty logic devices indicates that a 5-valued logic algebra was needed to describe MOS circuit behavior under physical failure [Ban82]. Therefore, the classical fault model of describing physical failures as lines in the gate level description of the circuit stuck at zero or one is inadequate.

Therefore, for this research, a functional fault model is being used instead of the classical stuck-at fault model. Relatively complex functional units such as registers and multiplexers are treated as primitives. These primitives are tested for functional failures; faults affecting their internal lines are not explicitly considered.

1.3.1.3. Exhaustive Testing vs. Random Testing

Exhaustive testing of a device means that the device is tested with test patterns that are guaranteed to detect all possible faults in the device as long as the number of states does not increase. For example, a combinational device can be exhaustively tested by 2^N test patterns where N is the number of primary inputs of the device. Random testing of a device means that the device is tested by random test patterns which are subsets of the set of test patterns necessary to test the device exhaustively. Therefore, the fault coverage obtained from random test patterns is not as high as that obtained from exhaustive test patterns.

The simplicity of random testing, the ease of generation of random test patterns and the increasing size of circuits made random testing very attractive. Whether random or exhaustive testing should be used to test a particular device depends on the number of primary inputs to the device, the number of gates in the longest path between the primary inputs and the primary outputs, and the average fan-in which is the sum of fan-ins of all gates in the circuit divided by the number of gates [Agr78].

1.3.1.4. C-Testability

An ILA is C-testable if it can be tested by a constant number of test patterns, the length of which is independent of the size of the array [Sri81]. Therefore, test pattern generation only has to be done for a single cell, and the test patterns generated can be easily extended to test the whole array. ILA's that are not C-testable can be made C-testable by modifying the basic cell. An example is a non C-testable ILA that is made up of 1-bit incrementer cells, an extra input added to each 1-bit incrementer cell and tied together as a single input for the ILA would make it C-testable [Sri81].

1.3.1.5. Test Pattern Generation for Built-In-Test

For Built-In-Test, there are several methods of generating test patterns:

(1) A pseudo-random number generator (some form of linear-feedback registers) can be used to generate pseudo-random test patterns. If random testing is found to be more advantageous than exhaustive testing, then this method can be used to generate random test patterns.

(2) A counter can be used to generate test patterns to test a combinational device exhaustively. It cannot be used to generate patterns to test a sequential device completely because specific testing sequences are needed in order to test a sequential device completely. Large combinational modules should be broken down into smaller combinational modules with smaller numbers of primary inputs to make the use of counters to generate test patterns feasible.

(3) A ROM can be used to store test patterns that are needed to test a device or a group of devices in a chip. For sequential circuits, such as registers, which can only be tested properly by specific testing sequences, a ROM would be more suitable as a source of test patterns.

1.3.1.6. Test Response Verification for Built-In-Test

For Built-In-Test, there are several methods of monitoring test responses:

(1) Signature analysis is an economical way to deal with large amounts of test data and operating speed that are often required for testing digital systems. It works by compressing test data using simple compression algorithms. It can result in a considerable reduction of test data storage. The data compression algorithm used in signature analysis is based on the linear feedback shift registers (LFSRs). Beginning with all the registers in an initial state (typically all 0's), serial test data are shifted into the first register for a serial signature analyzer, or parallel test data are applied to the input of each register for a parallel signature analyzer. Test data are not simply shifted off the end of the signature analyzer but are shifted back in a way that depends on the feedback polynomial. This data compression algorithm is simple enough to be performed at high speeds. This process will compress the stream of test data to the length of the LFSR and form the signature which is the final contents of the registers. The signature can then be compared with the known correct signature to determine if there are faults in the circuit being tested.

As with any other data compression techniques, signature analysis allows some errors to go undetected. Hence, some circuit faults may go undetected. But with a carefully chosen primitive feedback polynomial, the percentage of errors, hence circuit faults, that go undetected can be made very low. There are signature analyzers with feedback polynomials that can detect all single errors, signature analyzers with feedback polynomials that are geared to detect certain types of burst errors, and signature analyzers with feedback polynomials that can detect certain types of errors due to repeated-use faults [Smi80], [Sri82].

(2) An equality checker is a device that can produce a zero as its output if all its inputs are equal, a one as its output if its inputs are not identical. They can be used to monitor test responses of a circuit which produces identical test responses when it is fault-free, but non-identical test responses when it is faulty. Equality checkers are most often used to monitor test responses of ILA.

I-Testability and CI-Testability:

An ILA is I-testable if the test responses from every cell of ILA can be made identical. Let T be a test set that tests an ILA completely under the single-cell fault assumption. The ILA is I-testable with respect to T if the expected responses to T appearing at the outputs of every cell of the ILA are identical. The input patterns forming T are called I-tests [Sri81].

An ILA is CI-testable if it is both C-testable and I-testable with respect to some test set. Let T be a test set that tests an ILA completely under single-cell fault assumption. The ILA is CI-testable with respect to T if the expected responses to T appearing at the outputs of every cell of the ILA are identical and the length of T is independent of the size of the ILA [Sri81].

Therefore, equality checkers are often used to monitor test responses of ILA's that are I-testable or CI-testable with respect to some test set. One advantage of the use of equality checkers to monitor test responses is that there is no loss in fault coverage due to the process of monitoring test responses, because there is no compression of test data.

1.3.2. Sridhar and Hayes' Design

Sridhar and Hayes proposed the design of a self-testing computer [Sri81]. The processor and microprogram sequencer are both made up of bit-sliced cells. The bit-sliced proces-

sor cell is similar to the AMD2901. The mapping ROM and control store are both duplicated to provide continuous testing. According to their paper, the bit-sliced processor is CI-testable with 168 test patterns, and the bit-sliced microprogram sequencer is I-testable with 499+2N test patterns where N is the microinstruction address length. Therefore, the test responses can be monitored by simple equality checkers. The test patterns for testing the bit-sliced processor and microprogram sequencer are stored in the control store, so they can be regarded as test programs at the microinstruction level.

There are two modes of operation for their computer: normal mode and test mode. During test mode, microinstructions are read out from the part of the control store where the test patterns are stored. These test patterns are then applied to the processor and the microprogram sequencer. This will continue until the last test pattern has been read and executed. If there are no error signals from the equality checkers, this means there are no faults in the circuit. Actually, the equality checkers themselves are also being tested by test patterns stored in the control store.

On first look, their self-testing computer seems to be a good design and every component is tested thoroughly. But, on closer look, their design is not workable at least for the processor part.

Careful studies show that for a bit-sliced processor to be CI-testable, not only do specific control signals have to be applied to the bit-sliced processor during testing, but specific data signals have also to be applied to the bit-sliced processor during testing. The shifter and the ALU are not CI-testable or even C-testable unless specific data patterns are applied to the bit slices of the processor. Since data signals come from the memory of the computer, this means specific test data patterns have to be stored in the memory and there have to be some ways to read them out during testing. But the word in memory to be read out during a read cycle is controlled by the contents of the external instructions. In Sridhar

and Hayes' design, the test patterns, which are control signals for the processor, and which are stored in the control store as microinstructions, have no control over which word in memory should be read out during the next read cycle. Therefore, their design would not work unless external instructions which specify the address of the word in memory to be read out are also used. Then, in this case, the test patterns would be made up of both external instructions and microinstructions.

1.4. Research Overview

The overall objective of this research is to present an integrated approach to the design of a self-testing bit-sliced processor array. The self-testing mechanism is not built into each bit-sliced processor chip, but is built into the whole bit-sliced processor array. This research is based on a bit-sliced processor chip similar to the commercially available AMD2901. The approach can be easily extended to other types of bit-sliced processor chips.

In Chapter 2, the algorithms used in making a bit-sliced processor array self-testing are presented. The processor array is made C-testable and I-testable without having to make any modifications on any modules of the bit-sliced processor chip. Exhaustive testing is used as opposed to random testing for higher fault coverage consideration. The test patterns are stored in a ROM as opposed to the use of counters and pseudo-random number generators. The test responses are monitored by equality checkers to avoid any loss in fault coverage due to the compression of test responses. The reasons for not using the conventional approach of making each bit-sliced processor chip self-testing are discussed. The advantages of using an extra tester chip to test a cascade of bit-sliced processor chips are presented. The algorithms used to make the tester chip able to test itself are also presented. Functional fault models suitable for the current VLSI technology are discussed. The operations of the bit-sliced processor array in normal and test modes are presented. Finally,

performance degradation due to the incorporation of self-testing features is discussed. The bit-sliced processor array with built-in-test is illustrated in Figure 1.2.

In Chapter 3, the design of the tester chip using an NMOS process is discussed. The timing is based on a two-phase clock. The area of the chip, the area of each module and the speed of each module are also presented. A discussion of area overhead to provide the self-testing ability is included.

In Chapter 4, some design experiences acquired during this research are discussed. Also presented are the extensions of the algorithms developed for other types of ILAs.

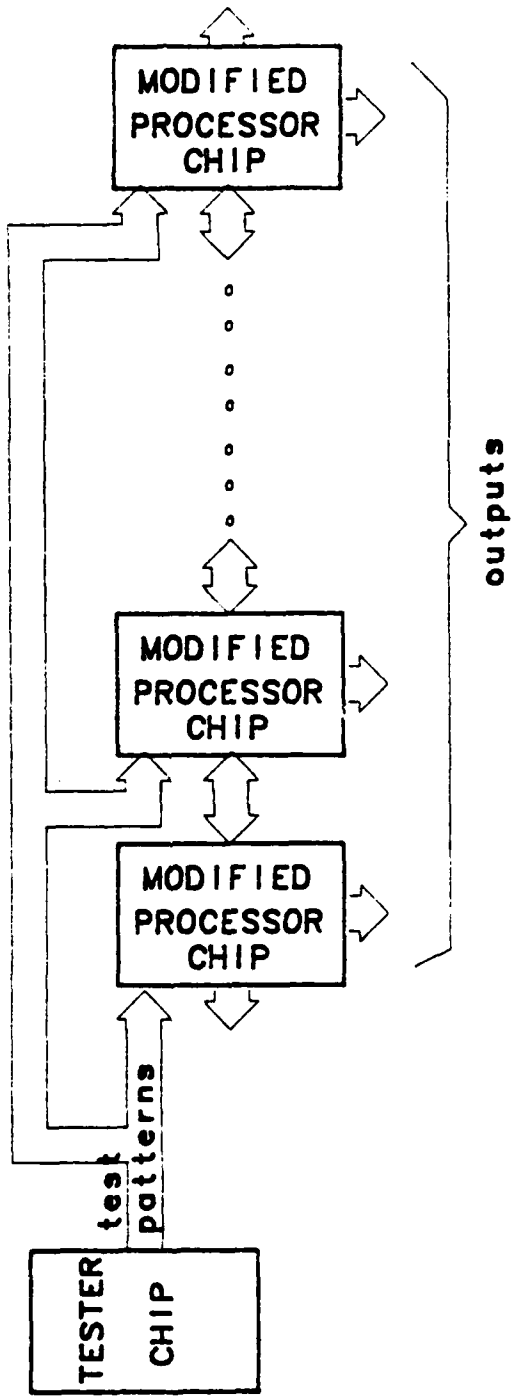


Figure 1.2 Overall Built-In-Test Scheme

CHAPTER 2

APPROACH FOR BUILT-IN-TEST IN BIT-SLICED PROCESSOR ARRAY

2.1. Basic Cell Model

The bit-sliced processor chip on which our research is based is similar to the commercially available AMD2901 which is a four-bit slice [AMD76]. The block diagram of our four-bit processor chip is shown in Figure 2.1.

The circuit is a four-bit slice cascadable to any number of bits. Therefore, all data paths within the circuit are four bits wide.

Data in any of the four words of the Random Access Memory (RAM) can be read from the A-port of the RAM as controlled by the four-bit A address field input. Likewise, data in any of the four words of the RAM as defined by the B address field input can be simultaneously read from the B-port of the RAM. The same code can be applied to the A select field and B select field in which case the identical file data will appear at both the RAM A-port and B-port outputs simultaneously. When enabled by the write enable (RAM EN), new data are always written into the file (word) defined by the B address field of the RAM.

The RAM data input field is driven by a shifter. The shifter scheme allows the data to be shifted up one bit position, shifted down one bit position, or not shifted in either direction. The RAM A-port data outputs and RAM B-port data outputs drive separate four-bit latches. These latches hold the RAM data during phase-two. This eliminates any possible race conditions that could occur while new data are being written into the RAM.

The high-speed Arithmetic Logic Unit (ALU) can perform three binary arithmetic and five logic operations on the two four-bit input words R and S. Both R and S are

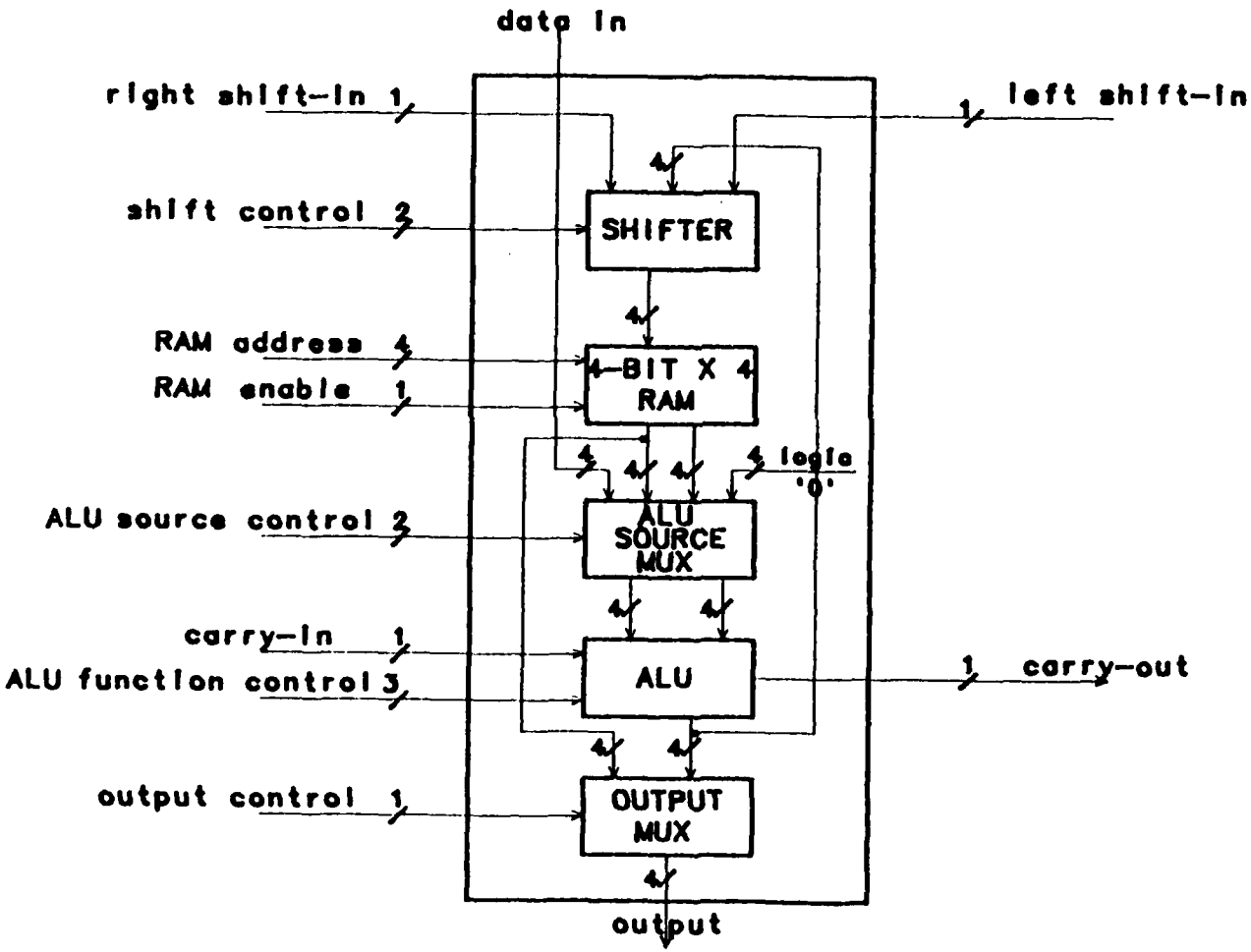


Figure 2.1 Block Diagram of Four-Bit Processor Chip

outputs of a four inputs to two outputs multiplexer; the four inputs are: outputs of A-latches, outputs of B-latches, external data inputs (D) and logic 0. This multiplexer scheme gives the capability of selecting various pairs of the A, B, D and "0" inputs as source operands to the ALU. The microinstruction inputs used to select the ALU source operands are the I_0 and I_1 inputs. The D input is the four-bit wide direct data field input. This port is used to insert all data into the working registers inside the device. Likewise, this input can be used in the ALU to modify any of the internal data files.

