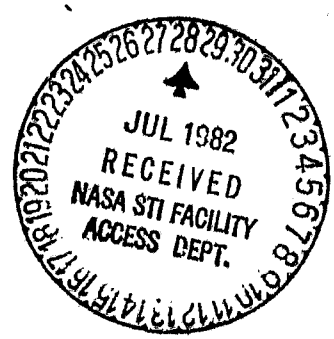
JPL PUBLICATION 82-50

LSI/VLSI Design for Testability Analysis and General Approach

Albert Y. Lan

June 1, 1982



Prepared for
Naval Ocean Systems Center
San Diego, California
through an agreement with
National Aeronautics and Space Administration
by
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL PUBLICATION 82-50

LSI/VLSI Design for Testability Analysis and General Approach

Albert Y. Lam

June 1, 1982

Prepared for

Naval Ocean Systems Center
San Diego, California

through an agreement with

National Aeronautics and Space Administration

by

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Naval Ocean Systems Center, San Diego, California, through an agreement with NASA.

ABSTRACT

For many years since the advent of large-scale integrated circuit technology, digital system testing has been a challenging problem. Intensive research efforts are directed towards the discovery of techniques (systematic and/or ad hoc) for designing digital systems with good testability and maintainability. This report presents a survey of most major contributions to the theory and practice of digital design for testability. Detailed analysis of each of the contributions is also presented, providing the reader with necessary background materials for the main objective of this report -- from the comparison of all design for testability techniques studied, some conclusions may be drawn to establish a general guideline/approach to designing testable circuits for large scale integration and very large scale integration.

ACKNOWLEDGMENT

This study was initiated by the Naval Ocean Systems Center, Code 923, and represents a facet of a block-funded program entitled Integrated Circuit Technology, sponsored by the Naval Electronics Systems Command, Technology Division. The work was performed by agreement with NASA under Contract NAS7-100 at the Jet Propulsion Laboratory of the California Institute of Technology. This program is continuing under NASA sponsorship, and related system studies are being conducted at the University of California, Los Angeles under sponsorship of the Office of Naval Research.

ACRONYMS

AC	Accumulator
ac (AC)	alternating current
ALU	Arithmetic Logic Unit
ATE	Automatic Test Equipment
BIT	Built-In Test
BITE	Built In Test Equipment
CM	Control Module
CPU	Central processing unit
CUT	Circuit Under Test
DC	Direct Current
DFT	Design For Testability
DMAR	Data Memory Address Register
DoD	Department of Defense
DP	Diagnostic Processor
EXOR	Logical Exclusive-or Function
FR	Flag Register
GLF	General Logic Function
GLS	General Logic Structure
HFPH	Hazard Free Polarity Hold
IC	Integrated Circuit
I/O	Input/Output
IR	Instruction Register
ITSC	Input Test Set Complex
LNCR	Liaison Network Control Registers
LSI	Large Scale Integration
LSSD	Level Sensitive Scan Design
MM	Memory Module
MOS	Metal Oxide Silicon
MSI	Medium Scale Integration
MUX	Multiplexer
NAND	Logical And-Not Function
NMOS	N-channel Metal Oxide Silicon
NOR	Logical Or-Not Function
OR	Logical Or Function
OTSC	Output Test Set Complex
PC	Program Counter
PCR	Partition Control Register

PFF	Parasitic Flip Flop
PLA	Programmable Logic Array
PM	Processing Module
PMOS	P-channel Metal Oxide Silicon
PU	Processing Unit
RAM	Random Access Memory
ROM	Read Only Memory
s-a-0	stuck-at-zero
s-a-1	stuck-at-one
SEC/DEC	Single Error Correction/Double Error Detection
SSI	Small Scale Integration
SRL	Shift Register Latch
SSRL	Stable Shift Register Latch
TSC	Totally Self Checking
ULM	Universal Logic Modules
UUT	Unit Under Test
VDD	Main Power Supply
VLSI	Very Large Scale Integration

CONTENTS

1 INTRODUCTION 1-1
1.1 TESTABILITY AND GOOD TESTABILITY 1-2
1.2 PREREQUISITES OF DESIGNING FOR TESTABILITY 1-3
1.2.1 Well Specified Design Framework 1-3
1.2.2 Well Defined Failure Universe 1-5
1.2.3 Implementation Technology 1-6
1.3 HARDWARE FAILURES, MOS CIRCUIT FAULTS AND ERRORS .. 1-6
1.4 INTRODUCTION TO DESIGN FOR TESTABILITY 1-10
1.4.1 On-Line on-Chip Testing 1-11
1.4.2 On-Line off-Chip Testing 1-12
1.4.3 Off-Line on-Chip Testing 1-13
1.4.4 Off-Line off-Chip Testing 1-14
1.4.5 General Approach 1-15

2 OVERVIEW OF MAJOR DESIGN FOR TESTABILITY TECHNIQUES ... 2-1
2.1 CLASSICAL APPROACH - DUPLICATE AND COMPARE TECHNIQUE 2-1
2.1.1 Morphic Boolean Algebra and Morphic AND gate 2-3
2.1.2 Totally Self-Checking Code Checkers and Comparators 2-9
2.1.3 Simple CMOS Implementation of Morphic AND Gate 2-12
2.2 LEVEL SENSITIVE SCAN DESIGN (LSSD) 2-14
2.2.1 Hazard-Free Polarity-Hold Latch and Shift Register Latch 2-15
2.2.2 Design Structure 2-17
2.2.3 Fundamental Testing Techniques of LSSD-Structured Circuits and Systems 2-18
2.2.4 Advanced Testing Techniques of LSSD-Structured Circuits and Systems 2-21
2.2.4.1 LSSD to NON-LSSD Interface 2-26
2.2.4.2 Utilization of Existing Latches in LSSD Logic 2-27
2.2.4.3 LSSD Parity Checking 2-27
2.2.4.4 On-Line Dynamic Scan 2-27
2.2.4.5 On-Line Error Detection in Memory ... 2-28
2.3 MODULARIZED DECOMPOSITION VIA MULTIPLEXED ROUTING . 2-29
2.3.1 Basic Circuit Decomposition Scheme 2-30
2.3.2 Generalized Routing Scheme 2-31
2.3.3 Reliability and Testability of Liaison Network 2-32

3 COMPARISON OF DFT TECHNIQUES 3-1
3.1 SELF-CHECKING TECHNIQUE 3-1
3.2 LEVEL SENSITIVE SCAN DESIGN TECHNIQUE 3-5
3.3 MULTIPLEXED ROUTING TECHNIQUE 3-8

4 A GENERAL APPROACH TO DESIGN FOR TESTABILITY 4-1
4.1 GENERAL DESIGN FOR TESTABILITY GUIDELINES 4-2

4.2 A GENERAL DFT STRUCTURE 4-4
4.3 TEST SET AND RESPONSE SET GENERATION AND STORAGE... 4-6

5 CONCLUSIONS 5-1

6 REFERENCES 6-1

APPENDIXES
A. PARASITIC FLIP FLOP CAUSES A-1
B. CIRCUIT FAULT EFFECTS ON TSC CIRCUITS B-1

Figures

1.1 Trend of Fault Coverage Obtained in Practical Cases Versus Network Size 1-20
1.2a Nearest Neighbor Fault Model 1-20
1.2b Neighborhood Fault Model 1-20
1.3 On-Line On-Chip Testing Block Diagram 1-21
1.4 On-Line Off-Chip Testing Block Diagram 1-22
1.5 Off-Line On-Chip Testing Block Diagram 1-23
1.6 Micro/Macrocell Structure Development 1-24
2.1 Block Diagram of TSC Circuit 2-34
2.2 2-Level NAND-NAND Logic Implementation of Morphic-AND Function 2-35
2.3 2-Level NOR-NOR Logic Implementation of Morphic-AND Function 2-35
2.4 Morphic-AND Function Truth Table 2-36
2.5 n-bit TSC Even Parity Checker 2-37
2.6 n-bit TSC Comparator 2-38
2.7 CMOS Morphic-AND Gate 2-39
2.8 n-bit TSC Parity Checker 2-40
2.9a Flow Table of Level-Sensitive Polarity-Hold Latch 2-41
2.9b Excitation Table, Excitation Equation, and Logic Implementation of a Level-Sensitive Latch 2-41
2.9c Excitation Table, Excitation Equation, and Logic Implementation of an HFPH Latch 2-42
2.10 Symbolic Representation, Excitation Equation, and Logic Implementation of a Shift Register Latch ... 2-43
2.11 Interconnection of SRLs for Scan-in/Scan-out 2-44
2.12a Basic Sequential Network Structure: Type 1 .. 2-45
2.12b Block Diagram of LSSD Single Latch Design Structure 2-45
2.12c Block Diagram of LSSD Double Latch Design Structure 2-46
2.13 Basic Sequential Network Structure: Type 2 2-47
2.14 Block Diagram of LSSD to non-LSSD Network Interface 2-48
2.15 LSSD Single Latch Design Structure Implemented with SSRLs 2-49
2.16 Logic Implementation of a Stable Shift Register Latch 2-50
2.17 Original SSRL Implementation 2-51

2.18 Example of Use of SSRL in LSSD Logic 2-52
2.19 LSSD Parity Checker Implemenatation 2-53
2.20 LSSD On-Line Memory Error Detection 2-54
2.21 Basic Network Decomposition 2-Module Partitioning 2-55
2.22 An Example of Network Partition 2-56
2.23a A 2-Module Routing Scheme 2-57
2.23b Block Diagram of Input/Output Buses Routing - G_1 under Test 2-57
2.24 Protocol of a Routing Circuit of a Single Module . 2-58
2.25 Block Diagram of a 3-Module Routing Design 2-59
2.26 A Generalized Routing Model 2-60
2.27 An Implementation of a 1-out-of-4 Multiplexer 2-61
3.1 Block Diagram of an IBM S/360 Processing Unit 3-14
3.2 Scheme for Decoder Checking 3-15
3.3 DFT Techniques Comparison Table 3-16
4.1 A General DFT Structure 4-12
4.2 Block Diagram of the Structure of Intel 8080 Microprocessor 4-13
4.3 Real-Time Monitoring of Critical Module, an Example 4-14
4.4 General "Store/Generate and Compare" Technique ... 4-15
4.5 General "Store and Generate Built-in Testing" 4-15
4.6 Block Diagram of Test Set "Store and Generate" Scheme 4-16
4.7 Block Diagram of Syndrome-Test Scheme 4-17
A1 Two-Input CMOS NOR Gate A-3
B1 NOR Gate Parasitic Flip Flop Fault Detection vs. Input Patterns B-4