The I_2, I_3 and I_4 microinstruction inputs are used to select the ALU function. The ALU data output is routed to several destinations. It can be a data output of the device and it can also be stored in the RAM.

A two-input multiplexer is also used at the data output such that either the A-port of the RAM or the ALU outputs (F) are selected at the device Y outputs. This selection is controlled by the I_8 microinstruction inputs.

As was discussed previously, the RAM inputs are driven from a shifter. This allows the ALU outputs to be entered non-shifted, shifted up one position (multiply by 2) or sifted down one position (divide by 2). The shifter has two tri-state ports labeled RI and LI. Either one of these two ports will act as an input port while the other one will act as an output port depending on whether the shifter is in shift up or shift down mode. Both of these two ports will be in high impedance state when the shifter is in no shift mode.

The clock input controls the RAM and the A and B data latches. During phase-one, the A and B latches are open and will pass whatever data is present at the RAM outputs. During phase-two, the latches are closed and will retain the last data entered. If the RAM-EN is enabled, new data will be written into the RAM file (word) defined by the B address field during phase-two.

A bit-sliced processor array based on our four-bit slice has the ILA structure depicted in Figure 2.2. To allow arithmetic operations to be extended to operands of arbitrary length, neighboring cells communicate via carry (borrow) signals. Each cell generates a carry output signal CO which can be connected to the carry input line CI of the cell to its right. This allows ripple carry propagation through the entire array. Similar left-shift and right-shift connections between adjacent cells allow shift operations to take place across the ILA. No communication between cells is needed by the logical operations.

In order to obtain a manageable yet reasonably realistic processor cell model, and to make the use of the powerful functional fault model feasible, the operand size of the cell is limited to one bit. In other words, the functional fault model used in this research, the test set generated, etc., are based on a one-bit processor slice. But the test set and all the results obtained from this research are applicable to wider processor slices [Sri81]. The block diagram of our one-bit processor slice C and its microinstruction control fields are shown in Figure 2.3 and Table 2.1.

2.2. Functional Fault Model

For testing purposes, C is treated as a network of small register-level modules as depicted in Figure 2.3. The modules are regarded as black boxes whose input-output behavior is completely defined. For example, components like multiplexers and registers are treated as primitive modules in our analysis. The internal structure of these modules is not considered. Corresponding to this functional view of primitive module behavior, we now define a fault model based on functional considerations.

Let M be a primitive combinational or synchronous sequential logic module in a circuit U under test. Let z denote the function realized by M and let s be the number of internal states of M; s = 1 if M is combinational. A malfunction F of M is called a (functional)

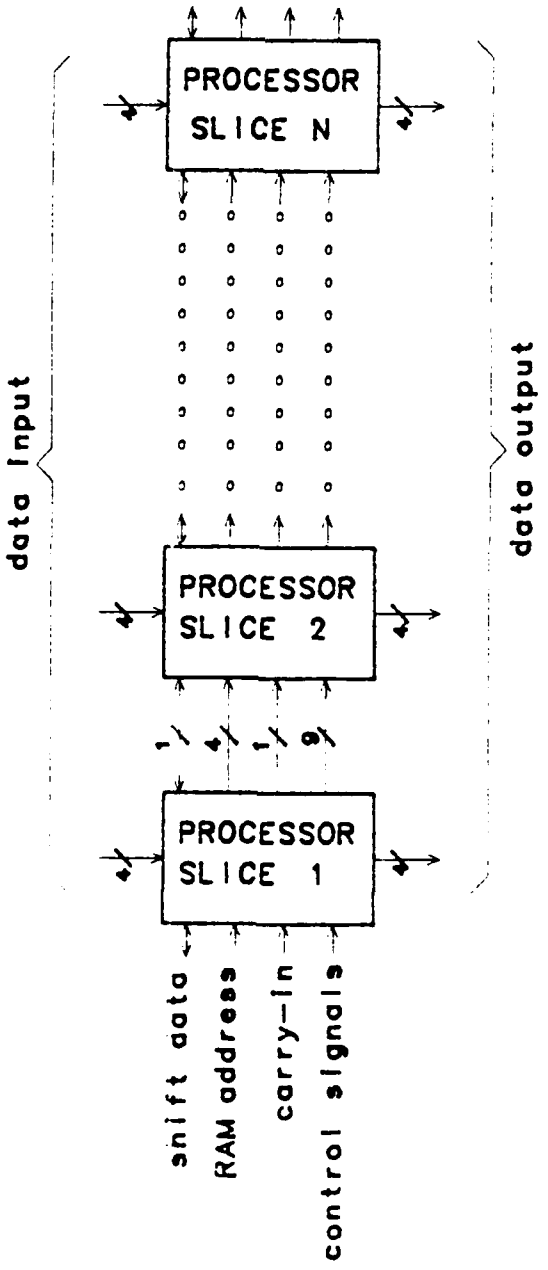


Figure 2.2 Structure of Bit-Sliced Processor Array

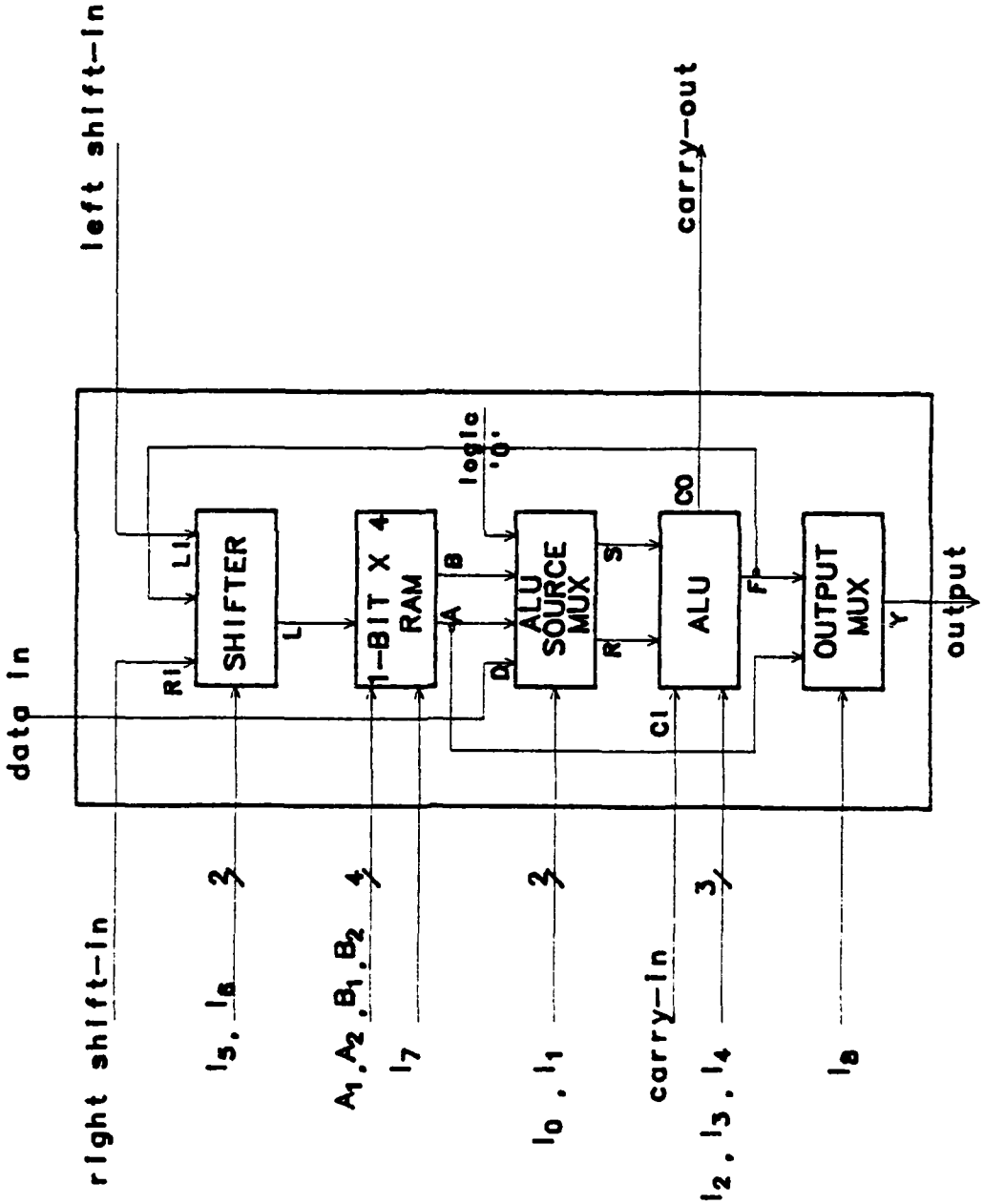


Figure 2.3 Block Diagram of One-Bit Processor Slice

Table 2.1 Microinstruction Control Fields of Processor

I_0	I_1	R	S
0	0	A	B
0	1	A	0
1	0	D	B
1	1	D	0

ALU source control

I_5	I_6	L
0	0	F
0	1	RI
1	0	LI
1	1	not used

shift control

I_7	
0	disable RAM
1	enable RAM

RAM control

I_8	Y
0	F
1	A

output control

I_2	I_3	I_4	F
0	0	0	R plus S
0	0	1	S minus R
0	1	0	R minus S
0	1	1	R OR S
1	0	0	$\underline{R}$ AND S
1	0	1	$\overline{R}$ AND S
1	1	0	R EXCLUSIVE-OR S
1	1	1	R EXCLUSIVE-NOR S

ALU function control

fault of M if f permanently changes M to a module M^F realizing z^F, where $z \neq z^F$ and the number of states s^F of M^F is not greater than s.

Thus, faults in a combinational module can induce arbitrary changes in the truth table of the module but cannot convert it into a sequential circuit. To detect these faults, it is necessary and sufficient to apply all 2^N input vectors to an N-input module. This fault model is relatively powerful. It includes as a proper subset all single and multiple faults of the standard stuck-line fault model. The restriction excluding sequential behavior appears to be relatively minor.

When M is a sequential circuit, we allow faults to cause any change in the state table of M that does not increase the number of states. This is quite realistic in the case of modules in which there are k binary memory elements and exactly 2^k states; only sequential modules of this type will be considered.

Based on physical failures in memory, like metallization shorts and capacitive coupling, the following functional fault model is proposed for the memory cell array [Tha77].

(1) One or more cells are stuck at 0 or 1.

(2) There exist one or more pairs of cells which are coupled. By this we mean that a transition from x to y in one cell of the pair, say cell i, changes the state of the other cell, say cell j, from x to y or from y to x, where $x \in \{0, 1\}$ and $y = \bar{x}$. This, of course, does not necessarily imply that a similar transition in cell j will influence cell i in a similar manner. (Note that we allow arbitrary pairs of cells with such coupling. Thus, a cell k could be coupled to cell i or cell j in another coupled pair.)

It is further assumed that only one module in the circuit U is faulty at any time. This single fault assumption is included in most fault models. It is justified if the module failures are independent, and if U is tested frequently.

It should be emphasized that the foregoing model will only be applied to n-input s-state modules where n and s are relatively small. The modules of the cell C, for example, are all of this type. The small size of the modules is necessary to make practical the essentially exhaustive testing methods required by the fault model. Although individual modules are tested exhaustively, networks of these modules are tested in an efficient nonexhaustive manner.

2.3. Testing Algorithm

The conventional approach of built-in-test is to put extra circuits into each bit-sliced processor chip to make it self-testing. The approach we are using in this research is to use an extra chip (tester chip) which can generate test patterns to test a cascade of bit-sliced processor chips (independent of the size of the array) and monitor the test responses. The tester chip can test itself also.

The advantages of this approach over the conventional approach are:

(1) Circuit overhead and area overhead required to provide the self-testing ability are lower because for our approach, one extra tester chip is required to test a bit-sliced processor array of any size. The size of the tester chip is about the same as the size of each bit-sliced processor chip. The modification on each bit-sliced processor chip is minimal and results in very little area overhead. Therefore the percentage of area overhead required to provide the self-testing ability becomes smaller as the size of the processor array increases since the major area overhead is the extra tester chip. For the conventional approach, the percentage of area overhead is independent of the size of the processor array since each bit-sliced processor chip has the same percentage of area overhead required for self-testing.

(2) For our approach, there is very little performance degradation resulting from the addition of extra circuits to provide the self-testing ability because in normal operation, the tester chip is isolated and is transparent to the processor array. The small performance degradation comes from the fact that each normal D-input has to go through a two-to-one multiplexer whose function is to choose between normal and test D-inputs before arriving at the input of the ALU source multiplexer.

For the conventional approach, each bit-sliced processor chip has to be isolated from each other during testing because each processor chip is supposed to test itself independently of the others. Therefore, all the connections between bit-sliced processor chips have to be broken during testing, possibly using multiplexers. The left-shift and right-shift inputs/outputs have to be separated from the left-shift and right-shift inputs/outputs of neighboring chips by multiplexers. The carry-in input has also to be separated from the carry-out output of the neighboring chip by multiplexer. During normal operation, these multiplexers will let signals from neighboring cells to pass through. During test mode, these multiplexers will cause each chip to isolate from each other and will allow signals from the test pattern generation circuits within the chip to pass through.

The problem is that during normal operation, signals that connect the bit slices together like the shift inputs/outputs, carry-in inputs and carry-out outputs have to go through extra circuits of multiplexers that are used to separate the bit slices. This will introduce delay in the propagation of these signals since multiplexers are made up of pass transistors. This problem becomes more serious for the critical carry chain since carry signals ripple through the entire processor array in the bit-sliced architecture. Therefore, for the conventional approach, there is performance degradation due to the addition of extra circuits to provide the self-testing ability. The larger the size of the processor array, the larger the degree of performance degradation.