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. JPL Pub 82-50	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle LSI/VLSI Design for Testability Analysis and General Approach		5. Report Date June 1, 1982 6. Performing Organization Code
7. Author(s) Albert Y. Lam		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91109		10. Work Unit No. 11. Contract or Grant No. NAS 7-100 13. Type of Report and Period Covered JPL Publication
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code X141-40-00-01-37

15. Supplementary Notes

16. Abstract

For many years since the advent of large-scale integrated circuit technology, digital system testing has been a challenging problem. Intensive research efforts are directed towards the discovery of techniques (systematic and/or ad hoc) for designing digital systems with good testability and maintainability. This report presents a survey of most major contributions to the theory and practice of digital design for testability. Detailed analysis of each of the contributions is also presented, providing the reader with necessary background materials for the main objective of this report -- from the comparison of all design for testability techniques studied, some conclusions may be drawn to establish a general guideline/approach to designing testable circuits for large scale integration and very large scale integration.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Integrated Circuits Electronics and Electrical Engineering Computer Operations and Hardware Computer Systems	Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	126	

JPL 0184 1180

SECTION 1

INTRODUCTION

Integrated circuit technology is moving from SSI/MSI to LSI/VLSI. The problem of circuit testing (at chip, board, and system levels) is thus aggravated not proportionally but exponentially. Since the advent of SSI, fault modeling techniques have been exclusively used for test generation/fault simulation, and exhaustive explicit testing of SSI/MSI devices has been possible and practical. For LSI/VLSI devices, however, fault modeling and exhaustive testing are clearly impractical if not impossible. For instance, as many as 30 defects could be associated with a two input AND gate and a 10,000-gate chip could contain 300,000 defects, each of which may be a fault cause. This large number of faults, plus the multiplicity of fault occurrences, tremendously complicates the modeling problem. As for exhaustive testing, the total number of test vectors N, the number of inputs m, and the number of storage elements n have the following relationship:

$$N = 2^{m+n}.$$

Therefore, an LSI chip with 40 inputs and 140 storage elements may require 2^{180} test vectors. If these vectors are run at a rate of 10^7 vectors per second (10 MHz), it will take 10^{47} seconds or 3×10^{39} years to run an exhaustive functional test on the chip once [1]. Figure 1.1 shows the trend of fault coverage obtained in practical cases versus network size [2]. We see that if the network (sequential) contains 2000 or more elements, the ability

of Automatic Test Equipment (ATE) to generate tests decreases to unacceptable levels.

Therefore, incorporating testability characteristics into a digital design such that the final product (the device) can be tested cost-effectively should be the basic guideline to reliable IC design.

1.1 TESTABILITY AND GOOD TESTABILITY

Let us define the terms "Testability" and "Good Testability". "Testability" may be defined as the ability to determine the status (functioning as specified or not functioning as specified) of the unit under test (UUT) within a prescribed time period. A testing or design technique that leads to "Good Testability" of the UUT should provide the following desirable features [3] :

(1) Contains no logical redundancy;

(2) Reasonable test set size;

(3) Test set can be derived fairly easily;

(4) Results of the test are easily observable at the outputs;

(5) Results of the test can be interpreted fairly easily;

(6) Good fault coverage for a specified fault set;

(7) Faults locatable to the desired degree.

If additional hardware and I/O's are used to enhance testability, the following features are also desirable :

(8) Reasonable hardware overhead;

(9) Minimal additional I/O pins;

(10) Minimal functional processing speed interference.

Although not quantitative, the above features of "Good Testability" give some means for measuring and evaluating the testability of a digital design. We will use these features as a benchmark to qualitatively measure and evaluate the effectiveness and efficiency of the DFT techniques discussed in sections 2 and 3.

1.2 PREREQUISITES OF DESIGNING FOR TESTABILITY

Design for testability can become an accepted practice only if the following prerequisites are established and well understood :

1.2.1 Well Specified Design Framework

(i) What design techniques are to be employed ?

(a) On chip testing* only:

- Hardware built-in test (BIT)
- Hardware/Software BIT

(b) Off chip testing* only:

- Design for easily testable logic functions
- Architectural design to enhance testability
- Modularized design approach
- Design for ATE compatibility

(c) On chip and Off chip testing:

- A combination of part or all of the above techniques.

In order to achieve good testability and optimal life-cycle cost, on chip hardware/software BIT is a must. For

* See section 1.4 for descriptions.

instance, operation/support costs may run as high as 50% of its total life-cycle cost for a typical weapon system over its life-cycle (about 20 years), (Engineering evaluation 3-4%, Prototype/Preproduction 12%, Production 35%)[3]. Hence operation/support of the system represents the primary cost factor. Moreover, DoD studies [3] show that systems designed with built-in testability achieve up to 70% of life-cycle cost reduction. Therefore, incorporating built-in testabil''y into the design of a system not only enhances its testability but also reduces its ownership cost.

(ii) What are the standards for measuring and evaluating testability ?

A standard tool is needed to relate testability measures to system availability and life-cycle cost, to calculate the testability of a design, and to relate a design in progress to the requirements. The testability measures can be defined in terms of the "CONTROLLABILITY" and "OBSERVABILITY" of the circuit. A paper by Stephenson and Grason [4] gives formulae for assigning weights to these parameters. Also, an IBM 360 program which is based on this paper has been written at Bell Telephone Laboratories as a tool for measuring the testability of an arbitrary circuit. This program is also reported to be able to indicate the difficulty of test generation. A circuit,

if shown difficult to test, can be modified before being released for test generation.

1.2.2 Well-defined Failure Universe

It is very important to specify a finite and realistic "failure universe" (fault set) to serve as the basis of testability design and evaluation. For LSI/VLSI circuit (including memory circuits), stuck-at, bridging, floating, parasitic flip flop, and pattern sensitive faults are the common circuit faults that may produce error(s)* at the output(s). All of these faults originate from two types of hardware failures : Manufacturing Defects and Wear-out Mechanisms. In the following subsection, we will discuss hardware failures, MOS LSI/VLSI circuit faults and errors in greater detail.

Different fault sets accepted as a coverage basis often result in different approaches to designing for testability. The more types of faults the product (the device) is expected to cover (i.e., the occurrence of any of the specified fault types can be detected in accordance with the prescribed testability requirement(s)), the less is the possibility of achieving such expectation.

* A chain of faults may originate from the source of hardware failure. As the fault propagates and affects the circuit output(s), we say an error results.

1.2.3 Implementation Technology

It is critical for the circuit designer to be able, while considering all design and testability specifications, to determine what implementation technology is to be employed for the realization of his/her design. For example, with increasing integration (SSI/MSI to LSI/VLSI), both the memory and general purpose logic device manufacturers recognize the inadequacy of the classical stuck-at fault model for characterizing their new products. Neighbor intraction and interaction are becoming more of a problem. Hence guidelines for DFT, fault model(s), and testing techniques all have to be revised. Further, as will be shown in the following subsection, the CMOS implementation and the NMOS implementation of a design have different impacts on DFT. Different approaches have to be employed to fulfill the same testability requirements.

1.3 HARDWARE FAILURES, MOS CIRCUIT FAULTS AND ERRORS

Hardware failures are mainly due to (1) manufacturing defects such as lithographic misalignment, wafer imperfection, etc. and (2) wearout mechanisms, such as metal migration and ion contamination. Hardware failures may result in the following kinds of circuit faults common in MOS circuits [5] :

(1) Stuck-at Fault - the logical value of a line is always stuck at 1 or 0. This kind of fault is caused by (1) oxide pinhole, (2) missing or defective pull-down transistors, (3) pull-down

transistors source-to-drain shorts, and (4) pull-up transistors source-to-drain shorts.

(2) Bridging Fault - two adjacent conducting layers or overlapping layers short together. Lithographic misalignment is one of the causes of this fault type.

(3) Floating Fault - a conducting wire is separated or broken and may "float" between logic 1 and logic 0 depending on the physical geometry and electrical enviroment of its neighborhood. Missing or defective pull-up transistors; missing contact cut or metal cover; and metal, polysilicon, and diffusion opens are the causes of this fault type.

(4) Parasitic Flip Flop (PFF) Fault - This fault has five potential causes as listed in appendix A. The occurrence of this fault in a combinational gate (a NOR gate, for example) can turn the gate into a D flip flop type sequential circuit. Hence, the logic function of the affected circuit is altered in a very unpredictable fashion. Interestingly, CMOS circuits may have PFF faults, but NMOS circuits may not have them unless implemented with enhancement mode pass transistors only. Therefore, in the design for testability of CMOS devices, one may have to consider this PFF fault problem. A method is presented in appendix B to eliminate the undesirable effects due to PFF faults. This method is effective but somewhat disadvantageous in the sense that its implementation in a circuit may slow down the processing speed significantly.

(5) Pattern Sensitive Fault - this type of fault occurs in high density memory circuits (a 64K RAM chip, for example). It may also occur in high density general logic devices such as microprocessors. In high density general logic circuits, switching of a conducting wire (master wire) may cause switching of a neighboring wire (slave wire) to the same logic value, depending on the current density in the master wire. Hence this fault may be viewed as an intermittent bridging fault. For memory circuits, there are two pattern sensitive fault models:

(a) Nearest Neighbor fault model (figure 1.2a) - when the content of a memory cell (master cell) is switched, the contents of any one or all of its nearest neighboring cells (slave cells) are inductively switched.

(b) Neighborhood fault model (figure 1.2b) - this model is similar to the nearest neighbor fault model except that the affected neighboring (slave) cells cover a wide area of the master cell.

The following shows the test pattern requirements for a 64K RAM for the stuck-at, nearest neighbor, and neighborhood fault models with N representing the number of memory cells [1].

Fault Model	Pattern	Number of Test Vectors
Stuck At	$2N$	1.311×10^5
Nearest Neighbor	$2N^{3/2}$	3.355×10^7
Neighborhood	$2N^2$	8.590×10^9

The neighborhood model requires almost five orders of magnitude more test vectors than the stuck at model. To fully test a 64K RAM with test cycle rate of 375 ns, it takes 49ms/3221sec if the stuck-at/neighborhood model is used [1].

It is seen that MOS LSI circuit faults fall into five catagories, namely, CLASSICAL STUCK-AT, BRIDGING, FLOATING, PARASITIC FLIP FLOP, and PATTERN SENSITIVE faults. These faults are, however, caused not only by manufacturing defects but also sometimes by wearout mechanisms in field use. There are two major wearout mechanisms:

(1) **METAL MIGRATION** -- which eventually results in wire separation. Electrical current in aluminum conductors induces movement of aluminum material in the conductors and eventually leads to separation of conducting wires. A metal wire disconnected from a signal source can result in a floating fault. This natural migration process can be reduced by reducing the current density in the conductors.

(2) **SODIUM ION CONTAMINATION** -- which eventually results in an oxide pinhole at the input of a n-channel enhancement mode transistor. Sodium ions tend to move towards and accumulate in regions of relatively high electrical field (for example, under transistor control input ports). Continuous accumulation of

sodium ions in a region will eventually result in that region being shorted to ground and equivalently stuck at zero. Sodium ion contamination has been found to be the cause of 35% of field failures [6].

Since the above postulated circuit faults are the most common faults in MOS LSI/VLSI devices, they may be considered as the elements of our "Fault Set" or "Failure Universe". Every DFT technique that we will discuss later on deals with these fault elements only. However, multiple, random occurrence of any or all of these faults in a circuit is assumed and considered. Some of the techniques have the restrictions that in the circuit under test, the fault is single and/or the errors at the output(s) are unidirectional*.

1.4 INTRODUCTION TO DESIGN FOR TESTABILITY

In section 2 of this report, we will present an overview of most of the major DFT techniques developed in the academic and industrial sectors. Here we first introduce the general concepts of testing of digital systems, networks and devices. Also we want to discuss some basic problems of design for testability and some general approaches to these problems.

---

*A unidirectional error is a random, symmetric, multiple error where all the fault bits fail to the same direction. Here "symmetric" means that both 0->1 and 1->0 are possible.

It is generally known that the cost and level of difficulty of circuit testing is an exponential function of the complexity of the circuit under test. The following shows the cost to detect and diagnose a fault at different circuit levels [2].

Cost to detect and diagnose a fault at different levels

CHIP	BOARD	SYSTEM	FIELD
$0.30	$3.00	$30.00	$300.00

We see that if a fault can be detected at the earliest stage, then the cost per fault detection can be substantially reduced. Further, as we have stated earlier, testing of VLSI devices is far from a fully solved problem. Testing of systems of a good quantity of VLSI devices at this stage would not only be infeasible but impossible. Therefore, much attention should be given to the level of testability at the chip and sometimes, board levels.

Chip testing techniques can be grouped into two categories : (1) On-Line Testing and (2) Off-Line Testing. Both of these categories have two subcategories: (1) On-Chip Testing (Built-In Test) and (2) Off-Chip Testing.

1.4.1 On-Line on-Chip Testing [7,8,9] (Figure 1.3) :

All testing is concurrently performed while the chip is in normal

operation or in idle state. All or most of the supporting facilities for the self-testing, may they be hardware and/or software, are built into the chip. This means that additional hardware (monitors, microprocessors, etc.) and/or additional non-volatile memory for test/diagnosis programs storage have to be installed into the chip which may result in a substantial increase of chip area overhead. The employment of this technique is most justifiable and beneficial to the performance/reliability of totally self-supporting systems such as space-borne digital systems.

1.4.2 On-Line off-Chip Testing [10,11] (Figure 1.4):

This is also concurrent testing, but the monitoring/testing hardware/software are all built off the chip. A self-testing distributed multiprocessor network is an example. Processors in different chips test/check each other periodically. In case of error occurrence, according to some predetermined algorithms and sequences, the majority processors "out-vote" the minority processor(s) (usually a single processor) which is then subject to some rollback and recovery attempts. If such attempts are unsuccessful, the minority processor is cut off from the network, and the majority processors may take over and share the tasks of the "cast-away" processor(s). In this case, the processing speed is decreased. Another way is to use "spare" processor(s) to replace the "abandoned" one(s). But this has the disadvantage of in-

creased hardware cost and redundancy. This testing scheme has some other disadvantages which will be discussed in section 3.

1.4.3 Off-Line on-Chip Testing [12,13,14,15,16,17,18] (Figure 1.5): This testing scheme is generally known as "Built-In-Test/Built-In-Test Equipment" testing and is viewed as the most useful and promising technique(s) to improve and enhance testability of LSI/VLSI devices. In this scheme, additional logic is first built into the chip to monitor the functional core (circuit under test (CUT)), to increase controllability and observability of the CUT, and/or to allow for testing with reduced/compressed test data set. With the help of automatic test equipments (ATE), the testing process is greatly simplified. Also in some designs, the additional built in logic which is in the form of totally self-checking (TSC) checkers, will inform the outside world of most functional or signal transmission errors, thus enhancing the diagnosibility of the device.

One of the drawbacks of this scheme is that the checking and testing of the device is not quite spontaneous (concurrent), which may aggravate the error latency problem in some critical systems. This problem is intrinsic even in systems with concurrent self-testing/self-checking capabilities. Integrated circuit manufacturers, however, favor this scheme because of its good cost-effectiveness and excellent fault coverage of their prescribed fault set. They can minimize the error latency problem of their products in field use by specifying stringent dynamic

specifications such as maximum clock cycle time and by recommending automated dynamic diagnostics of critical systems.

In sections 2 and 3 we will study some "BIT/BITE" testing techniques and discuss their advantages and disadvantages.

1.4.4 Off-Line off-Chip Testing

In this scheme, very little is done to the original functional core. The only additional logic that may be built in are circuits at the two ends (input and output ports) of the CUT for the enhancement of ATE compatibility. Very complex testing algorithms and data sets may be required and for LSI/VLSI devices, the testing process, if effective, is usually very time-consuming and costly.

As will be discussed in section 4 of this report, different design, performance, and testability requirements would call for different DFT techniques and strategies. The following lists some basic problems of DFT and some general approaches to these problems.

Basic Problem

---

1. The complexity of the device - the vast number of gates in LSI/VLSI devices ranges from 500 to 50,000. This poses problems for the ATE to assume its diagnostic role as a general IC testing equipment.

2. The inaccessibility of individual cells - limited pin-to-gate ratio restricts the accessibility of cells in a chip. Testing the cells individually would simplify the test generation and testing processes.
3. The irregular structure of general purpose logic devices - the more irregular the structure of a device, the more difficult and complex test generation/fault simulation and testing procedures are.
4. Limited availability of pins - the incorporation of a good quantity of built-in testing logic may be impossible due to the shortage of pins available for the primary inputs and outputs of these additional logic.
5. The difficulties in testing sequential logic - no adequate and efficient test generation algorithms for LSI/VLSI sequential devices exist. Some alorithms, however, are useful for testing large combinational (latch free) circuits (e.g. >5000 gates) [19,20].
6. Checking the checker problem - Additional circuits (checkers) are often built into the devices to monitor the operations and signal propagations. Hence correct functioning of these checkers is vital. To aviod the checking the checker problem, the monitoring circuits must at least be built self-checking.

1.4.5 General Approach

1. Modularization - To ease the problem of employing ATEs as general testing equipment for complex LSI/VLSI devices,

modularized design of the functional core may be a good approach. The functional core is divided into several sub-functional logic entities called macrocells. Each of these macrocells consists of several modules called microcells which may be identical or readily available from the microcell libraries of the CAD system (figure 1.6). The upperbound of the microcell baseline may be restricted to 800-1200 gates [3]. The microcells may be PLAs (programmable logic arrays [21]), GLSs (general logic structures [22]), GLFs (general logic functions [23]), ULMs (universal logic modules [24]), and cellular arrays [24]. Each of the macrocells may contain up to 12,000 gates [3] and should be made accessible and observable through the primary inputs and outputs of the device, either directly or indirectly through multiplexed routes. This technique not only eases the problems of automatic testing and fault simulation but also facilitates the design process. The irregular structure of the general logic devices can be made regular by implementing the defined circuit functions with the aforementioned general microcell circuit structures.

2. Multiplexed Routing - The accessibility of each of the cells of a device can be drastically increased by putting in multiplexed signal propagation routes between primary inputs/outputs and individual cells [16]. Bus multiplexers are placed at the input and output buses of each cell such that the primary inputs/outputs can be directly connected(routed)

to the inputs/outputs of the cell. If the device can be partitioned into many cells such that each cell has only a few inputs and outputs, exhaustive testing can be performed on each cell, and test generation is virtually eliminated.

3. Level Sensitive Scan Design (LSSD) [12] - The enormous number of internal states of a large sequential network makes it impossible to effectively test such a network. The LSSD technique, first proposed by IBM, not only can be used as an effective tool for the structured design of easily testable sequential devices but also for "System Bring-up" and for field serviceability. Basically, this technique allows a sequential network, while switched to the "Test" mode, to be physically converted into a combinational (latch-free) network. Therefore, along with the employment of the multiplexed routing and modularization design techniques, a large combinational/sequential device can be cost-effectively tested.

4. Totally Self-Checking (TSC) Checkers/Comparators and Fault Monitors - The incoporation of signal flow checkers and circuit fault monitors into a design is essential to the upgrading of the reliability and testability of such a design. It is clear that these checkers and monitors should be, if not failure-proof, totally self-checking for reliable performance. It has been shown [25] that a class of totally self-checking circuits, which can be used as checkers and monitors, can be designed using a very simple coding scheme called the "1-out-

of-2 code" (see section 2). Basically, if there is any single fault in a TSC circuit, the output will be a non-code word which is interpreted as an error, indicating malfunctioning.

5. A General Structured Approach to Design for Testability - All of the aforementioned DFT techniques are somewhat non-general in the sense that their applications are only practically suited for some specific circuit structures. To provide a general scheme for the design for testability, a general structured approach is needed. At the present time, no such approach has been proposed. A DFT technique called the "Store and Generate" technique, proposed by Agarwal and Cerny [18], may be by far the most promising scheme ever proposed. Although several unresolved problems still exist, as mentioned in their paper, this scheme provides a general circuit design structure which, with little modifications, can be adapted to any set of DFT specifications and requirements.

   As suggested by its name, this technique has the test set and the calculated test response either stored and/or generated on chip. Thus, periodic real-time testing of the CUT is possible. The CUT is first divided (with additional hardware as described in (2) of subsection 1.3) into several macrocells. All sequential elements in each macrocell are converted into latch-free combinational elements (using LSSD), and thus, full testing of each macrocell becomes possible and practical. Using TSC checkers and comparators, the actual test response

of the CUT is compared with the pre-stored/generated calculated response. Any errors resulting from circuit faults and/or signal transmission errors in the CUT are detected. Quite clearly, this technique can achieve a relatively large fault coverage.

There are still several unresolved problems. The main problem is the large storage requirement for the test set and the calculated response set. Although the test set can be generated on chip without much difficulty, the on-chip generation of the calculated response set is non-trivial. Trade-offs between the storage and generation of the test and response sets have to be made. Scheme(s) for lossless compression of these two data sets should also be investigated.

In the following section (section 2), we will present an overview of several useful techniques and schemes for the enhancement of testability and maintainability of LSI/VLSI devices. In section 3, we will discuss the relative strong and weak points of these techniques. In section 4, we will try to merge some of these techniques together to form a general structured approach to designing testable LSI/VLSI devices.