(3) For our approach, all the connections between bit slices are tested since all bit slices are connected in the same fashion in test mode as they are in normal mode.

For the conventional approach, no connections between bit slices are tested because all bit slices are isolated from each other during testing. The multiplexers used to separate the bit slices from each other, as previously discussed, cannot be tested for their correct operation in normal mode.

Based on the functional fault model described earlier, each module of the basic cell (Figure 2.3), namely, shifter, RAM, ALU source multiplexer, ALU, output multiplexer are tested exhaustively. But the basic cell itself is not tested exhaustively. It is necessary and sufficient to test a combinational module with 2^N test patterns where N is the number of its primary input lines. For the RAM, the memory functional fault model described earlier can be used in constructing the test set. Therefore, the test patterns for the shifter, the ALU source multiplexer, the ALU and the output multiplexer which are combinational modules can be easily generated by counters. But the test patterns for the RAM which is a sequential module cannot be generated by a counter and have to be stored in a ROM because a sequential module needs a specific sequence of test patterns in order to be tested completely.

A set of four counters and a ROM in the tester chip can be used to generate test patterns for a processor array of any size. The problem with this approach is that the test patterns for a combinational module generated by a counter are valid and can test the combinational module exhaustively if and only if all the primary inputs of the module are also the primary inputs of the basic cell. Otherwise, each test pattern T for the combinational module has to be transformed to a test pattern T^* which, when applied to the primary inputs of the basic cell, will cause T to apply to the primary inputs of the combinational module. But now the test pattern T^* may not be able to be generated by a regular counter.

Therefore, for this research, a ROM is used which contains test patterns to exhaustively test each module, namely, the shifter, the ALU source multiplexer, the RAM, the ALU and the output multiplexer. A pseudo-random number generator is not used because the fault coverage obtained from random test patterns is not high enough and is dependent on some properties of the circuit under test as discussed in Chapter 1.

In Chapter 1, different methods of verifying test responses were discussed. For this research, the bit-sliced processor cells are made CI-testable so that test responses can be monitored by simple equality checkers. The Y output from each bit slice is monitored by equality checkers built into each processor chip. The CO output of the chip is also monitored by an equality checker built into the chip whose other input is the CI input of the chip. Signature analyzers are not used because there is a loss in fault coverage due to the compression of test data. Signature analysis has a further disadvantage that the signature is dependent on the size of the processor array. A more detailed block diagram of the processor array with the tester chip is shown in Figure 2.4.

2.4. Test Generation for a Single Cell

The basic processor cell is divided into 5 modules. The shifter, the ALU source multiplexer, the ALU and the output multiplexer are combinational while the RAM is sequential. The primary inputs of the basic processor cell are:

I_0, I_1 – control inputs of the ALU source multiplexer

I_2, I_3, I_4 – control inputs of the ALU

I_5, I_6 – control input of the shifter

I_7 – RAM enable

I_8 – control input of the output multiplexer

A_1, A_2 – A address field of RAM

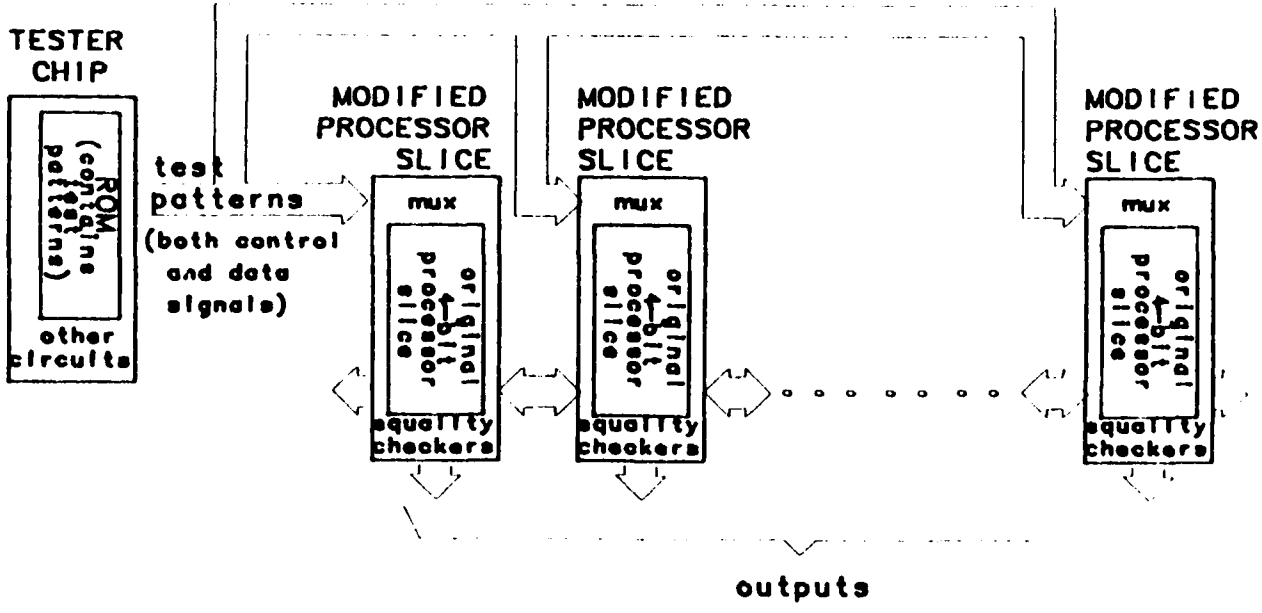


Figure 2.4 More Detailed Built-In-Test Scheme

B_1, B_2 – B address field of RAM

RI – right shift-in/shift-out of shifter

LI – left shift-in/shift-out of shifter

D – external data input

CI – carry-in input

The output of every module is observable at the Y and/or CO outputs of the cell.

Each test pattern T^* should have the following properties:

(1) Application of T^* to the primary inputs of the basic cell will cause test pattern T to be applied to the primary inputs of the module intended to be tested by this test pattern, where T is one of the test patterns required to test this module exhaustively.

(2) Application of T^* to the primary inputs of the basic cell will propagate the outputs of the module being tested to the observable outputs of the basic cell.

2.4.1. The Shifter

Since the shifter has five primary inputs, the test set T(SH) has thirty-two members. From T(SH) we can easily construct T^*(SH). For example, suppose that both control lines I_5 and I_6 of the shifter are set to 0, so that the output L of the shifter is F. There are eight tests in T(SH) of the form (I_5, I_6, RI, F, LI) = (0, 0, d, d, d), where d denotes DON'T CARE. To extend these eight tests to the corresponding members of T^*(SH), we need to define suitably all the other primary inputs of C. The input D may be selected so that the desired F input required in the next test for the shifter is generated. The other primary inputs of C are kept at a constant value throughout the testing of the shifter. These constant values are chosen to permit the signal L to be observed at the primary output Y. One way of doing this is to write signal L into the RAM, then propagate it to the output by setting

$(I_0, I_1, I_2, I_3, I_4, I_5) = (1, 0, 1, 1, 0, 0)$. Table 2.2 shows the resulting sequence of eight test patterns applied to the shifter by $T^*(SH)$. A similar sequence of eight tests can be derived for each of the other three combinations of I_5 and I_6. Hence, there exists a test sequence $T^*(SH)$ for the shifter of length thirty-two.

2.4.2. The RAM

The set of necessary and sufficient conditions for all faults in the stated memory fault model to be detected by a test sequence are listed below. In the following discussion, a forced transition of a cell is defined as one that is initiated by the testing algorithm by writing into the cell; this may cause transitions in other cells because of coupling.

Condition 1. Each cell must undergo

(a) a 0 - 1 transition,

(b) a 1 - 0 transition

and must be read after each transition, before undergoing any subsequent forced transitions.

Condition 2. For every pair of cells (i,j), cell i (i.e., the cell with address i) must be read after cell j makes a forced transition and before cells i and j make any further forced transitions for the following states of cell i and transitions in cell j:

(a) cell i in state 0, cell j making a 0 - 1 transition,

(b) cell i in state 1, cell j making a 0 - 1 transition,

(c) cell i in state 0, cell j making a 1 - 0 transition,

(d) cell i in state 1, cell j making a 1 - 0 transition.

The sequences of the algorithm which can detect all the faults in the stated fault model can be found in Table 1 of [Nai78]. This algorithm takes thirty-two vectors to test

Table 2.2 Generation of T(SH) from $T^{'}$(SH)

Test No.	I_5	I_6	RI	F	LI	D
1	0	0	0	0	0	0
2	0	0	0	1	0	1
3	0	0	1	0	0	1
4	0	0	1	1	0	1
5	0	0	0	0	1	1
6	0	0	0	1	1	1
7	0	0	1	0	1	1
8	0	0	1	1	1	1

Table 2.3 Generation of T(SM) from $T^{'}$(SM)

Test No.	I_0	I_1	A	B	D	A_1	A_2	B_1	B_2
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0	0	1
4	0	0	0	1	1	0	0	0	1
5	0	0	1	0	0	0	1	0	0
6	0	0	1	0	1	0	1	0	0
7	0	0	1	1	0	0	1	0	1
8	0	0	1	1	1	0	1	0	1

note: logic 0 was stored in address 00
and logic 1 was stored in address 01
prior to this test

the 4 x 1 bit RAM.

The latches are tested while the RAM is being tested; therefore, no special test patterns are required to test the latches.

2.4.3. The ALU Source Multiplexer

The ALU source multiplexer has three data inputs and two control inputs; therefore, it can be tested exhaustively by thirty-two tests. The two control inputs and the external data input are primary inputs of the basic cell; therefore, they can be set to any logic values by T^*(SM). The other two data inputs of the ALU source multiplexer, which are outputs of the A-latch and the B-latch, can be controlled by storing appropriate logic values in different addresses of the RAM. By specifying appropriate A address field and B address field of the RAM, the outputs of the A-latch and the B-latch can be set to any logic values. Therefore, T^*(SM) can be constructed from T(SM). Table 2.3 shows the resulting sequence of eight test patterns applied to the ALU source multiplexer by T^*(SM). A similar sequence of eight tests can be derived for each of the other three combinations of I_0 and I_1. Hence, the ALU source multiplexer can be exhaustively tested in thirty-two tests.

Since the ALU source multiplexer has two inputs, any fault exercised by the test set can either cause one of the outputs to switch from the the correct logic value or both outputs to switch from the correct logic value. In order to detect faults in the ALU source multiplexer, the following should be done to propagate the outputs of the ALU source multiplexer to the primary outputs of the basic cell:

(1) If the expected R and S outputs are 00, the ALU should perform an OR operation on these two operands.

(2) If the expected R and S outputs are 0 and 1, respectively, the ALU should perform an S - R operation on these two operands with the borrow-in equal to zero.

(3) If the expected R and S outputs are 1 and 0, respectively, the ALU should perform an R - S operation on these two operands with the borrow-in equal to zero.

(4) If the expected R and S outputs are 11, the ALU should perform an AND operation on these two operands.

2.4.4. The ALU

The ALU has three data inputs, three control inputs; therefore, it can be tested exhaustively by sixty-four tests. The three control inputs and the carry-in input are primary inputs of the basic cell; therefore, they can be set to any logic values by T^*(ALU). The other two data inputs of the ALU which are outputs of the ALU source multiplexer can be set to any logic values by controlling the external data input D and the B address field of the RAM. By setting (I_0, I_1) to (1, 0) and by storing appropriate logic values in different addresses of the RAM, the R and S outputs of the ALU source multiplexer can be made dependent on the D-input and the B address field of the RAM. Therefore, T^*(ALU) can be constructed from T(ALU). Table 2.4 shows the resulting sequence of eight test patterns applied to the ALU by T^*(ALU). A similar sequence of eight tests can be rived for each the other seven combinations of I_2, I_3 and I_4. The outputs of the ALU can be propagated to the primary outputs of the basic cell by setting $I_8 = 0$ throughout the testing of the ALU. Hence, the ALU can be exhaustively tested in sixty-four tests.

Table 2.4 Generation of T(ALU) from T^*(ALU)

Test No.	I_2	I_3	I_4	R	S	CI	B_1	B_2	D
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0
3	0	0	0	0	1	0	0	1	0
4	0	0	0	0	1	1	0	1	0
5	0	0	0	1	0	0	0	0	1
6	0	0	0	1	0	1	0	0	1
7	0	0	0	1	1	0	0	1	1
8	0	0	0	1	1	1	0	1	1

note: logic 0 was stored in address 00
and logic 1 was stored in address 01
prior to this test

Table 2.5 Generation of T(OM) from T^*(OM)

Test No.	I_8	A	F	A_1	A_2	D
1	0	0	0	0	0	0
2	0	0	1	0	0	1
3	0	1	0	0	1	0
4	0	1	1	0	1	1

note: logic 0 was stored in address 00
and logic 1 was stored in address 01
prior to this test

2.4.5. The Output Multiplexer

The output multiplexer has two data inputs, one control input; therefore, it can be tested exhaustively by eight test patterns. The control input is a primary input of the basic cell; therefore, it can be set to any logic values by $T^*(OM)$. The two data inputs of the output multiplexer are outputs of the ALU and the A-latch, respectively. If appropriate logic values are stored in different addresses of the RAM, the output of the A-latch can be set to any logic values by controlling the A address field of the RAM. The output of the ALU can also be set to any logic values by controlling the external data input D and by setting $(I_0, I_1, I_2, I_3, I_4)$ to (1, 1, 0, 1, 1). Table 2.5 shows the resulting sequences of four test patterns applied to the output multiplexer by $T^*(OM)$. A similar sequence of four tests can be derived for the other combination of I_8. Hence, the output multiplexer can be exhaustively tested in eight tests.

2.5. Test Generation for the Processor Array and CI-Testability

A linear cascade of N copies of the 1-bit cell C is required to expand the word size of the operands being processed to N bits. For shift operations, the Y output of each cell is connected to the left shift-in line of the cell on its left and also to the right shift-in line of the cell on its right. The ILA thus executes the same operations as the basic cell C using N-bit instead of 1-bit operands. The fault model for an individual cell is the same as before with at most one cell in the array assumed to be faulty.

Each module of the basic cell can be made CI-testable with respect to some test set. All modules of the same type in the processor array can be tested by a constant number of tests independent of the size of the processor array, and the test outputs of the modules can be propagated to the primary outputs of the processor array in such a way that:

(1) The test outputs of the modules do not become masked in propagating to the outputs of the processor array.

(2) All test outputs from the processor array that have to be verified are identical so that they can be monitored by equality checkers.

2.5.1. The Shifter

To apply the test patterns obtained in the previous section for the shifter of a single processor cell to any shifter cells in a processor array, the desired RI or LI input signal must be generated at the Y output of the left cell $C_{(i-1)}$ or right cell $C_{(i+1)}$, respectively. This can be done by appropriately selecting the D inputs of the neighboring cells $C_{(i-1)}$ and $C_{(i+1)}$. The shifter slices in the processor array can be made CI-testable if:

(1) During the first clock cycle, the test patterns for testing each shifter slice are generated by appropriately selecting the D inputs of all the cells in the array. By setting $(I_0, I_1, I_2, I_3, I_4) = (1, 1, 0, 1, 1)$, the F_i input of each shifter slice can be made equal to D_i, while the RI_i and LI_i inputs of each shifter slice can be made equal to $D_{(i-1)}$ and $D_{(i+1)}$, respectively.

As far as making the shifter slices C-testable is concerned, the following is true:

(a) Test pattern $(I_5, I_6, \text{RI, F, LI}) = (\text{d, d, 0, 0, 0})$ can be applied to every shifter slice of the processor array at the same time. The same is true for test pattern $(I_5, I_6, \text{RI, F, LI}) = (\text{d, d, 1, 1, 1})$.

(b) The other combinations of RI, F and LI cannot be applied to every shifter slice at the same time. Since the RI and LI inputs for each shifter slice have to be generated by its neighboring cells, the minimum length of the cycle which is the minimum number of shifter slices that test patterns can repeat is three. Using

this minimum length cycle, the other combinations of RI, F and LI can be applied to every shifter slice in six tests as shown in Figure 2.5(a). Therefore, the shifter slices can be made C-testable and can be exhaustively tested in thirty-two tests.

(c) For reasons that will become obvious later in the chapter, the minimum length cycle is not used. Rather, a cycle of length four is used, i.e., test patterns applied to shifter slices can repeat themselves every four shifter slices, as shown in Figure 2.5(b). In this case, the shifter slices are still C-testable and can be exhaustively tested in thirty-two tests.

During the clock cycle that a test vector is applied, the outputs of the ALU slices should not be propagated to the primary outputs of the processor array to be monitored because they are not identical. Therefore, during this clock cycle, the outputs of the A-latches, which can be made identical for all the cells in the array, should be propagated to the primary outputs of the array to be monitored. The test responses of the shifter slices are stored in the RAM slices.

(2) During the next clock cycle, the test responses of the shifter slices are verified. They are first read out of memory, then each of them is exclusive-ored with a D-input. The D-input patterns are arranged in such a way so as to make the outputs of all the ALU slices in the processor array identical. The outputs of the ALU slices are then propagated to the primary outputs of the array where they can be monitored by equality checkers.

Therefore, two test patterns of $T^{*}(SH)$ are actually needed to realize one test pattern of T(SH). So the shifter slices in the processor array can be made CI-testable and can be exhaustively tested in sixty-four tests.

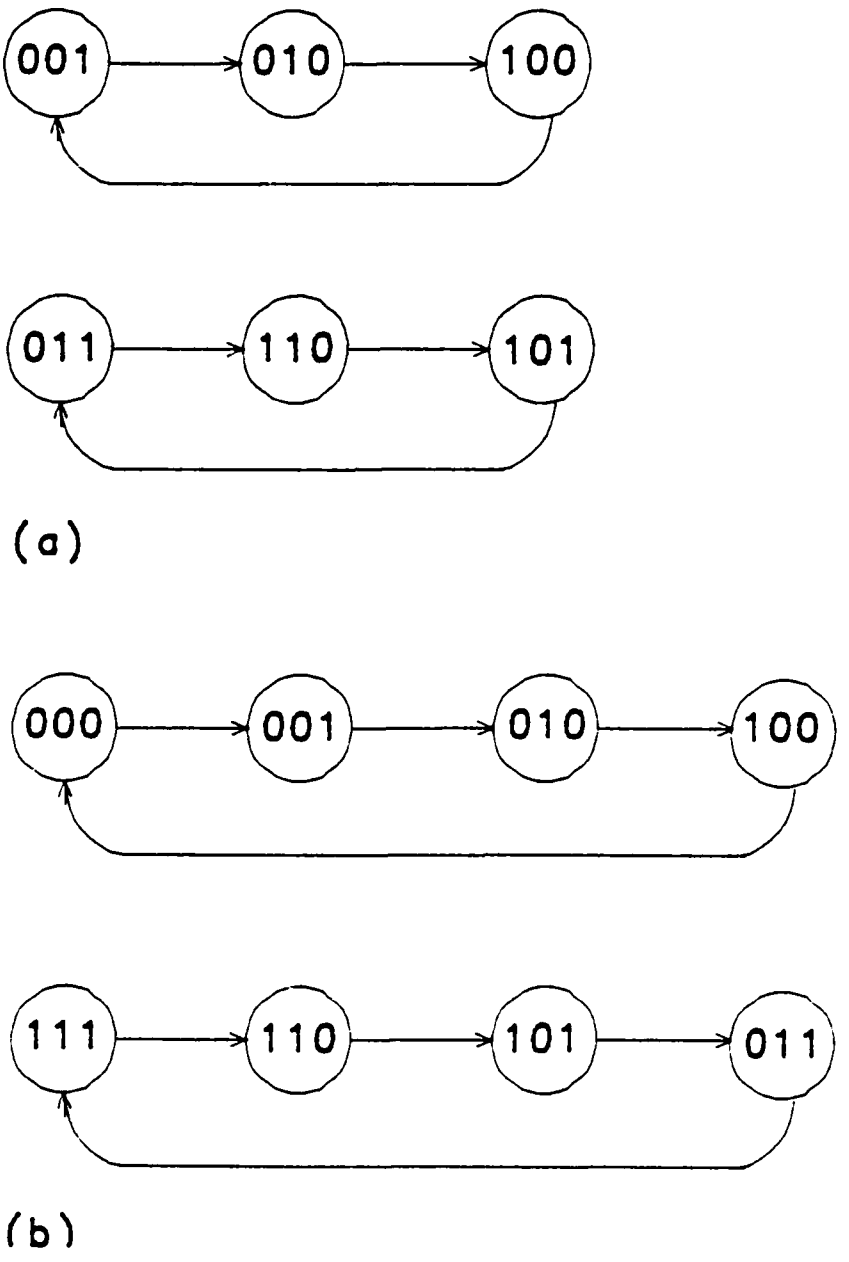


Figure 2.5 Test Pattern Cycles for the Shifter Array

2.5.2. The RAM

Since the RAM slices in each processor cell do not interact with the RAM slices in other processor cells, the test patterns developed for the RAM slice in a single cell are also applicable to any RAM slices in a processor array without any modifications. Therefore, the RAM slices are CI-testable and can be completely tested in thirty-two tests.

2.5.3. The ALU Source Multiplexer

Since the ALU source multiplexer slice in each processor cell does not interact with the ALU source multiplexer slices in other processor cells, the test patterns developed for the ALU source multiplexer slice in a single cell are also applicable to any ALU source multiplexer slices in a processor array without any modifications. Therefore, the ALU source multiplexer slices are CI-testable and can be exhaustively tested in thirty-two tests.

2.5.4. The ALU

The ALU performs eight functions; five are logical functions, and the remaining three are arithmetic functions. For logical operations, the ALU slices do not interact with each other, i.e., no data is passed between the ALU slices although they perform the same function. For arithmetic operations, each ALU slice in the processor array interacts with the ALU slices in the neighboring cells through its carry-out or borrow-out output and its carry-in or borrow-in inputs. To apply the test patterns obtained in the previous section for the ALU of a single processor cell to any ALU slices in a processor array, the desired CI input signal must be generated at the CO output of the neighboring cell. For the ALU slices to be CI-testable, the following conditions must be satisfied:

(1) The ALU slices in the processor array must be exhaustively tested by a number of tests, the length of which is independent of the size of the array. It can be readily seen that this does not present any problems as far as the logical operations of the ALU are concerned. The arithmetic operations of the ALU can also be tested independently of the array size if the R and S inputs of the ALU slices can be controlled. As shown in Figure 2.6, the ALU slices can be made C-testable if the R and S inputs of neighboring ALU slices can be set to different values for some tests and the same values for other tests.

(2) The test outputs of the ALU slices to be verified must be identical. From Figure 2.6, it can be seen that the outputs of the ALU slices in the processor array are not identical for some tests on arithmetic operations. Therefore, these outputs should not be propagated to the primary outputs of the array to be verified by the equality checkers.

The above conditions can be satisfied as follows:

(1) By controlling the D-inputs, appropriate logic values can be stored in the RAM slices of the processor array.

(2) By setting the ALU source multiplexer control signals (I_0, I_1) to (1, 0), and by controlling the B address field of the RAM and the D-input, the R and S inputs of the ALU slices of neighboring cells can be made the same or different.

(3) The test results are not verified during the first clock cycle but are stored in the RAM instead. By controlling the A address field of the RAM, and by propagating the outputs of the A-latches to the primary outputs to be monitored, the inputs to the equality checkers can be made identical.

(4) During the next clock cycle, the test responses of the ALU slices are read out of memory, then they are exclusive-ored with D-inputs to make the outputs of the ALU slices identical, which are then propagated to the primary outputs to be verified.

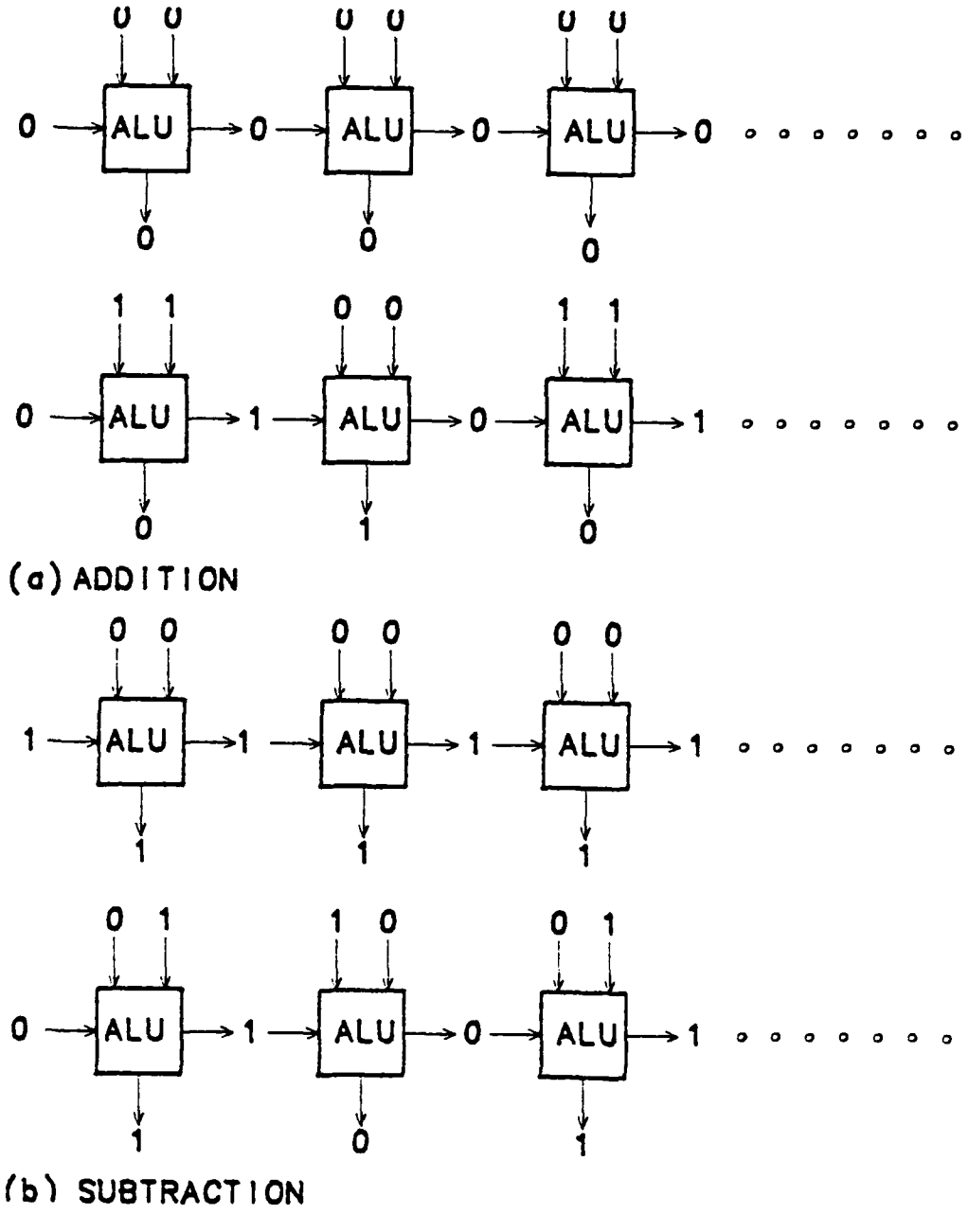


Figure 2.6 Some Tests for the ALU

Steps (3) and (4) are not necessary if the test results are identical for all ALU slices. In this case, the test results are propagated to the primary outputs to be monitored during the first clock cycle.

Therefore, some members of T(ALU) can be realized by one test pattern of $T^*(ALU)$, while other members of T(ALU) require two test patterns of $T^*(ALU)$ to be realized. Overall, seventy-two test vectors are required to test the ALU.

2.5.5. The Output Multiplexer

Since the output multiplexer slice in each processor cell does not interact with the output multiplexer slices in other processor cells, the test patterns developed for the output multiplexer slice in a single cell are also applicable to any output multiplexer slices in a processor array without any modifications. Therefore, the output multiplexer slices are CI-testable and can be exhaustively tested in eight tests.

2.5.6. Common Control Lines

From Figure 2.7, it can be seen that if one or more of the common control signals of the processor array stuck at 1 or 0 at the positions indicated, then this fault may not be detected by the test patterns generated so far because the effect of the fault is the same for every cell of the array. In other words, if the effect of the fault is to change an intended test to an unintended valid I-test, then the equality checker will fail to indicate an error. This would further result in causing some other faults developed later in a processor chip to be non-detectable because some of the modules may not be tested exhaustively due to the faults in the common control lines. Therefore, twenty-eight more test patterns are required to test each control line and address line for stuck at 1 or 0. Under the single fault assump-

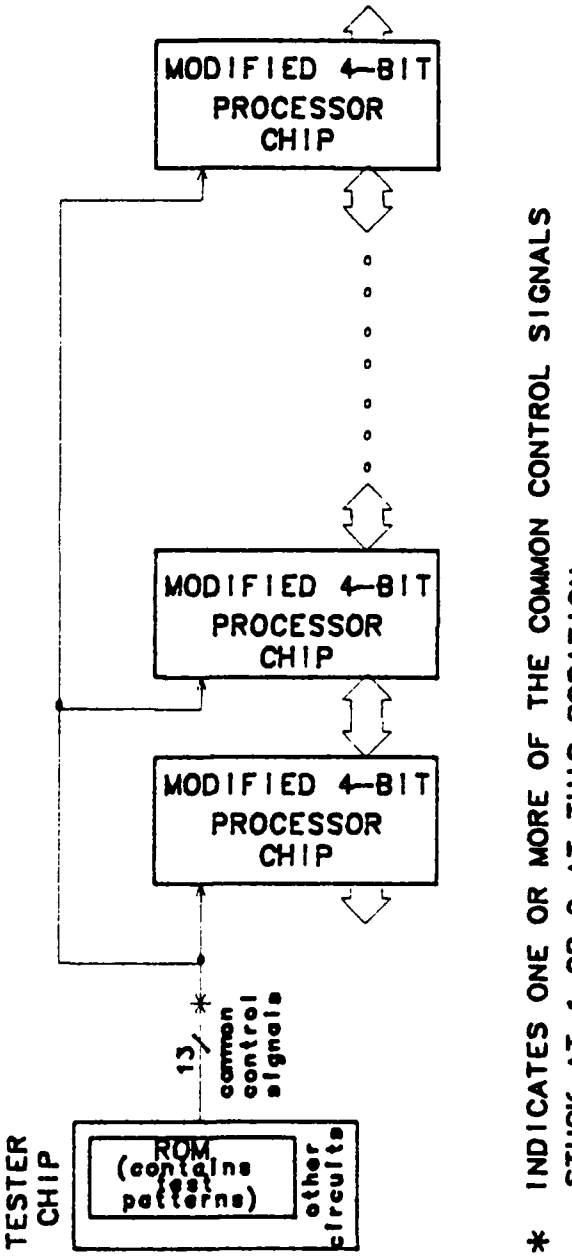


Figure 2.7 Showing Problems of Common Control Signals in I-Tests

tion, these twenty-eight test patterns, specially designed, can cause the inputs to the equality checker to be non-identical if one of these faults exists.