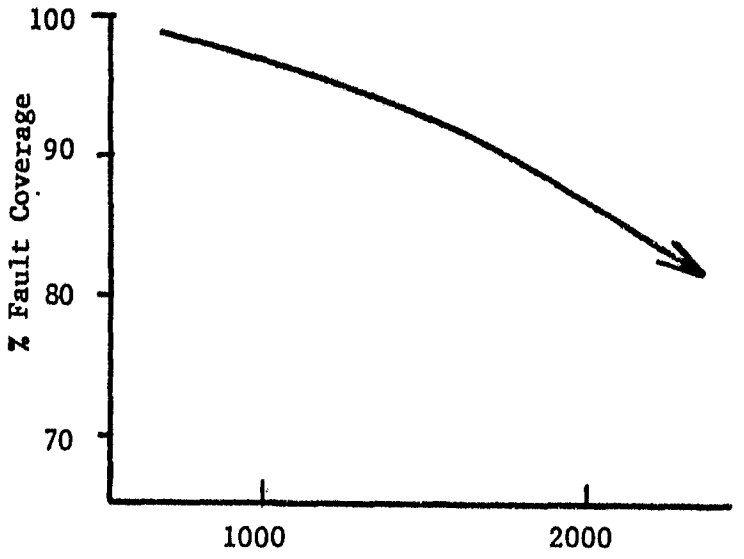


Figure 1.1 Trend of Fault Coverage Obtained in Practical Cases Versus Network Size