2.6. Implementation

From the previous sections, it should be obvious that the only modification required on each bit-sliced processor chip in order to make an array of these to be self-testable is the addition of equality checkers and multiplexers. There are five equality checkers in each four-bit processor chip. Four of them are for monitoring the Y outputs of the four bit slices, while the fifth one is for monitoring the CI input and CO output of the chip. The equality checkers themselves are bit-sliced too and are therefore cascadable. An ILA implementation of the equality checker is shown in Figure 2.8. Since the equality checkers are cascaded together as a one-dimensional ILA, there is only one output from the equality checker ILA that has to be monitored by the tester chip. There are four two-to-one multiplexers for choosing between the normal D-inputs and the test D-inputs. The control signal for these multiplexers is the TEST input. The modified four-bit processor chip is shown in Figure 2.9. A comparison of this figure with the unmodified four-bit processor chip shown in Figure 2.1 would show that there is very little modification on each processor chip and that the area overhead required for self-testing is basically the extra tester chip.

The tester chip contains a ROM which stores the test patterns for testing a bit-sliced processor array of any size. We now determine the width of the ROM in bits. As previously discussed, the ROM contains all the test control signals of the processor array. This includes I_0 to I_8 which are control signals of the ALU source multiplexer, the ALU, the output multiplexer, the shifter, and the enable signal of the RAM. The RAM also contains A_1, A_2, B_1, B_2, which are the A address field and B address field of the RAM. All the signals described so far are common to all bit slices.

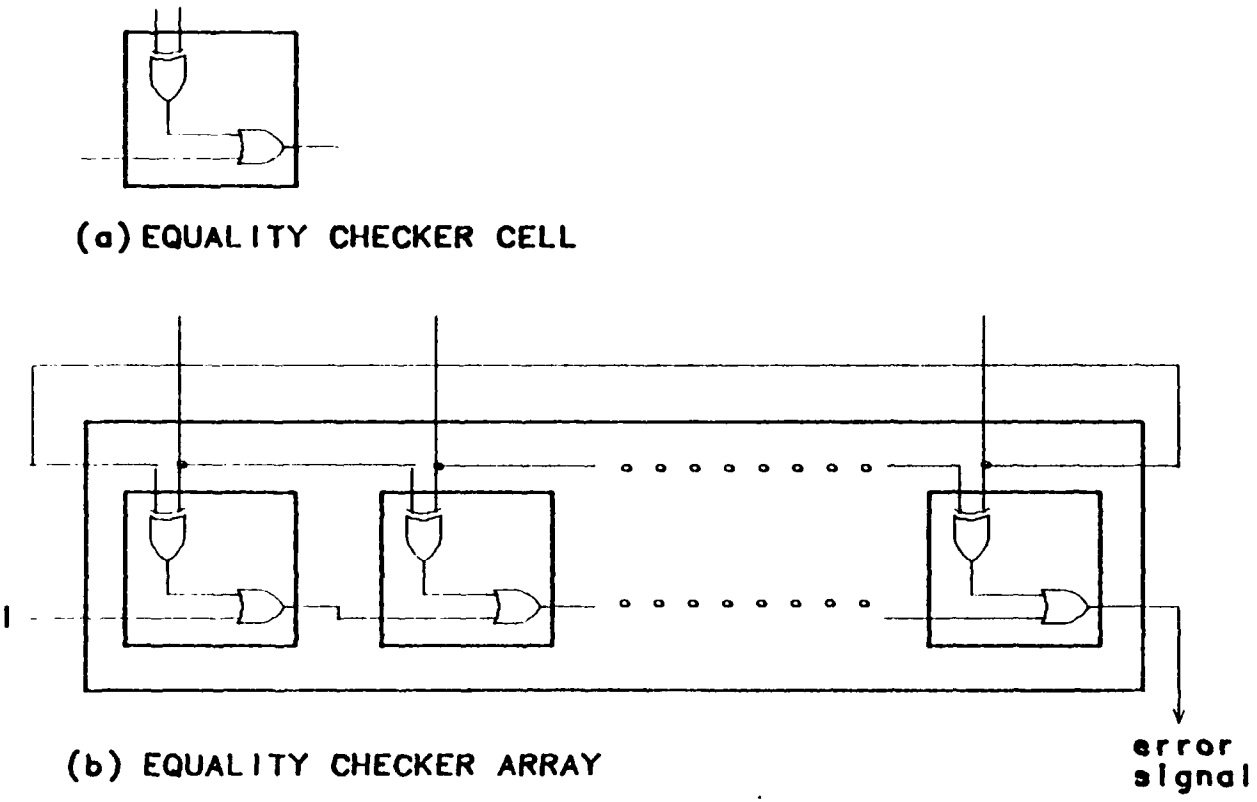


Figure 2.8 ILA Implementation of the Equality Checker

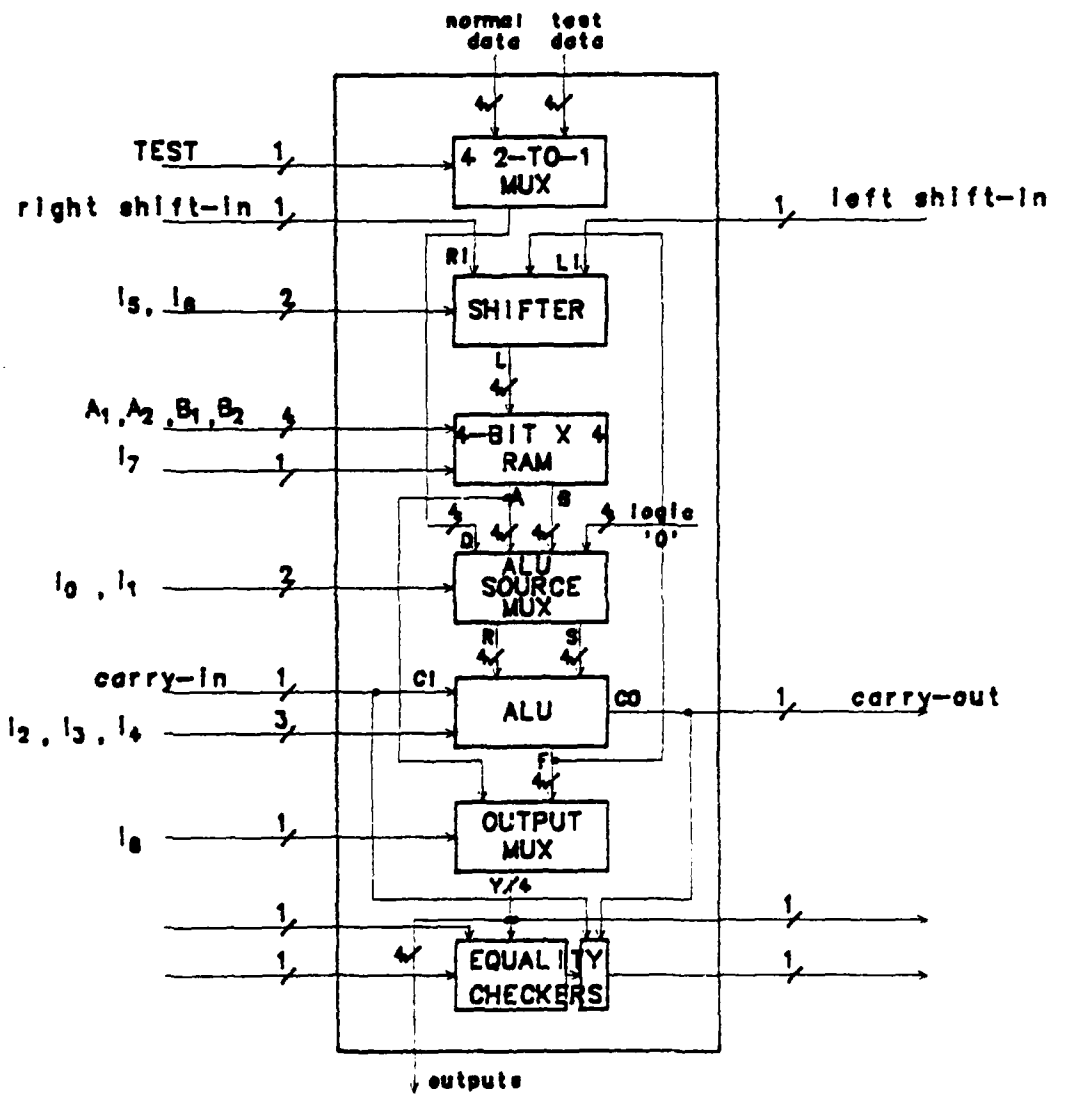


Figure 2.9 Block Diagram of the Modified Four-Bit Processor Chip

From the discussion on making the shifter slices CI-testable, and as shown in Figure 2.5, we know that four different combinations of test D-inputs are necessary to make the shifter slices CI-testable. Therefore, the ROM should also contain four different patterns of test D-inputs, namely, D_1, D_2, D_3 and D_4.

During normal operation, the LI and RI inputs of each shifter slice come from neighboring cells except the RI input of the first slice and the LI input of the last slice which come from some source external to the processor array. During testing, as we mentioned before, the connections between processor slices do not change, but now the RI input of the first slice and the LI input of the last slice have to come from the ROM of the tester chip instead of from some external source. Therefore, in addition to the test patterns of I_0 to I_8, A_1, A_2, B_1, B_2, D_1 to D_3, the ROM also contains the test patterns of the RI input of the first slice and the LI input of the last slice. There should be one test pattern of the RI input of the first slice, and one test pattern of the LI input of the last slice. If the minimum length cycle was used to apply test patterns to the shifter slices, then three different test patterns of the LI input of the last slice would be needed because of the consideration of variable processor array sizes. The major function of the test patterns of the RI input of the first slice and the LI input of the last slice is to test the shifter. During the testing of the other modules of the processor array, these test patterns are immaterial. Therefore, the test pattern of the RI input of the first slice can be the same as the test pattern D_4, while the test pattern of the LI input of the last slice can be the same as the test pattern D_1. So, no extra test patterns of the RI input of the first slice and the LI input of the last slice are needed.

From the previous discussion on making the ALU slices CI-testable, and as shown in Figure 2.6, we know that two different combinations of test D-inputs are necessary to make the ALU slices CI-testable. Since four is a multiple of two, the four different combi-

nations of test D-inputs that are required to make the shifter slices CI-testable can also be used to make the ALU slices CI-testable. But if the minimum cycle of length three was used to apply test patterns to the shifter slices, six different combinations of test D-inputs would be necessary to make both the shifter and the ALU slices CI-testable. The way in which these different patterns of test D-inputs are connected to the D-input pins, the RI input of the first slice and the LI input of the last slice of the array of four-bit processor chips is shown in Figure 2.10.

The carry-in (CI) input of the first slice comes from an external source during normal operation; it should come from the ROM of the tester chip during testing.

Finally, the I-input of the first equality checker slice should come from the ROM of the tester chip instead of being tied to logic one in order to make the equality checker array more testable.

Therefore, the width of the ROM is nineteen bits; these are: I_0 to I_8, A_1, A_2, B_1, B_2, D_1, D_2, D_3, D_4, CI, I.

We now determine the number of test patterns that have to be stored in the ROM. From the previous discussions, we know that all the modules in the processor array and the common control and address lines can be exhaustively tested by 32 + 64 + 72 + 32 + 8 + 28 = 236 test patterns.

Two more test patterns are required to test the equality checker, while three more test patterns are added as part of the scheme to make the tester chip able to test itself as will be discussed in the next section. Therefore, the total number of test patterns is 241. The size of the ROM in the tester chip is 19 bits by 241 words. The complete test set is listed in the Appendix.

The nineteen bits output field of the tester chip features tri-state outputs. During test mode (when the TEST input is high), these outputs of the tester chip are the same as the

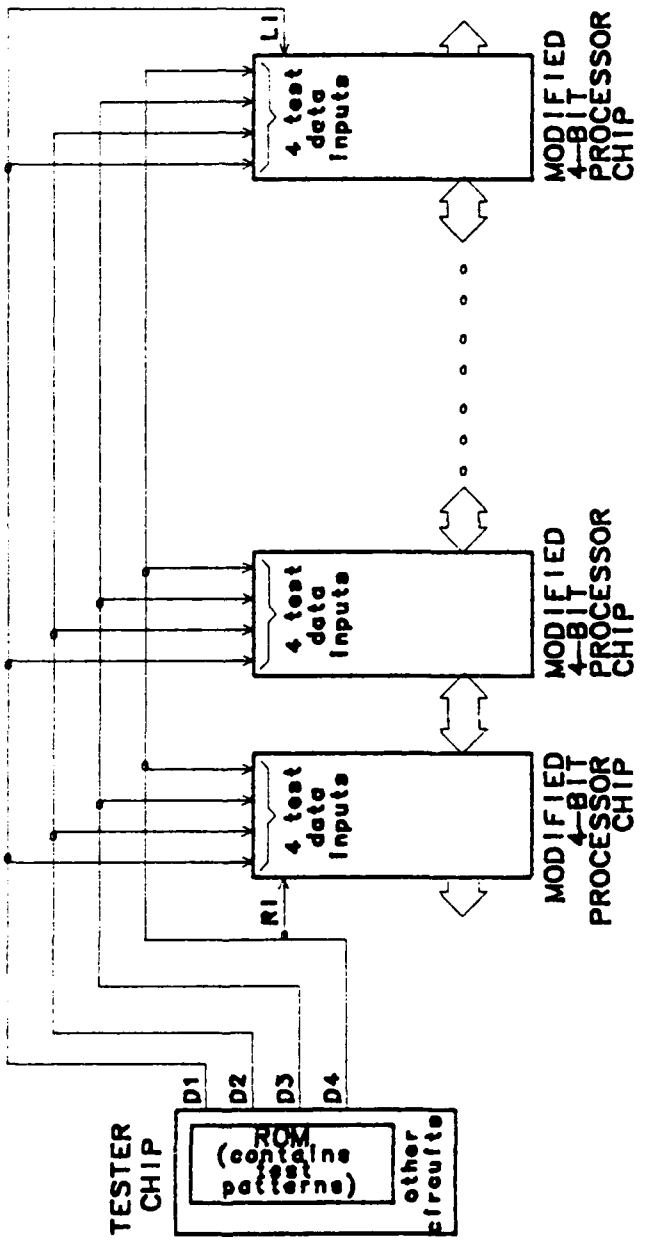


Figure 2.10 Connections of the Four Different Patterns of Test D-Input

outputs of the ROM. During normal mode (when the TEST input is low), these outputs are in the high-impedance state. By the same token, the outputs of the microprogram sequencer should also be tri-state. During test mode, the outputs of the tester chip should become the inputs of the processor array. During normal mode, the outputs of the microprogram sequencer should become the inputs of the processor array.

2.7. Self-Testable Tester Chip

The tester chip has been determined in the previous section to contain a ROM of 19 bits by 241 words. The other modules in the tester chip other than the ROM are an eight-bit counter and a decoder for the counter. The function of the eight-bit counter is to generate the address of the ROM which is then decoded by the decoder to select a certain word of the ROM. The counter is just a regular count-up counter. During normal operation, the content of the counter remains at zero. Once test mode is initiated, the counter will start counting upwards from zero until it reaches the address of the last test pattern stored in the ROM, then the counter will stop counting and its content remains to be the address of the last test pattern in the ROM. After the test responses have been verified, the TEST input can be brought low (normal mode), at which time the content of the counter would go back to zero.

A nineteen-bit parallel signature analyzer with the outputs of the ROM as its nineteen parallel inputs can be used to test the counter, the decoder and the ROM. Parity code or hamming code can only detect errors in the ROM, they cannot detect faults in the counter or the decoder, i.e., whether the test patterns are read out in the proper order or not cannot be checked by parity or hamming codes. But a nineteen-bit parallel signature analyzer can determine if the test patterns are read out in the proper order and the correctness of the contents of the ROM; therefore, a signature analyzer is used. The feedback

polynomial of the signature analyzer used is $X^{18} + X^{17} + X^{16} + X^{15} + X^{10} + X^{9} + X^{8} + X^{7} + 1$ which is a primitive polynomial whose roots are linearly independent [Pet61].

During normal operation, the signature analyzer contains all 0's. Once test mode is initiated, the signature analyzer will start shifting in the outputs of the ROM. This data compression process will continue until the TEST input is brought low (normal mode) again after the signature at the end of the test has been verified, at which time the contents of the signature analyzer will go back to all 0's. The signature can be verified by an AND gate with the address following that of the last test pattern used in producing the signature as the enable input.

The signature analyzer can detect most of the faults in the counter, the decoder, the ROM and even itself except for the following types of fault:

(1) One or more of the outputs of the signature analyzer stuck at the logic value of the correct final signature.

(2) There are faults in the counter, the decoder, the ROM and/or the signature analyzer whose effects are equivalent to the fault described in (1) above.

It is important to detect this type of fault because it is possible that some other faults developed later in one of the modules which should result in a bad signature, but will not because of the existence of this type of fault.

In order to detect this type of fault, a testing scheme which involves the monitoring of three separate signatures is used. The three signatures monitored (S1, S2 and S3 where S3 is the final signature as previously discussed) must satisfy the following condition:

For $0 \leqslant i \leqslant 18$, $\quad S1_i \text{ or } S2_i = \overline{S3_i}$

where the signature with the subscript i denotes the logic value of the bit at bit position i of the signature.

The monitoring of three separate signatures that satisfy the above condition not only can detect the type of fault listed above, but it can also increase the fault coverage in general. As is known for all test response verification methods that involve data compression, loss of fault coverage due to the compression of test responses is inherent. By monitoring three separate signatures evenly spread throughout the test set, this loss in fault coverage can be reduced.

Some programs in C-language have been written to determine the final signature at the end of the test and the other two signatures to be monitored that satisfy the above condition. They are found to be octal 204062, 575703 and 522055 which are the contents of the signature analyzer right after test patterns 33, 191 and 239, respectively, have been verified. It can be seen that they are evenly spread throughout the test set.

The circuit for monitoring the three signatures and the output of the equality checker is shown in Figure 2.11. As previously discussed, the AND gate that is used to verify the signature is enabled by the address following that of the test pattern used in producing the signature. This scheme is used so that the output of the AND gate would not become logic one unless the correct signature arrives at the right time. The outputs of the three-bit count-up counter are connected to the output pins of the tester chip and function as an indicator of the success of the test. The state diagram of the counter is shown in Figure 2.12. During normal mode, the output of the counter is 001. At the end of the test, a fault-free bit-sliced processor array and tester chip is indicated by a 110 at the output of the counter, while the presence of one or more faults in the bit-sliced processor array and/or tester chip is indicated by any outputs of the counter other than 110. The counter will stop counting once it reaches 111 regardless of the input. Each correct signature causes the counter to advance one, so three correct signatures cause the counter to count up to 100; two more logic one outputs from the equality checker will advance the counter to 110.

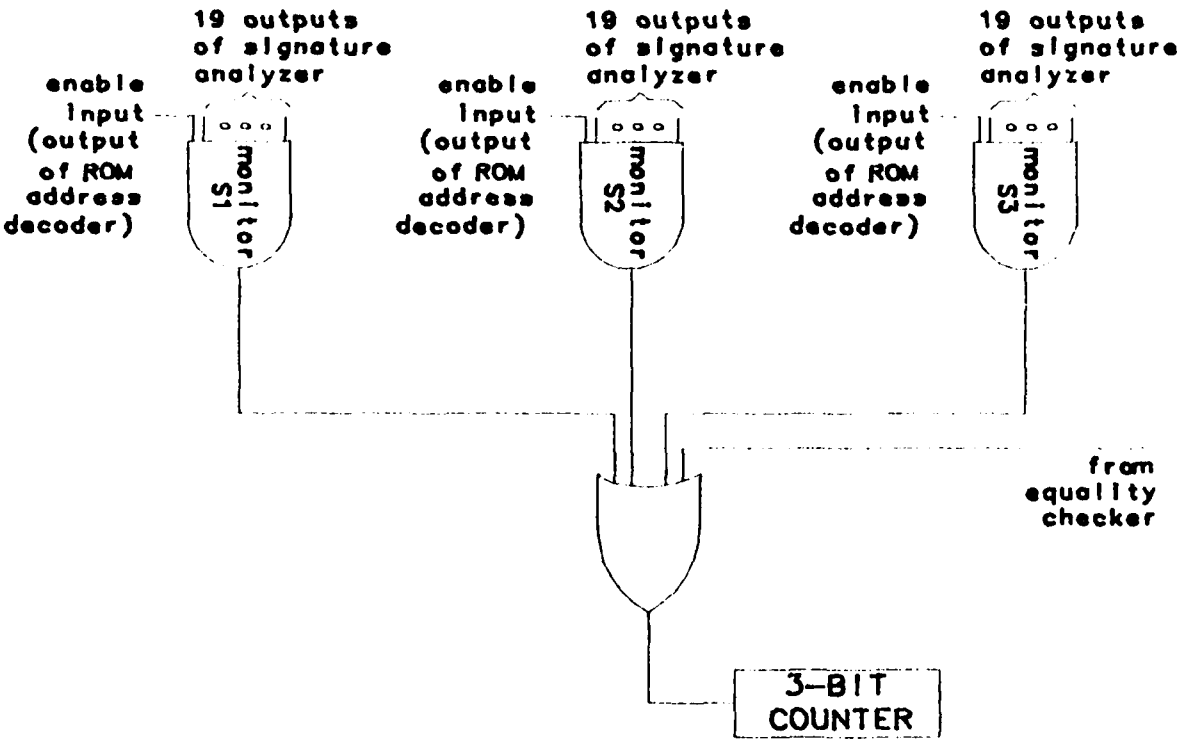
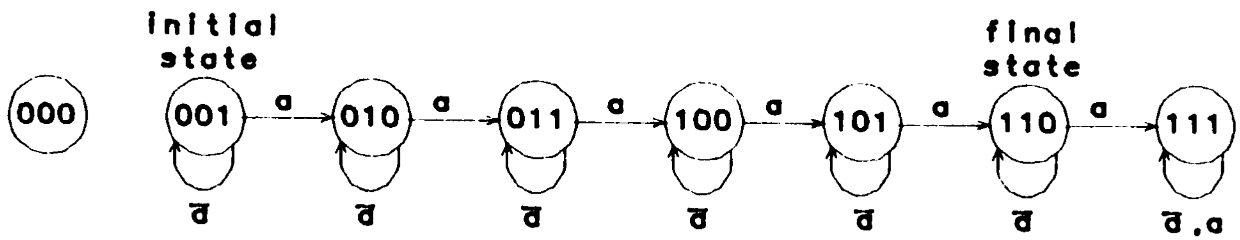


Figure 2.11 Circuit for Monitoring Signatures and Equality Checker Output



COUNTER ADVANCES AT SIGNAL a

a=1 IF A SIGNATURE MATCHES OR
EQUALITY CHECKER OUTPUT=1

a=0 OTHERWISE

Figure 2.12 State Diagram of the Three-Bit Counter

The counter output during normal mode is chosen to be 001 while the output at the end of a successful test is chosen to be 110 so that the output of the counter and/or the output pins of the tester chip can be known not to stuck at any logic values.

The tester chip is equipped with another output pin which is normally low and will only go high when the end of the test is reached. The return to normal mode will cause this output to go low again. Therefore, during test mode, the three-bit counter output is not checked to determine the success of the test until this test-end indicator pin becomes logic one.

2.8. Performance

We now discuss the performance degradation due to the incorporation of extra circuits to provide the self-testing ability. As was previously discussed, the tester chip is isolated from and transparent to the processor array during normal operation. The control signals and data signals arrive at the inputs of the processor array as if the tester chip does not exist because the outputs of the tester chip are in high-impedance state during normal mode. The only modification within each processor chip is the addition of equality checkers to monitor the Y outputs and multiplexers to choose between normal and test D-inputs. The equality checkers are not in the path of any signals; therefore, they do not induce any performance degradation. The only performance degradation comes from the multiplexers because normal D-inputs have to go through them before arriving at the inputs of the ALU source muitiplexer. Hence, there is very little performance degradation which is one of the advantages of this approach.

2.9. Fault Coverage

2.9.1. The Processor Array

As was previously discussed, the processor array is CI-testable. Each combinational module of each cell in the array is tested exhaustively based on the functional fault model. The RAM slice of each cell is tested completely based on the memory functional fault model [Nai78]. Therefore, any faults confined to a single module (shifter, RAM, ALU source multiplexer, ALU, output multiplexer) can be detected by our test set. The added equality checkers are partially tested because they are not CI-testable; an exhaustive test would require a special tester chip for each array size. The added multiplexers are also partially tested because their operation in normal mode cannot be tested. Therefore, the fault coverage for the processor array is almost 100 percent. Since the test responses are monitored by equality checkers, there is no loss in fault coverage resulting from the verification of test responses.

2.9.2. The Tester Chip

The signature analyzer can detect most of the functional faults in the ROM address counter, the ROM address decoder, the ROM and itself. The loss in fault coverage due to test results compression is minimized by the monitoring of three signatures instead of one.

The circuit for monitoring signatures and equality checker output is designed in such a way, as previously discussed, that any line stuck at 1 or 0 would almost certainly be detected.

Therefore, the fault coverage of the tester chip should be very high although no fault simulation has been done.

CHAPTER 3

PHYSICAL DESIGN OF THE TESTER CHIP

3.1. Global Description

The logic circuitry and physical layout of the tester chip are presented in this chapter. The physical layout is implemented in a single metal layer NMOS process with lambda equal to two microns [Mea80].

The area of the chip, not including input/output pads, is about 9500 * 1900 square microns. The power and ground busses are arranged as interlocking forks to avoid the crossover of power lines.

The tester chip is made up of five modules. They are the ROM address counter, the ROM address decoder, the ROM, the signature analyzer and the circuit for monitoring signatures and equality checker output. The block diagram of the tester chip is depicted in Figure 3.1.

The tester chip has thirty-one input/output pads:

VDD, GND – Power and ground, respectively.

CLK1, CLK2 – Phase-one and phase-two clocks, respectively.

TEST – Test mode is initiated by applying a logic one to this input and terminated by applying a logic zero.

R_0 to R_{18} – Tri-state pads of the nineteen outputs of the ROM. They are in high-impedance state if the TEST input is low (normal mode).

EQCK – Input from the equality checker of the processor array.

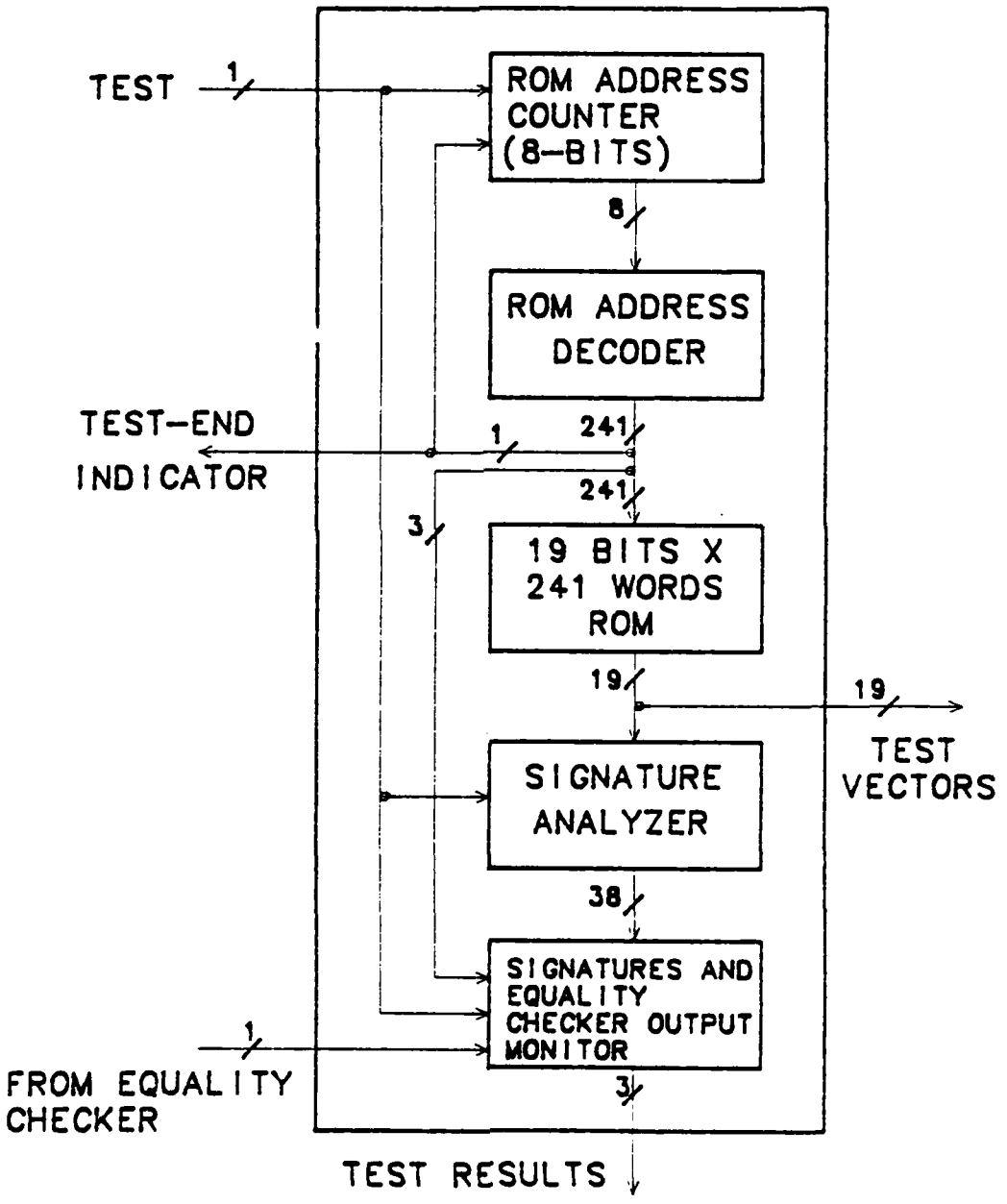


Figure 3.1 Block Diagram of the Tester Chip

TEST END – A logic one at this output pin indicates that the end of the test has been reached. In normal mode, this pin is at logic zero.