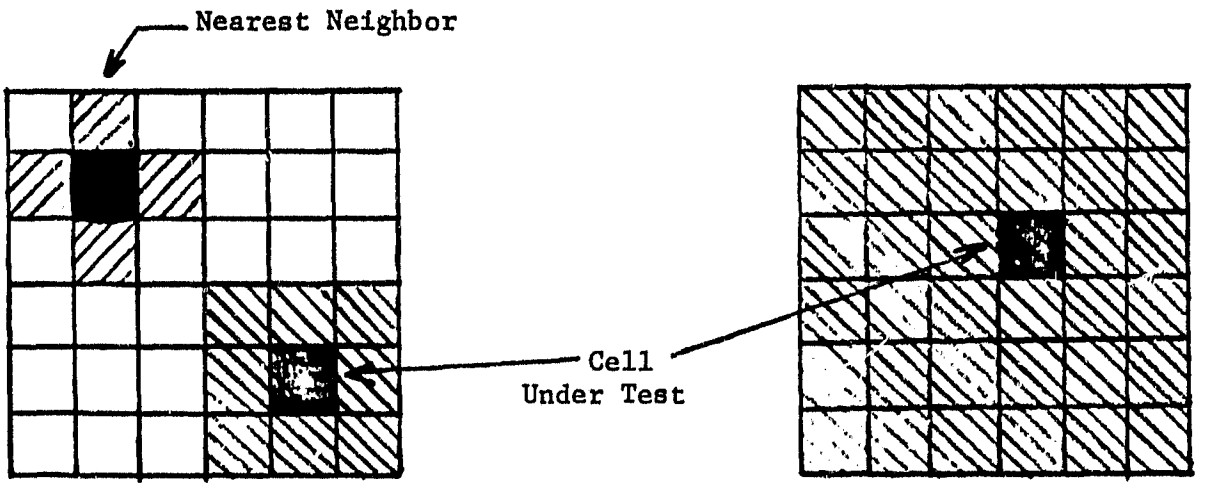


Fig. 1.2a Nearest Neighbor Fault Model

Fig. 1.2b Neighborhood Fault Model

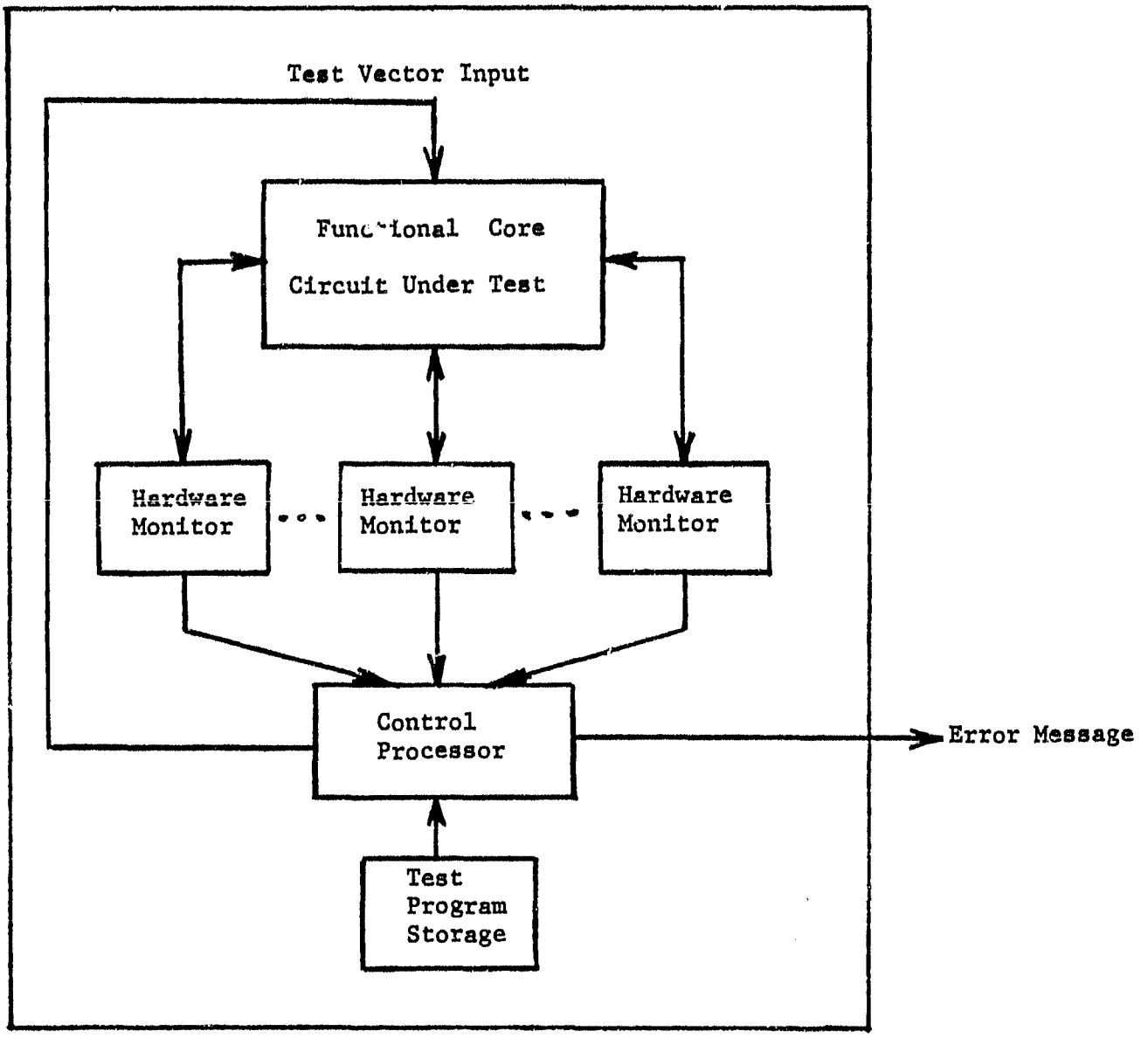


Figure 1.3 On-Line on-Chip Testing Block Diagram

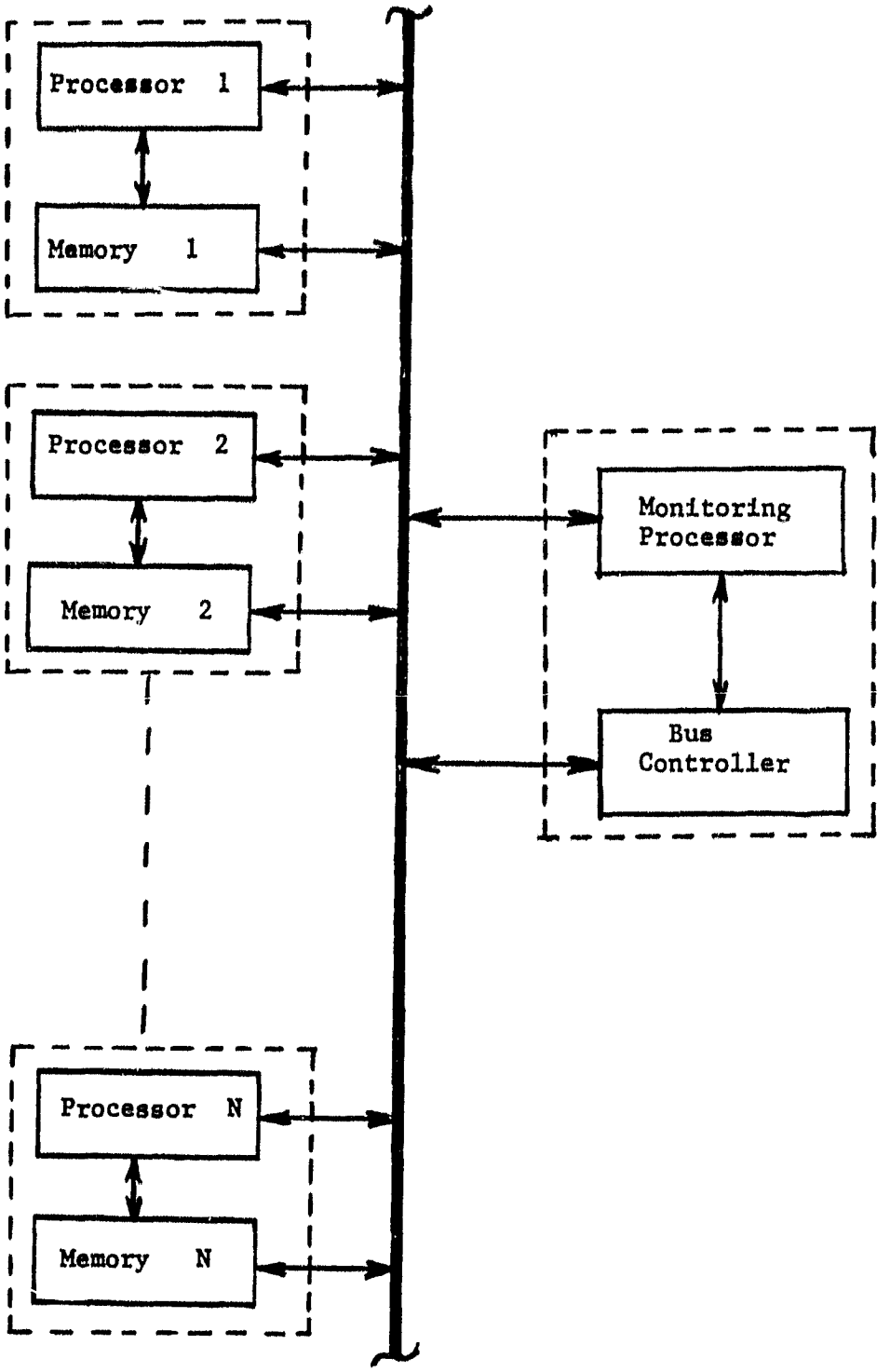


Figure 1.4 On-Line off-Chip Testing Block Diagram

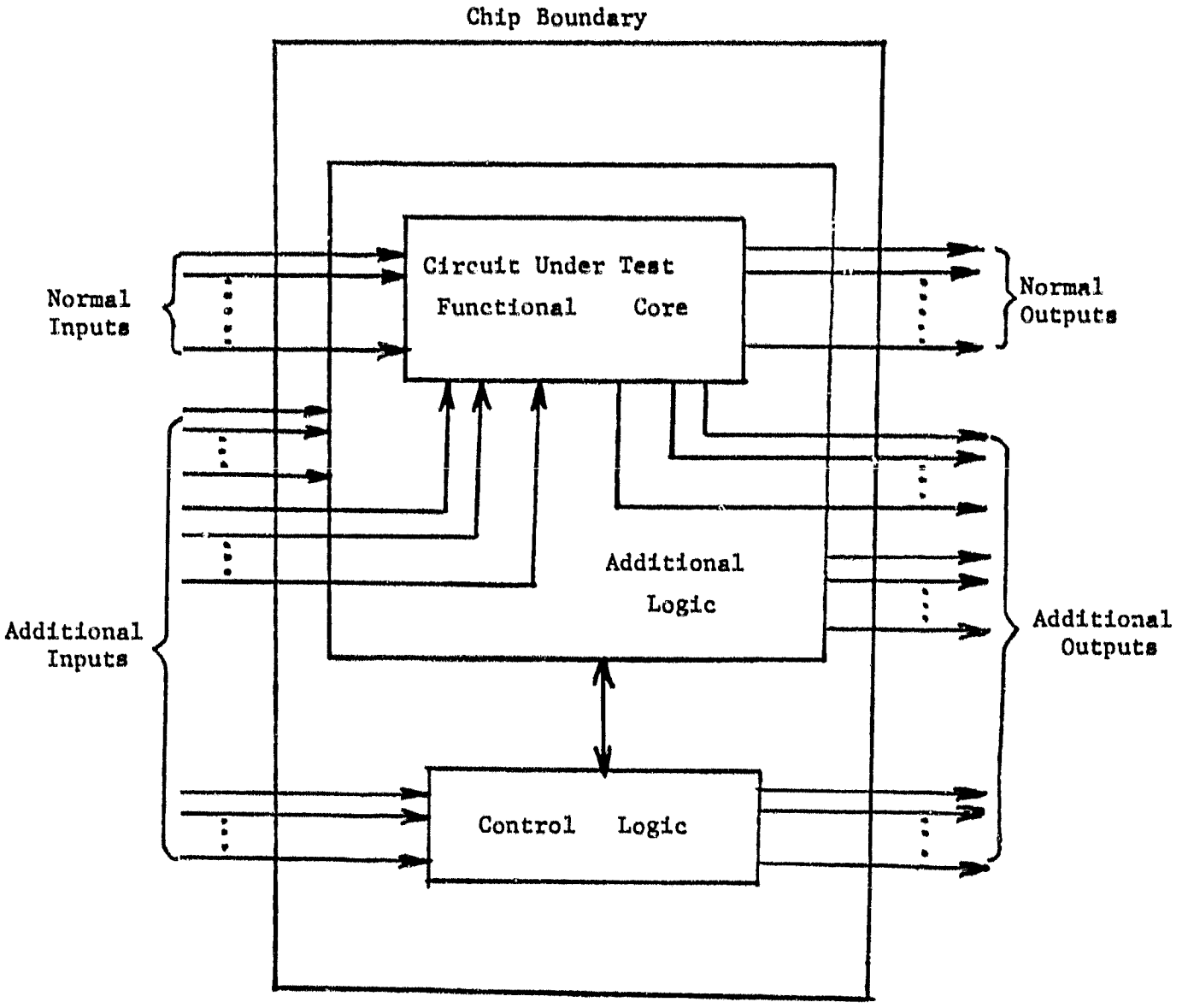


Figure 1.5 Off-Line on-Chip Testing Block Diagram

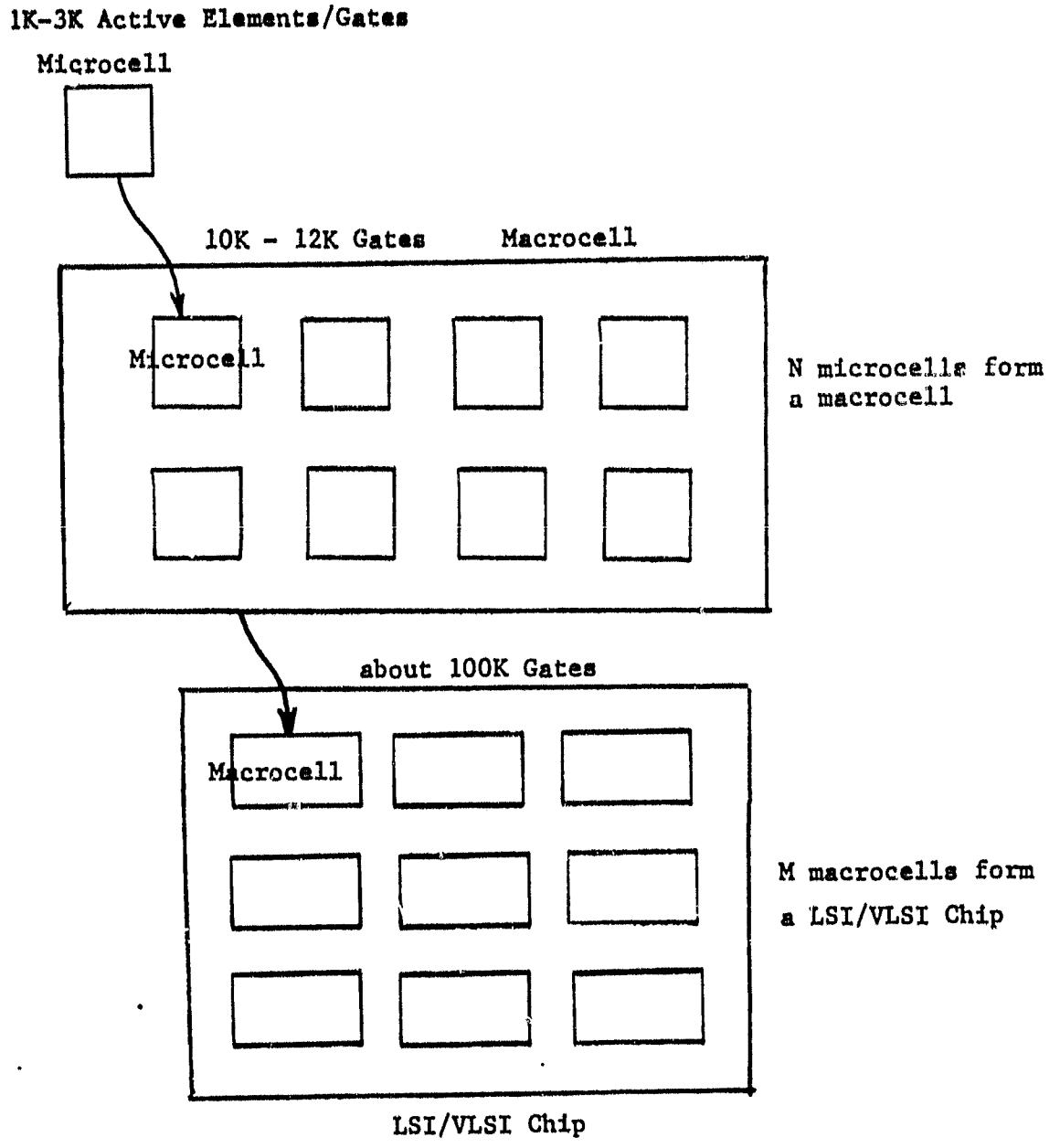


Figure 1.6 Micro/Macrocell Structure Development

SECTION 2

OVERVIEW OF MAJOR DESIGN FOR TESTABILITY TECHNIQUES

In this section we will discuss the design principles of most major DFT schemes and techniques.

2.1 CLASSICAL APPROACH - DUPLICATE AND COMPARE TECHNIQUE

Quite clearly, it is desirable that if a device is to be designed for testability and maintainability, it should be made capable of informing the user of any malfunctions or errors during operation. The classical approach to achieving this desirable feature is very simple in principle:

> The circuit under test is duplicated on chip (figure 2.1). During normal operation, the input vectors are applied to both circuits. The responses of the circuits are then compared for identicalness, otherwise circuit malfunction or error is said to have occured.

Clearly this scheme has a major disadvantage - hardware redundancy. Nevertheless, there are also some <u>technical</u> problems to be resolved before this scheme becomes applicable:

1. The assurance of valid input vectors to both of the circuits under test - Since the input vectors from the primary input port of the device are split and sent to the inputs of the two CUTs, they might be erroneous due to transmission problems or circuit faults on input buses A and/or B. Hence, both of the input vectors should be checked for validity just before they

are applied to CUT A and CUT B. This can be accomplished by first encoding all input vectors according to some selected coding scheme (parity coding, for instance) and then having these input vectors checked by some code-checkers, as shown in figure 2.1. If there is an erroneous input vector to either one or both of the CUTs, the code-checkers should flag this error to the user at the "Input Vector Error" pin. In the following subsection (2.1.2), we will present a design of such code checkers.

2. The assurance of correct functioning of the checkers and comparators - As mentioned in section 1.4, the checkers and comparators should be designed to be totally self-checking to guarantee the validity of the error flags (i.e. the "Input Vector Error" flag and the "Circuit Malfunction" flag). At all times, if there are no circuit faults inside the checkers and comparators, then the checkers will be fully capable of examining the input words for code-validity and so will the comparators in determining the identicalness of its two input vectors. If there are some circuit faults inside the checkers and the comparators, then the checkers and comparators may still be functioning for most of the time. The properties of the sequence of the input vectors to these TSC circuits diversely affect the checking capabilities of these TSC circuits under the influence of circuit faults.

   Nevertheless, in this SELF-CHECKING design scheme, the checkers should be able to flag <u>internal circuit faults</u> as well as

input vector errors, provided there are no circuit faults. As for the output comparator, it should also be able to flag internal circuit faults as well as output errors from either or both of the CUTs.

The design principles and techniques of TSC circuits are shown in the following subsection.