Q_0, Q_1, Q_2 – The three outputs of the three-bit counter indicating the success of the test.

The floor plan of the tester chip is depicted in Figure 3.2.

3.2. Modular Description

3.2.1. The ROM Address Counter

The outputs of the eight-bit counter are all 0's when the TEST input is low. The counter will start counting upwards as soon as the TEST input becomes high. The counter outputs do not change in phase-one, they will only change in phase-two. The counter will stop counting as soon as the TEST END signal from the ROM address decoder becomes logic one. A logic zero applied to the TEST input will cause the outputs of the counter to return to all 0's.

The eight-bit ROM address counter is formed by cascading eight one-bit counters together in a one-dimensional ILA. Each one-bit counter cell has a count enable input E. This input E is generated by the one-bit counter cell of the previous stage except for the first stage. The input E_0 for the first stage is given by:

$$E_0 = \text{TEST} \,.\, \overline{\text{TEST END}}$$

The count enable signal for any stage of the counter will not be valid until the count enable signals for all its previous stages are valid. Therefore, the count enable signal propagates like the carry signal of the ALU of the processor array we discussed earlier.

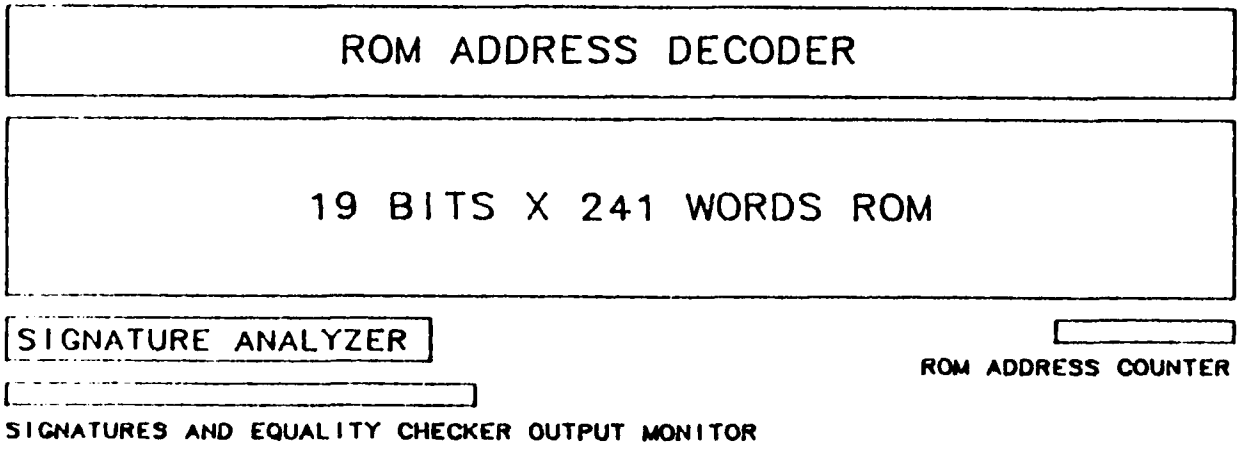


Figure 3.2 Floor Plan of the Tester Chip

A timing simulation has been done on the eight-bit ROM address counter. The minimum phase-one duration plus phase-two duration is found to be 70 nanoseconds, while the minimum phase-one duration is 10 nanoseconds and the minimum phase-two duration is 20 nanoseconds. The reason for the discrepancy is that some of the operations can be done in either phase-one or phase-two. The decision on how long a phase-one and a phase-two duration should be used is dependent upon the operations and delays of the other modules of the tester chip.

3.2.2. The ROM Address Decoder

The block diagram of the ROM address decoder is shown in Figure 3.3. The inputs of the decoder are the outputs and the complements of the outputs of the ROM address counter. These input lines run vertically through all the cells of the ROM address decoder. The ROM address decoder consists of 241 decoder cells. Each decoder cell is basically a NOR gate because it takes less area and is much faster than a corresponding AND gate (much larger pull-up for an eight-input AND gate). The outputs of the decoder are the word select lines of the ROM. Only one of the output lines can be at logic one at any time for a valid input.

Since the inputs of the decoder are long metal and polysilicon (polysilicon at the inputs of NOR gates, metal otherwise) lines that run vertically through all 241 decoder cells, there is a long delay before these lines are charged up or discharged to the proper voltages by the outputs of the address counter. This means that the outputs of the ROM address decoder would not be valid until after a long delay. Therefore, the following steps have been taken to minimize this delay:

(1) The outputs of the ROM address counter drive larger transistors at the inputs of the ROM address decoder; these larger transistors can in turn drive the long metal and

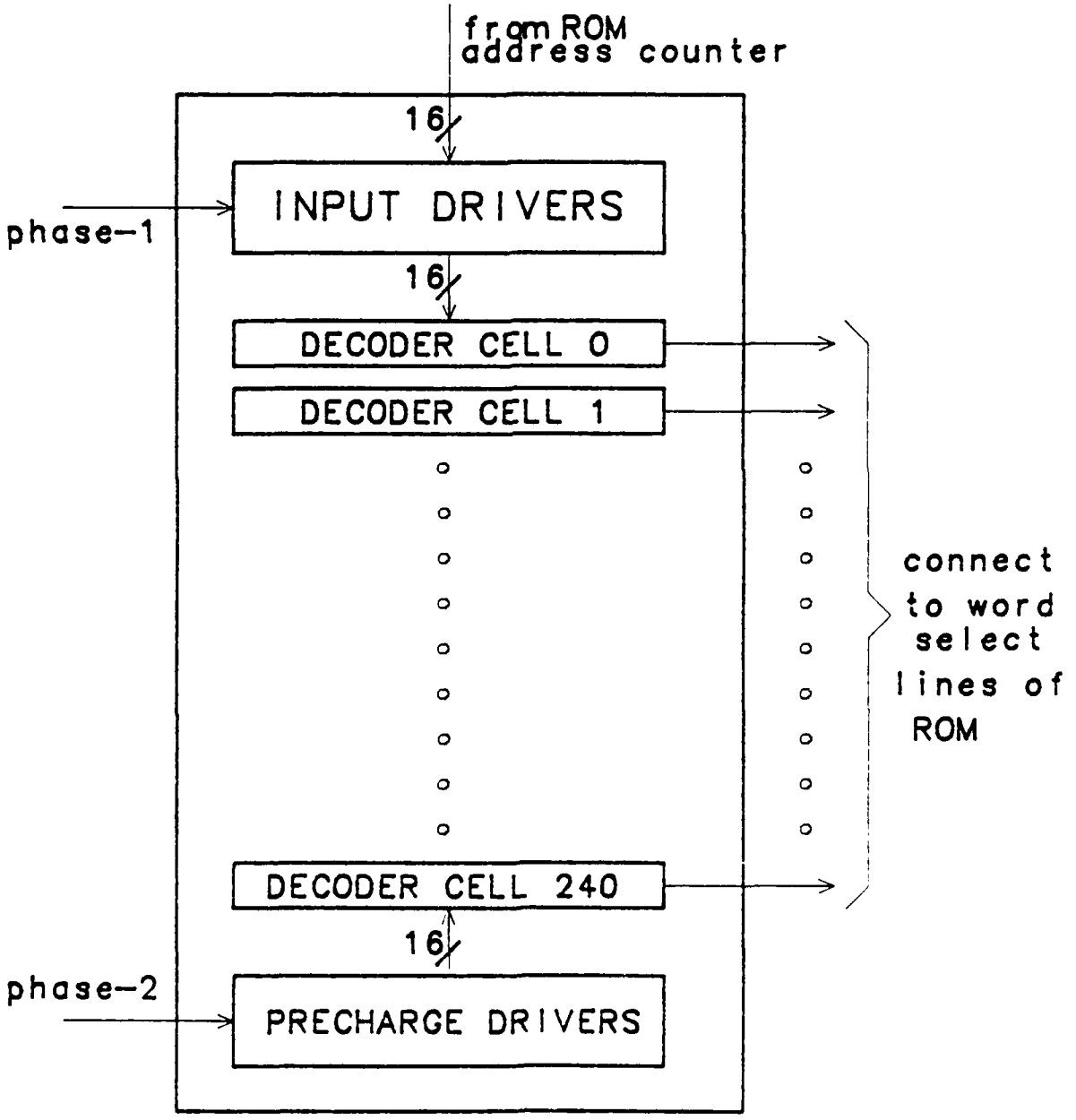


Figure 3.3 Block Diagram of the ROM Address Decoder

polysilicon lines faster.

(2) All the input lines of the address decoder are precharged to logic one during phase-two (even for input lines that are normally complements of each other), so that all the outputs of the address decoder, which are word select lines of the ROM, would go to logic zero. Therefore, during phase-two, none of the words of the ROM are selected. During phase-one, half of the input lines of the ROM address decoder would be discharged, resulting in the selection of one word of the ROM. This precharge scheme provides speedup because pull-down is faster than pull-up. The other advantages of this precharge scheme are discussed in the next section.

Timing simulation has been done on the decoder and the ROM; the results are presented in the next section.

3.2.3. The ROM

The block diagram of the ROM is shown in Figure 3.4. The size of the ROM is 19 bits by 241 words. The inputs of the ROM are its word select lines which are the outputs of the ROM address decoder. Therefore, the ROM has 241 inputs. The ROM consists of 19 * 241 = 4579 memory elements. Each memory element is quite small and two neighboring elements share the same power bus and ground bus as shown in Figure 3.5. Each word select line runs horizontally through the nineteen memory cells that make up each word. The power bus, ground bus and bit lines all run vertically through all 241 words of the ROM. The nineteen bit lines are connected to output buffers.

Since the bit lines are long metal lines that run vertically through all 241 words, it takes a long time in order to charge them up to the proper voltage. In order to reduce this charge-up time, the bit lines are precharged during phase-two.

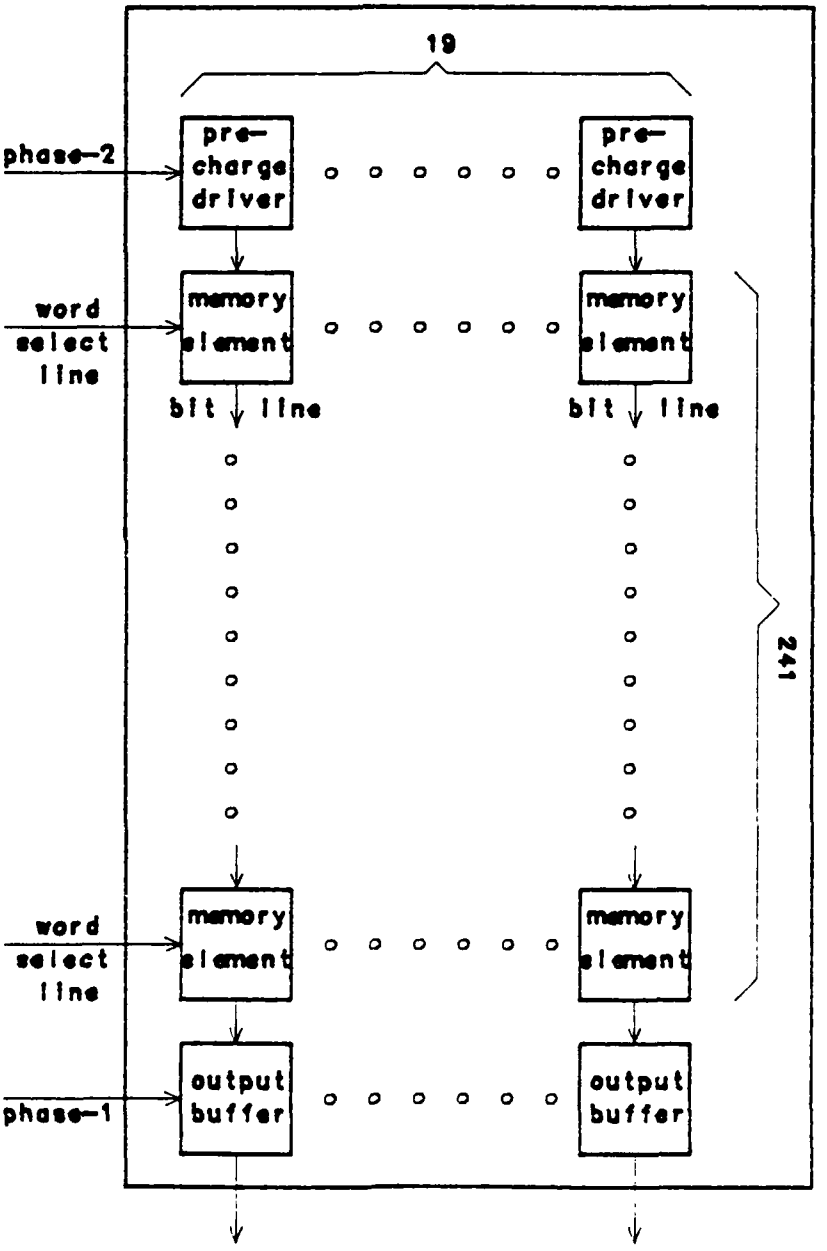


Figure 3.4 Block Diagram of the ROM

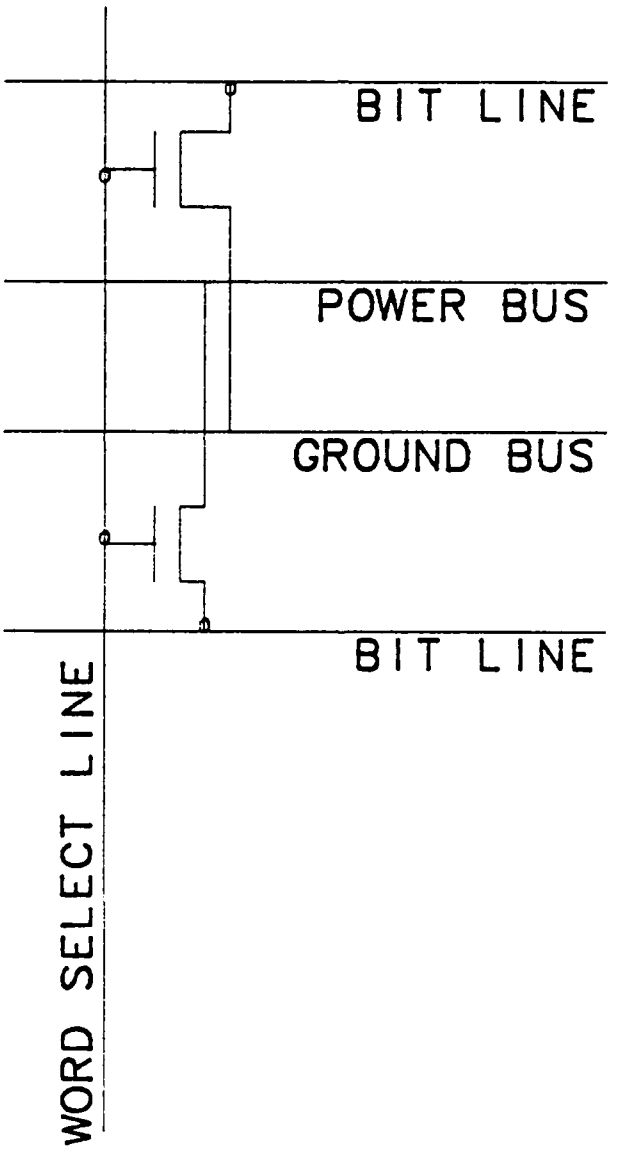


Figure 3.5 Memory Elements

During phase-two, the bit lines are precharged and isolated from the output buffers. This is required so that the outputs of these buffers do not change in phase-two because they have to drive the signature analyzer and the processor array. In order for all the bit lines to be properly precharged, none of the words should be selected during phase-two; therefore, as discussed in the previous section, all the input lines of the ROM address decoder are precharged during phase-two so that all its output lines, which are the word select lines of the ROM, are at logic zero.

During phase-one, a word is selected which results in some of the bit lines being discharged. The bit lines are connected to the output buffers in phase-one.

A timing simulation has been done on the decoder and the ROM. The minimum phase-two duration, which is the time required for the input lines of the decoder to be precharged and the bit lines of the ROM to be at least partially precharged after all the word select lines have been discharged, is found to be 55 nanoseconds. The minimum phase-one duration, which is the time required for the bit lines and the output buffers to be properly charged or discharged after one of the word select lines becomes logic one, is found to be 150 nanoseconds.

3.2.4. The Signature Analyzer

The signature analyzer has nineteen inputs which are outputs of the ROM buffers. Its thirty-eight outputs form the signature and its complement. During normal mode, the outputs of the signature analyzer are all 0's and the inputs have no effect on the outputs. During test mode, the inputs are compressed to form signatures. The outputs do not change in phase-two, they change only in phase-one. The inputs have to be stable during phase-two when they are monitored. From the previous section, the outputs of the ROM buffers are known to be stable during phase-two.

A timing simulation has been done on the signature analyzer. The minimum phase-one plus phase-two duration is found to be 180 nanoseconds, while the minimum phase-one duration is 5 nanoseconds and the minimum phase-two duration is 80 nanoseconds. The reason is again that some operations can be performed in either phase-one or phase-two. The decision on how long a phase-one and a phase-two duration should be used is again dependent upon the operations and delays of the other module of the tester chip.

3.2.5. The Circuit for Monitoring Signatures and Equality Checker Output

The circuit for monitoring the three signatures and the equality checker output is shown in Figure 2.11. The three signatures are monitored by three twenty inputs NOR gates. The reason for using NOR gates instead of AND gates here is the same as that for using NOR gates in the ROM address decoder. The nineteen inputs to each NOR gate are the regular outputs or their complements of the signature analyzer. The other input is an output of the ROM address decoder which functions as an enable input of the NOR gate, as explained in Chapter 2.

The three-bit counter is made up of three one-bit counters cascaded in a one-dimensional ILA similar to the ROM address counter. The count enable input for the first stage of the counter E_0 is given by:

E_0 = (output of OR gate) . $\overline{Q_0 \cdot Q_1 \cdot Q_2}$

where Q_0, Q_1 and Q_2 are the outputs of the three-bit counter.

Since the input lines of the ROM address decoder are precharged during phase-two, the outputs of the NOR gates for monitoring the signatures are not valid during phase-two. The outputs of the three-bit counter do not change in phase-one, they change only in phase-two.

A timing simulation has been done on this circuit. The minimum phase-one duration is found to be 95 nanoseconds while the minimum phase-two duration is 30 nanoseconds.

3.3. Timing

The overall minimum phase-one and phase-two durations of the tester chip are now determined.

During phase-two, the outputs of the ROM address counter change, but they are isolated from the ROM address decoder because the input lines of the ROM address decoder are being precharged during this phase. The bit lines of the ROM are also precharged and are isolated from the output buffers in phase-two. The outputs of the ROM buffers, being stable during phase-two, are shifted into the signature analyzer. Therefore, the minimum phase-two duration of the tester chip is just the largest of the minimum phase-two durations of all the modules, which is 80 nanoseconds.

During phase-one, the ROM address counter outputs are connected to the ROM address decoder. Half of the input lines of the ROM address decoder are discharged which results in the selection of a word of the ROM. The bit lines and output buffers are then charged or discharged to the proper voltages. The signature analyzer outputs and the equality checker output are verified during this phase. Therefore, the minimum phase-one duration of the tester chip is just the minimum phase-one duration of the decoder and the ROM, which is 150 nanoseconds.

3.4. Built-In-Test Area Overhead

Since the tester chip can test a processor array of any size and also itself, the percentage of area overhead decreases as the size of the processor array increases. The additional circuitry on each bit-sliced processor chip is just the equality checkers and the four two-to-

one multiplexers; therefore, the major area overhead is the tester chip. The tester chip has thirty-one input/output pins which is about the same as that of a four-bit processor chip. Therefore, for a sixteen-bit processor array, the area overhead is just over twenty percent; for a thirty-two bit processor array, the area overhead is just over eleven percent.

CHAPTER 4

CONCLUDING REMARKS

This thesis presents an approach to designing self-testing bit-sliced processor arrays. The conventional approach of making each bit-sliced processor chip self-testing is not used. A new approach of using an extra chip (tester chip) which can test a bit-sliced processor array of any size and itself is used. As the VLSI technology improves, it will be possible to integrate the tester in the same chip as the processor.

There is very little performance degradation due to the incorporation of extra circuits to provide the self-testing ability. The fault coverage of the bit-sliced processor array is almost 100 percent of the assumed faults with all the interconnections between chips tested. The fault model is a functional fault model and assumes that any single module of any one-bit slice can be faulty. The fault coverage of the tester chip itself is also very high. The area overhead is small especially for large processor arrays. The test length (time) is short so tests can be performed more frequently.

Although the approach developed is based on our processor slice which is similar to the commercially available AMD2901, it can be easily extended to any other types of processor slice.

The approach developed in this research can be extended to other types of one- or two-dimensional iterative logic arrays. Non C-testable and non I-testable ILA's can be made CI-testable by modifying the basic cell and/or designing special test sets. A possible application of this approach is the designing of a self-testing bit-sliced microprogram sequencer.

APPENDIX

TEST PATTERNS

Table A.1 Test Patterns for the RAM

I_0	I_1	I_2	I_3	I_4	I_5	I_6	A_0	A_1	B_0	B_1	I_7	I_8	D_1	D_2	D_3	D_4	CI	I
1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Table A.2 Test Patterns for the Shifter

I_0	I_1	I_2	I_3	I_4	I_5	I_6	A_0	A_1	B_0	B_1	I_7	I_8	D_1	D_2	D_3	D_4	Cl	I
1	1	0	1	1	0	0	0	1	0	0	1	1	0	1	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	0	0	1	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	1	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	1	1	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	1	1	0	1	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	1	1	1	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	0	1	1	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	0	0	1	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
1	1	0	1	1	0	0	0	1	0	0	1	1	1	0	0	0	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	1	1	0	1	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	0	1	1	0	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	0	1	1	1	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	1	1	1	1	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	0	1	1	1	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	0	1	0	0	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	0	1	0	0	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	1	0	1	0	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	0	1	1	0	0	0
1	0	1	1	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0
1	1	0	1	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0
1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	1	1	0	0	1	0	0	1	1	0	0	1	0	0	0
1	0	1	1	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0
1	1	0	1	1	1	0	0	1	0	0	1	1	0	1	0	0	0	0
1	0	1	1	0	1	0	0	1	0	0	0	0	1	1	0	0	0	0
1	1	0	1	1	1	0	0	1	0	0	1	1	1	1	0	1	0	0
1	0	1	1	0	1	0	0	1	0	0	0	0	1	0	1	1	0	0
1	1	0	1	1	1	0	0	1	0	0	1	1	1	1	1	0	0	0
1	0	1	1	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0
1	1	0	1	1	1	0	0	1	0	0	1	1	0	1	1	0	0	0
1	0	1	1	0	1	0	0	1	0	0	0	0	1	1	1	0	0	0
1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	1	1	0	0
1	0	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0
1	1	0	1	1	1	1	0	1	0	0	1	1	0	0	1	0	0	0
1	0	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0
1	1	0	1	1	1	1	0	1	0	0	1	1	1	0	0	0	0	0
1	0	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0
1	1	0	1	1	1	1	0	1	0	0	1	1	1	0	1	1	0	0
1	0	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0
1	1	0	1	1	1	1	0	1	0	0	1	1	0	1	1	1	0	0
1	0	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0

Table A.3 Test Patterns for the ALU Source Multiplexer

I_0	I_1	I_2	I_3	I_4	I_5	I_6	A_0	A_1	B_0	B_1	I_7	I_8	D_1	D_2	D_3	D_4	Cl	I
0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	1	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	0	0	0	0	1	1	1	1	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
1	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0
1	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
1	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0
1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0

Table A.4 Test Patterns for the ALU

I_0	I_1	I_2	I_3	I_4	I_5	I_6	A_0	A_1	B_0	B_1	I_7	I_8	D_1	D_2	D_3	D_4	CI	I
0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	0	1	1	0	0	1	0	1	1	1	1	1	0	1	0	0	0
1	0	0	0	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0
1	0	1	1	0	0	0	1	0	1	1	0	0	0	1	0	1	0	0
1	0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	1	1	0
1	0	1	1	0	0	0	1	0	1	1	0	0	1	0	1	0	1	0
0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	1	0	1	1	1	1	0	1	0	1	1	0
1	0	1	1	0	0	0	1	0	1	1	0	0	0	1	0	1	1	0
1	0	0	0	1	0	0	1	0	1	1	1	1	1	0	1	0	0	0
1	0	1	1	0	0	0	1	0	1	1	0	0	1	0	1	0	0	0
0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	1	0	1	1	1	1	0	1	0	1	0	0
1	0	1	1	0	0	0	1	0	1	1	0	0	1	0	1	0	0	0
1	1	0	1	1	0	0	1	0	1	1	1	1	0	1	0	1	0	0
1	0	0	1	0	0	0	1	0	1	1	1	1	0	1	0	1	0	0
1	0	1	1	0	0	0	1	0	1	1	0	0	1	0	1	0	1	0

Table A.5 Test Patterns for the Output Multiplexer

I_0	I_1	I_2	I_3	I_4	I_5	I_6	A_0	A_1	B_0	B_1	I_7	I_8	D_1	D_2	D_3	D_4	CI	I
1	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0
1	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	1	0	0	1	1	1	1	1	0	0
1	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	1	1	1	1	1	0	0

Table A.6 Test Patterns for the Common Control Signals

I_0	I_1	I_2	I_3	I_4	I_5	I_6	A_0	A_1	B_0	B_1	I_7	I_8	D_1	D_2	D_3	D_4	CI	I
1	1	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	0	1	0	0	0	0	1	0	1	0	0
1	0	1	1	0	0	0	1	1	1	1	0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1	1	1	1	0	0	1	1	1	1	0	0
1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1	0	1	0	0	1	1	0	1	0	0	0
1	0	1	1	0	0	0	1	1	1	1	0	0	0	1	0	1	0	0
1	1	0	1	1	1	0	1	0	0	1	1	1	0	0	1	0	0	0
1	0	1	1	0	1	0	1	0	0	1	0	0	0	1	0	0	0	0
1	1	0	1	1	0	1	1	0	0	1	1	1	0	1	0	0	0	0
1	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0
1	1	0	1	1	0	0	1	0	0	1	1	1	0	1	0	0	0	0
1	0	1	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0
1	1	0	1	1	0	0	1	0	0	1	1	1	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	0	1	0	1	0	1	0	1	0	0
1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	0	1	1	1	0	1	0	1	0	0
1	0	1	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0

REFERENCES

[Agr78] V. D. Agrawal, "When to Use Random Testing," *IEEE Transactions on Computers*, vol. C-27, pp. 1054-1055, November 1978.

[AMD76] Advanced Micro Devices, *Am2900 Bipolar Microprocessor Family*, Sunnyvale, CA, 1976.

[Ban82] P. Banerjee and J. A. Abraham, "Fault Characterization of VLSI MOS Circuits," *Proc. IEEE Int. Conf. on Circuits and Computers ICCC '82'*, pp. 564-568, September - October 1982.

[For65] R. E. Forbes *et al.*, "A Self-Diagnosable Computer," *Proc. FJCC*, pp. 1073-1086, 1965.

[Fri73] A. D. Friedman, "Easily Testable Iterative Systems," *IEEE Transactions on Computers*, vol. C-22, pp. 1061-1064, December 1973.

[Kau67] W. H. Kautz, "Testing for Faults in Cellular Logic Arrays," *Proc. 8th Symp. on Switching Automata Theory*, pp. 161-174, 1967.

[Kon79] B. Konemann *et al.*, "Built-In Logic Block Observation Techniques," *Dig. 1979 Test Conf.*, Cherry Hill, NJ, pp. 37-41, 1979.

[McC81] E. J. McCluskey and S. B. Nesbat, "Design for Autonomous Test," *IEEE Transactions on Computers, vol. C-30, pp. 866-875, November 1981.*

[Mea80] C. Mead and L. Conway, *Introduction to VLSI Systems*, Reading, MA: Addison-Wesley, 1980.

[Nai78] R. Nair, S. M. Thatte and J. A. Abraham, "Efficient Algorithms for Testing Semiconductor Random-Access Memories," *IEEE Transactions on Computers*, vol. C-27, pp. 572-576, June 1978.

[Par81] R. Parthasarathy and S. M. Reddy, "A Testable Design of Iterative Logic Arrays," *IEEE Transactions on Computers*, vol. C-30, pp. 833-841, November 1981.

[Pet61] W. W. Peterson, *Error-Correcting Codes*, Cambridge, MA: The M.I.T. Press, 1961.

[Smi80] J. E. Smith, "Measures of the Effectiveness of Fault Signature Analysis," *IEEE Transactions on Computers*, vol. C-29, pp. 510-514, June 1980.

[Sri81] T. Sridhar and J. P. Hayes, "A Functional Approach to Testing Bit-Sliced Microprocessors," *IEEE Transactions on Computers*, vol. C-30, pp. 563-571, August 1981.

[Sri81] T. Sridhar and J. P. Hayes, "Design of Easily Testable Bit-Sliced Systems," *IEEE Transactions on Computers*, vol. C-30, pp. 842-854, November 1981.

[Sri82] T. Sridhar, D. S. Ho, T. J. Powell and S. M. Thatte, "Analysis and Simulation of Parallel Signature Analyzers," *Proc. IEEE Int. Test Conf.*, November 1982.

[Tha77] S. M. Thatte, "Fault Diagnosis of Semiconductor Random-Access Memories," Coordinated Science Laboratory, University of Illinois, Urbana, Rep. R-769, May 1977.

[Wak78] J. Wakerly, *Error-Detecting Codes, Self-Checking Circuits and Applications*, New York: North-Holland, 1978.

[Wil82] T. W. Williams and K. P. Parker, "Design for Testability - A Survey," *IEEE Transactions on Computers*, vol. C-31, pp. 2-15, January 1982.

END

FILMED

2-85

DTIC