2.1.1 Morphic Boolean Algebra and Morphic-AND Gate

A mapping from the two-valued Boolean variable to Morphic Boolean variable is defined as follows :

$$M : [(e1,e2),(\overline{e1},\overline{e2})] \rightarrow 0$$

$$[(e1,\overline{e2}),(\overline{e1},e2)] \rightarrow 1$$

where e1 and e2 are elements of the set {0,1}.

Hence while in the positive Boolean algebra, TRUE and FALSE are represented by 1 and 0 respectively, Morphic TRUE (1_M) and Morphic FALSE (0_M) are represented by either (0,1) or (1,0) and (0,0) or (1,1) respectively. Morphic Boolean operations are defined exactly as those of the two-valued Boolean algebra. For example : Morphic-AND Function

$1_M * 0_M = 0_M$ <---> (0,1) * (1,1) = (1,1)

$1_M * 1_M = 1_M$ <---> (1,0) * (0,1) = (1,0)

where * is the AND function.

It is not difficult to see that the mapping of the two-valued Boolean algebra into morphic Boolean algebra can be used to solve

the detection problem of stuck-at faults of a single line which is semi-passive, i.e., a line that has a constant value under normal operation. Suppose in some digital network there is a line, A, whose logic value is always 1 under normal operation. There is no way, under normal operation, to detect the stuck-at-1 fault of this line. However if the network is designed with morphic logic instead, then line A will be mapped into two lines A1 and A2. Since (A1,A2) can take on either (0,1) or (1,0) under normal operation, if the same type of fault occurs at line A1 or line A2, it will eventually be detected because sooner or later, even under normal operation, (A1,A2) will become (1,1) or (0,0), which represents error if we assume a 1-out-of-2 code.

Definition 2.1 : 1-out-of-2 Code. A codeword (valid) is defined as Morphic logic 1 (1_M) while a non-codeword (invalid) is defined as Morphic logic 0 (0_M).

One of the logical elements that can be used as the basic building block of morphic circuits is called the Morphic-AND gate. Let $A=(a,a^*)$ and $B=(b,b^*)$ be two morphic variables and $Y=(y,y^*)$ be a morphic function of A and B, then the Morphic-AND function is defined as follows :

$$[Y=(y,y^*)] = [A=(a,a^*)] * [B=(b,b^*)]$$

where $y = ab^* + a^*b$

$$y^* = ab + a^*b^*$$

Figures 2.2 and 2.3 show the 2-level NAND-NAND logic and NOR-NOR logic implementations of the Morphic-AND function. Figure 2.4 shows the truth table of the Morphic-AND function. From the truth

table we can see that if we restrict all input vectors as being 1-out-of-2 codewords only (the last four cases in the truth table), the output of the gate will be a valid codeword. Any non-codeword input (the remaining twelve cases) resulting from a transmmsion problem or bus faults will generate a non-codeword at the gate output, which indicates error.

The Morphic-AND gate also self-checks for any single internal stuck-at fault, as proved in the following :

Definition 2.2 : Let X, Y, and F represent the sets of valid codeword inputs, valid codeword outputs, and prescribed faults.

Let $x_i \in X$ be a codeword input, $y_i \in Y$ be a codeword output, and $f_i \in F$ be a fault. A circuit is said to be **FAULT-SECURE** if for every correct input x_i, y_i is the correct fault-free output and y_i' is the output in the presence of a fault f_i, then $y_i' = y_i$ if y_i' is a valid codeword or else y_i' is an invalid non-codeword.

A circuit is said to be **SELF-TESTING** if for every fault f_i, there exists at least one valid input x_i for which the resulting output, y_i, is a non-codeword.

Definition 2.3 : A circuit is said to be **TOTALLY SELF-CHECKING** if and only if it is both **FAULT-SECURE** and **SELF-TESTING.**

To prove that this Morphic-AND gate is totally self-checking for

any single internal stuck-at fault it suffices to show that if there is a stuck-at fault on either line L1 or L2 in figure 2.2, the circuit remains fault-secure and self-testing.

Line L1 s-a-0 case : $y' = a^{*}b$

$y^{*'} = y^{*}=ab+a^{*}b^{*}$

From the truth table in figure 2.4, we see that there are, out of the sixteen input patterns, four valid codeword patterns. The output of the faulty circuit (y' and $y^{*'}$) due to these four input patterns are shown in the following table:

Input Pattern	a	a^{*}	b	b^{*}	y'	$y^{*'}$	Output Pattern
1	0	1	0	1	0	1	1
2	0	1	1	0	1	0	2
3	1	0	0	1	0	0	3
4	1	0	1	0	0	0	4

Input patterns 1 and 2 yield output patterns 1 and 2 which are the same outputs as if the circuit is fault-free. Input patterns 3 and 4 yield invalid non-codeword outputs. Hence the circuit is both fault-secure and self-testing.

Line L1 s-a-1 case: $y'=b^{*}+a^{*}b$

$y^{*'}=y^{*}$

The responses of the faulty circuit are shown as follows:

Input Pattern	a	a*	b	b*	y'	y*'	Output Pattern
1	0	1	0	1	1	1	1
2	0	1	1	0	1	1	2
3	1	0	0	1	1	0	3
4	1	0	1	0	0	1	4

Output patterns 1 and 2 show that the circuit is self-testing.

Output patterns 3 and 4 show that the circuit is fault-secure.

Hence the circuit is totally self-checking.

Line L2 s-a-0 case:

In this case, y' is always 1. Hence the responses are:

Input Pattern	a	a*	b	b*	y'	y*'	Output Pattern
1	0	1	0	1	1	1	1
2	0	1	1	0	1	0	2
3	1	0	0	1	1	0	3
4	1	0	1	0	1	1	4

Output patterns 1 and 4 show that the circuit is self-testing.

Output patterns 2 and 3 show that the circuit is fault-secure.

Line L2 s-a-1 case:

This fault is equivalent to line L1 s-a-0.

Since by symmetry, the stuck-at faults of lines L1 and L2 represent all possible single stuck-at fault in the circuit, the circuit is thus proved to be totally self-checking for any single stuck-at fault.

Note that Morphic-AND gates are not only totally self-checking for single stuck-at faults but also for single bridging, floating, and parasitic flip flop faults. The effect of circuit faults on TSC circuits is dicussed in appendix B.

It is very important to note that this circuit is TSC as long as there is only one circuit fault. It may not be TSC if (1) there is more than one circuit fault or (2) there are one or more circuit faults and one or more input errors (invalid non-codeword inputs). For example, suppose due to data transmission error, a valid input $(a,a^*,b,b^*)=(0,1,0,1)$ is changed to an invalid input (0,0,0,1). A fault-free Morphic-AND gate will give an invalid output $(y,y^*)=(0,0)$ under this invalid input. However, if there is also a circuit fault, say line L1 s-a-1, then the output will be (1,0). Hence the invalid input is not detected.

Also notice that the Morphic-AND gates remain totally self-checking even if one of the inputs (say a) at the far front end (input pad, for instance) sticks, which results in input error at every gate input to which that particular input is applied. That is, this primary input error will be detected by these gates.

It is easy to implement these gates using MOS technology. Following the Caltech (Mead & Conway) design rules [23], an NMOS Morphic-AND gate is designed and the layout has an area of 60 λ by 70 λ. With a 2 micron/λ process, the area consumed is merely 120 microns by 140 microns (i.e. 60 Morphic-AND gates can be fitted into an area of 1 mm^2). A much simpler implementation of the Morphic-AND gate in CMOS is presented in section 2.1.3.

2.1.2 Totally Self-Checking Code Checkers and Comparators

As mentioned before, in the design of fault tolerant digital systems, the following techniques are often used:

(1) coding of communication data between cells of chips, chips of circuit boards, etc.;

(2) redundancy of cells and/or chips whose responses to identical inputs are compared and checked against each other for correctness.

In order to use these techniques , one will need TSC code checker(s) of some sort and TSC comparators. Morphic-AND gates are well suited for the implementation of these types of TSC circuits. It will be shown in the following that the design and implementation of these TSC checkers and comparators, using Morphic-AND gates as the basic building blocks, are indeed straight forward.

A commonly used code for communication data coding is the parity code. Thus as a simple example, we describe the design of a TSC n-bit even parity checker:

Let the even parity coded input A be $a_0a_1.....a_{n-1}a_n$ where a_0, a_1, ..., a_{n-1} are the information bits and a_n is the parity bit. The general circuit structure is as follows:

First the n bits are partitioned into two subsets $A1=\{a_0a_1.....a_m\}$ and $A2=\{a_{m+1}a_{m+2}.....a_{n-1}a_n\}$ where m=integer[n/2] for optimal circuit performance. The complement of $A=\bar{A}=\{\bar{a}_0\bar{a}_1.....\bar{a}_{n-1}\bar{a}_n\}$ is then generated. The circuit shown in figure 2.5 checks the parity of A and is totally self-checking.

Proof:

Let the number of 1s in the subsets A1 and A2 be N_{A1} and N_{A2} respectively. There are two cases to be considered:

Case 1 : N_{A1}/N_{A2} = even/even

In the circuit of figure 2.5, under valid inputs (i.e., $(a_i,\bar{a_i})=1_M$, $i=0,1,....,n$) and fault-free operation, we have $y_i=y_i^*$, $i=0,1,....,n$. If there is a single stuck-at fault in one of the gates (say the j^{th} gate of the upper gate array, G_j, $1\leq j \leq m$), then $y_j=y_j^*$. This error will propagate to the end of the gate array such that $y_m=y_m^*$. Now, since for every valid input pattern and in the even/even case and under fault-free operation, $y_m=y_n=0$ and $y_m^*=y_n^*=1$. Hence if the circuit is faulty as before, we will have either one of the following at the output, (y_{out},y_{out}^*):

Output	$y_m=y_{out}$	y_m^*	y_n	$y_n^*=y_{out}^*$
1	0	0	0	1
2	1	1	0	1

The first output $(y_{out},y_{out}^*)=(0,1)$ shows that the circuit is fault-secure. The second output (1,1) shows that the circuit is self-testing. Hence the circuit is totally self-checking.

Case 2 : N_{A1}/N_{A2} = odd/odd

In this case, for every valid input and under normal operation, $y_m=y_n=1$ and $y_m^*=y_n^*=0$. If there is a circuit fault (the same fault as in case 1, for instance), then we will have either one of the following outputs:

Output	$y_m=y_{out}$	y_m^*	y_n	$y_n^*=y_{out}^*$
1	0	0	1	0
2	1	1	1	0

The first output shows $(y_{out},y_{out}^*)=(0,0)$ shows that the circuit is self-testing while the second output (1,0) shows that the circuit is fault-secure. Hence in this case the circuit is also totally self-checking.

It remains to show that the circuit does check for even parity of the input vectors. As mentioned before, if N_{A1}/N_{A2}=even/even, then $y_m=y_n=0$ and $y_m^*=y_n^*=1$; if N_{A1}/N_{N2}=odd/odd, then $y_m=y_n=1$ and $y_m^*=y_n^*=0$. Hence the outputs are valid and the even parity is checked.If the input vector is in error such that N_{A1}/N_{A2}=odd/even or even/odd, then $y_m=y_n=y_m^*=y_n^*=1$ or 0. Any one of these two output vectors indicates the input error.

Now we start discussing the design of TSC comparator circuit. Given two sets of data : $A=[a_0a_1a_2.....a_n]$ and $B=[b_0b_1b_2.....b_n]$, we want to design a checking circuit which checks the equivalence of A and B and also checks for internal single stuck-at circuit fault. Figure 2.6 shows an n-bit TSC

comparator. The working principle of this circuit is very simple: If A=B, then $a_i=b_i$, i=0,1,2,.....,n. Thus $y_n=y_n^*$, indicating such equivalence. If A≠B ($a_j\neq b_j$, $0<j<n$, for instance), then $y_j=y_j^*$ which would ultimately lead to $y_n=y_n^*$, indicating A ≠ B.

The proof of the TSC properties of this comparator circuit is similar to that of the TSC parity checker circuit and thus will not be presented here.

2.1.3 Simple CMOS Implementation of Morphic-AND Gate

Using pass transistors only, the Morphic-AND gate can be implemented very easily with CMOS technology. Figure 2.7 shows a CMOS Morphic-AND gate implemented with two n-channel transistors and two p-channel transistors. This gate has the same truth table as that in figure 2.4. Its functional equations are also the same as those of the Morphic-AND gate presented in section 2.1.2 and thus it is also totally self-checking.

It may be interesting to note that the same circuit can be viewed as a Morphic-EXOR gate if we define the following:

Morphic logic 1 = (0,1) ;

Morphic logic 0 = (1,0) ;

and fault condition = (0,0) or (1,1).

With these definitions and considering only the last four cases (legal inputs) of the sixteen cases of the truth table in figure 2.4, we notice that the circuit performs the EXOR function.

It is worthwhile to note that totally self-checking parity checkers can be implemented by ordinary EXOR gates [34]. The circuit of this scheme is shown in figure 2.8. We can see that this scheme is simpler than ours in principle. The basic building cell of this scheme is also simpler than our NMOS Morphic-AND gate. However, if we use our CMOS Morphic-EXOR gate as the basic building cell for our scheme, we will have an even simpler circuit.

In conclusion of section 2.1, the classical approach to designing for testability can be implemented fairly easily and so can the checker circuits and comparator circuits, especially in the CMOS case. The circuits' string type structure is particularly suited for LSI/VLSI circuit implementation. The design phase of these circuits is almost error-free if the basic building gates are designed and laid out correctly.

Although it is clear that this approach can achieve a relatively large coverage of the most common LSI/VLSI fault types, it is practical only if the size of the CUT is relatively small. For large devices, this scheme may only be applied to a number of macrocells and/or microcells that are very crucial to the "Life" of the device (the CPU of an one-chip microcomputer, for instance) and consequently requiring large fault coverage. In section 3, we will further discuss the hardware redundancy problem of this approach and suggest a method to reduce the hardware overhead.

2.2 LEVEL SENSITIVE SCAN DESIGN (LSSD)

In addition to the aforementioned classical approach, there are several other schemes proposed over the last several years. One of them is called "Level Sensitive Scan Design", proposed by Eichelberger and Williams of IBM in 1977 [12]. Their scheme is particularly suited for the simplification of problems in testing, diagnostics, and field services for LSI/VLSI devices containing complex sequential subsystems.

The LSSD scheme defines the requirements for the design of easily testable sequential circuits as follows:

(1) Correct operation of the device should not be dependent on signal rise and fall times or on circuit or wire delay;

(2) The state of all internal storage elements (except memory arrays) can be loaded and observed through primary inputs and outputs of the device.

To meet the first requirement, the following design rules are proposed:

R1: All internal storage elements (system latches' are implemented in hazard-free polarity-hold (HFPH) latches. (See section 2.2.1 for design detail.)

R2: No latch may feed the data port of another latch which is clocked by the same clock. (See section 2.2.2.)

R3: All clocks must be controlled by primary inputs.

To meet the second requirement, the following design rules are proposed:

R4: Every system latch is implemented as part of a shift register latch (SRL). (See section 2.2.1.)

R5: All SRLs can be interconnected into one or more shift registers, each of which has an input, an output, and shift clocks available at the terminals of the device.

These rules are excerpts from the complete set of design rules which can be found in [12] or [26].

2.2.1 Hazard-Free Polarity-Hold (HFPH) Latch and Shift Register Latch (SRL)

To fulfill the requirement that correct operation of logic subsystems be independent of ac characteristics such as rise/fall times and circuit delays, all storage elements should be level-sensitive latches that contain no hazard or race condition. A level-sensitive latch should operate as follows:

(1) when clock C = 0, the latch cannot change state and

(2) when clock C = 1, the latch is set to the value of the input.

(See flow table in figure 2.9a.)

The corresponding excitation table, excitation equations, and logic implementation are shown in figure 2.9b. This design, however, is not hazard free due to the probable fact that AND gate G_2 has a larger delay than G_1 does, then a change of clock C from 0 to 1, resulting in a transition from implicant * to implicant **, may generate a logic 0 at G_1 before a logic 1 is generated at G_2. Consequently G_3 may give an "unclean" glitch output ($1 \rightarrow 0 \rightarrow 1$) instead of the expected $1 \rightarrow 1$ output. A design

that eliminates this problem is shown in figure 2.9c in which implicants * and ** are combined to form an additional prime implicant yD, thus giving a new set of excitation equations and requiring an extra AND gate G_4 in the logic implementation. This latch is HFPH and can be used as the basic internal storage element described in design rule R1. Now, according to design rule R4, we want to modify this HFPH latch to realize a level-sensitive, HFPH shift register latch. Figure 2.10 shows the symbolic representation, excitation equations, and logic implementation of such an SRL.

Two more clocks, A and B, one more input, I, and one more output, L_2 are added to the original design. For A=B=0, the SRL behaves exactly like an HFPH latch, and the SRL is said to be in the "Normal Operation" mode. In the "Shift (Scan-in/Scan-out)" mode of the SRL, C is set to 0, and A and B are used in the following manner:

First the input value at I is clocked into latch L_1 by A. Then the data in L_1 is clocked into Latch L_2 by B. Clearly, clocks A and B must be complementary clocks; otherwise the SRL would not operate promptly. This SRL design allows all latches in a subsystem to be connected serially. Therefore, the states of all latches can be loaded and observed by shifting in (scanning in) and shifting out (scanning out) the states serially. At most, four additional I/O terminals are required at each level of packaging. (See figure 2.11.)

2.2.2 Design Structure

Due to design rules R2 and R3, there are two basic structures of digital networks:

(1) Only a single mx1 parallel SRL net in the feedback loop (figure 2.12a) - In this case the structure design is very simple. Only one system clock C is needed, as shown in figure 2.12b. The only circuit delay problem that the designer has to consider is: In between subsequent clock C pulses, is there enough time allowed for the propagation of input signals through the combinational network (N) ? Therefore, the inter-clock pulse time of C should be made greater than or equal to the maximum delay time of the network N.

This structure is often called the "Single Latch Design" due to the fact that all system inputs to network N are taken from the L_1 latches. There is another possible basic structure of this type called the "Double Latch Design", as shown in figure 2.12c. Here all system inputs are taken from the L_2 latches. Two system clocks (C_1 and C_2) are needed. Clock C_2 is also used as the B shift. The delay requirement in this case is: the signals from the L_2 latches should be given enough time to propagate fully through network N during the time between the beginning of C_2 and the beginning of C_1.

(2) More than one mx1 parallel SRL net in the feedback loop (figure 2.13) - In this case at least two non-overlapping

system clocks (C_1 and C_2) are needed. Clock C_1 controls all SRLs in all SC_i(Sequential Net i), $1 \leq i \leq n$ and i=odd. Clock C_2 controls all SRLs in all SC_{i+1}. Again the only delay requirement is that the inter-clock pulse time between C_2 and C_1 must be greater than or equal to the maximum delay time of, among the n combinational networks, the one network that has the maximum delay. Note that some of the combinational networks may just be conducting wires such that string type connections of latches can also be modeled.

Clearly in all these structures, the scanning-in and scanning-out capabilities are built-in. Once the system clock(s) is disabled, the scan-in/scan-out processes can be performed fairly easily by activating the A-shift and the B-shift non-overlapping clocks in an alternating manner.

2.2.3 Fundamental Testing Techniques of LSSD-Structured Circuits and Systems

The following outlines some fundamental techniques by which circuits designed with LSSD structures can be tested:

(1) Testing of the shift register latches.

All SRLs can be tested fairly easily by shifting a short sequence of 1s and 0s in and out of the SRLs. If one or more 0s are present at the "scan-out" output, in the output sequence of 1s, then one or more latches are faulty (s-a-0), and the relative positions of the 0s in the output sequence of 1s indicate which latches are faulty.

(2) Testing of the combinational networks.

Any combinational network in a LSSD-structured circuit can be tested with the following procedures (we use the structure shown in figure 2.12b as an illustrating example):

P1. Initialize the state of the L_2 latches to some desired value through a "scan-in" process.

P2. Apply a desired test pattern to the primary inputs (P). Allow enough time for the input signals to propagate through N, generating stable output signals x_1, $x_2,\ldots,x_m$.

P3. Turn on clock C to store the output signals into the L_1 latches.

P4. Scan out the states of the L_1 latches and compare them with the expected responses.

For an LSSD-structured system, the following test processes are also possible:

(1) Dynamic diagnosis.

The same test pattern from the test generation program(s) can also be utilized as diagnostics. An individual subsystem or individual device can be checked periodically by some built-in diagnostic processor. Every time the checks are positive

(repairs have to be made for any negative checks), the t_0 time* of the system is re-established. Thus the reliability of the system is greatly increased. Note that a field engineer with some necessary maintenance tools may assume the role of the diagnostic processor which may not be available in some small systems.

(2) System bring-up.

A new or unused system may be made up of many defective components or may have many system design errors. In an unstructured design, error isolation is usually a difficult problem. In an LSSD-structured design, however, it is possible to resolve problems to within a combinational net of a device. Thus the system bring-up process is greatly simplified.

(3) AC testing and diagnosis.

During system testing, some of the ac characteristics of a system can be measured by varying the rate of the scan clock(s). The upper bound of the system operating clock rate may thus be determined.

During field diagnosis, this technique can also be used to determine whether a particular logic block is too slow.

---

*The t_0 time of a system is defined as the time at which the system is determined to be fully functional with compliance of all system specifications.

2.2.4 Advanced Testing Techniques of LSSD-Structured Circuits and Systems

The fundamenal testing techniques described in section 2.2.3 are indeed theoretically sound but not very intriguing in real circuit design and testing practices for the following reasons:

(1) The testing procedures (P1 - P4) are indeed very time consuming. For every test in which a primary input test vector $P=(p_1, p_2,..., p_n)$ is applied to the network N, a new feedback test vector $Y=(y_1, y_2,..., y_m)$ has to be scanned into the L_1 latches (re-initialization) such that all inputs to network N are known.

(2) The LSSD technique does not provide the circuit designer the freedom of interfacing LSSD-structured circuits with non-LSSD-structured circuits under testing mode. For instance, if the unstructured circuit to be fed by a structured circuit is mainly a sequential network, then in testing mode, the test vectors from the structured circuit to the unstructured circuit have to be in some prescribed order of sequence. These test vectors are produced by shifting them into the SRLs of the structured circuit and then sending them to the unstructured circuit. It is, however, not always possible to shift a prescribed sequence of test vectors into the SRLs. For example, in order to shift the test vectors (0,0) followed by (1,0) into the SRLs of the LSSD circuit A shown in figure 2.14 such that an input test sequence of {(0,0),(1,0)} is applied to the unstructured sequential

circuit B, the test vector (0,1) has to be generated between the two desired vectors, making the input test sequence {(0,0),(0,1),(1,0)} which is no longer a valid test of the unstructured sequential circuit.

(3) When the LSSD circuit is in the normal operating mode, it is desirable that some of the HFPH latches (especially the L_2 latches) be merged into the functional core and utilized for functional operations (use the L_2 latches as data registers, for example). With the aforementioned design structure, this "unusual" utilization of existing hardware is just too difficult to accomplish.

In the following we will show some other LSSD testing techniques and design structures that, for a small amount of extra hardware cost, not only simplify the testing processes but also enhance circuit design feasibility.

A simplified way of LSSD circuit testing

A more effective and efficient way to test a LSSD circuit would be to test the sequential and the combinational networks separately and independently. Independent testing of the sequential network (the SRLs) has been described in section 2.2.3 to be a trivial task. It is not so for the testing of the combinational block. In order to test the combinational block independently, the feedback path must be broken and the feedback test vector $Y=(y_1, y_2,..., y_m)$ must be externally controllable such that it can be considered as part of the primary inputs to the combinational block. It seems that procedures P1-P4 do exactly

this, but the fact is that the number of times that the L_1 latches have to be re-initialized must be drastically reduced to make the scheme practical.

Figure 2.15 shows a LSSD design structure which differs from that shown in Figure 2.12b only in that every basic storage element is implemented in a so-called "Stable Shift Register Latch (SSRL)". Figure 2.16 shows the logic implementation of an SSRL, a modified version of the SRL. There is an additional input port Q for the L_1 latch with shift clock C_1 and an additional latch L_3 with shift clock C_2. With this structure the combinational block N can be tested as follows:

Since in testing of combinational circuits, the order in which test vectors are applied to the circuits can be arbitrary, we can reorder the computed test set in which every vector is of the form $(p_1, p_2,..., p_n, y_1, y_2,..., y_m)$ to form subsets (ST) of test vectors such that

(1) in each subset, the feedback test vectors $(y_1, y_2,..., y_m)$ are constant vectors,

(2) no two subsets have identical constant feedback test vectors,

(3) the constant feedback test vectors in the subsets are in an ascending order.

For example: Assume the combinational block N has two primary inputs p_1, p_2 and two feedback input y_1, y_2, and the computed test pattern is as follows:

p_1	p_2	y_1	y_2
0	0	0	1
0	1	0	0
0	0	1	0
1	1	1	0
1	0	0	1
0	0	1	1
1	1	1	1
0	1	1	0
1	0	0	0
1	1	0	1
0	1	1	1

Following the rules presented above, the test set is reordered as follows:

p_1	p_2	y_1	y_2	
0	1	0	0	constant feedback
1	0	0	0	test vector
0	0	0	1	constant feedback
1	0	0	1	test vector
1	1	0	1	
0	0	1	0	constant feedback
0	1	1	0	test vector
1	1	1	0	
0	0	1	1	constant feedback
0	1	1	1	test vector
1	1	1	1	

Now we can apply the recorded test set to network N as follows: Assume there are k subsets and q primary input test vectors in each subset.

(1) Let i=j=1.

(2) Scan in the constant feedback test vector (y_1, y_2,..., y_m) of ST_i into the the L_1 and L_3 latches by applying the A, B, and C_2 clock signals in an appropriate sequence.

(3) Apply P_j of ST_i to network N.

(4) Allow enough time for the input test signals to propagate through N, generating a stable output vector (Z, x_1, x_2,..., x_m).

(5) Scan out x_1, x_2,..., x_m, and compare them with the expected response.

If error is detected, stop.

If no discrepancy is detected and if $j \leq q$, increment j and go to (7), otherwise continue.

(6) If $i \leq k$, increment i and go to (2), otherwise stop.

(7) Turn on C_1 to load content of L_3 latches into L_1 latches and go to (3).

Clearly, with the modified structure, the number of times that the L_1 latches have to be re-loaded is reduced from kxq to k. This structured design scheme indeed converts a given sequential network (under test mode) into a combinational network. Therefore, the problem of testing sequential networks in an LSI/VLSI device would be greatly simplified if such a DFT technique is employed throughout the circuit design phase.

2.2.4.1 LSSD to NON-LSSD Interface. Using the modified LSSD design structure shown in figure 2.15, the problem of getting the correct test sequence to test the Non-LSSD network shown in figure 2.14 can easily be solved. For example, as previously stated, the sequence y_1y_2=[00,01] cannot be shifted into the SRLs of the structure shown in figure 2.12c. However, in figure 2.15, y_1y_2=00 can be shifted in the L_3 latches first. Then the sequence (01,10) can be shifted into the L_1 latches. By the time the pattern 10 is in the L_1 latches, it can be loaded into the L_3 latches by applying a C_2 clock pulse. Therefore, the non-LSSD logic does not see the intermediate 01 pattern. Figure 2.17 shows the original SSRL design proposed in the paper by DasGupta [27]. This SSRL, which has a lower gate count than the one shown in figure 2.16, cannot be used for the testing scheme described in pages 41-44 because direct feedback from the L_3 latches to the L_1 latches of the original SSRL would interfere with the data flow between the combinational net and the latches or the scan-in/scan-out data flow. Consequently, an additional data port (Q) is required.

2.2.4.2 Utilization of Existing Latches in LSSD Logic. Figure 2.18 shows an example of use of SSRLs in LSSD logic. 2n SRLs would be required to store the addend and the augend if the circuit is designed with SRLs. For the same adder circuit designed with SSRLs, however, only n SSRLs are required, resulting in a saving of 12n logic gates (there are 10 gates per SRL and 16 gates per SSRL).

2.2.4.3 LSSD Parity Checking. The scan-in/scan-out capabilities of an LSSD structure also allow for easy testing of the parity checkers of circuit data registers. For example, if the addend register in figure 2.18 is parity coded (e.g., a_0 is the parity bit) and its output is checked by a parity checker as shown in figure 2.19, then the correct operation of the parity checker can be tested by loading the register with bad parity through a scan-in process and then by observing the output of the parity checker. In an unstructured yet complex logic network, the loading of invalid parity into the SRLs may not be possible. Even if it is possible, a rather long homing sequence is required.

2.2.4.4 On-line Dynamic Scan. The same structure design shown in figure 2.15 can also be used for on-line dynamic scan. During normal operation and in every system cycle, the machine state is stored in the L_3 latches. When an error is detected, normal operation is halted and the L_3 latches contain the last known machine state prior to the error occurrence. This machine state

can now be reloaded into the L_1 and L_2 latches, and the system can be re-started and stepped through one cycle at a time until the error condition is re-created and diagnosed. This diagnosis process can be conducted by a diagnostic processor as mentioned in subsection 2.13.

2.2.4.5 On-line Error Detection in Memory. The SSRLs can also be used to detect stuck-at faults in the system memories. Figure 2.20 shows the block diagram of such a scheme. A word X is read out and stored in the L_2 latches. X from the L_2 latches is reloaded into the same memory locations. Then it is read out again and stored in the L_3 latches. X is now available at the L_3 outputs of the L_3 latches. It is again written into the same memory locations to restore the data. Now the contents of the L_2 and the L_3 latches are compared bit-for-bit. If there is any bit in the memory that has stuck-at faults, then the corresponding bits in the L_2 and the L_3 latches will be identical. The TSC comparator (see subsection 2.1) will detect this error and flag an error signal.

LSSD structured design using SSRLs surely introduces considerable hardware overhead at the gate level. However, the enhancement in circuit testability, system serviceability, and design feasibility provided by this scheme would indeed justify the hardware overhead cost. In section 3, we will discuss this aspect in further detail.

2.3 MODULARIZED DECOMPOSITION VIA MULTIPLEXED ROUTING

Noting the facts that test generation and fault simulation are perhaps the most costly and time consuming processes in LSI/VLSI device testing and that the stuck-at fault model may no longer be adequate for LSI/VLSI circuit fault characterization, S. Bozorgui-Nesbbat and E.J. McCluskey of Stanford University [16] proposed in 1980 a DFT scheme which they claimed would allow for exhaustive testing of the devices and thus would virtually eliminate test generation and fault modeling. The basic principle of their scheme is as follows:

To a given network G to be designed for testability, additional routing logic (bus multiplexers) are added such that G can be decomposed into modules G_1, G_2,..., G_n with the following properties:

(1) Each module G_i, $1 \leq i \leq n$, has few enough inputs that exhaustive testing of the module can be practically accomplished.

(2) All inputs and outputs of each module can be directly connected (routed) to the primary inputs and outputs of the network G respectively.

This scheme alone may be effective only if in every module, there is, if not none, a simple sequential net that is buried deep in the module. If the opposite is true, then the problem of fault diagnosis remains unresolved, even though test generation is still not required. In this case, the LSSD structured design technique may also be employed to ease the fault diagnosis problem.

2.3.1 Basic Circuit Decomposition Scheme

Given a network G with input bus X and output bus Y, it is always possible to decompose it into two modules G_1 and G_2 with disjoint sets of inputs (X_1 and X_2) and outputs (Y_1 and Y_2) and two internal linking buses (L_{12} and L_{21}) called the supplementary inputs and outputs, as shown in figure 2.21. In figure 2.22 we give a simple example of circuit decomposition. Notice that even though G_1 and G_2 have a common input (X_4), the decomposition can still be carried out by making the primary input X_4 to G_2 a supplementary input. Similarly the primary output f_1 of G_1 can also be made a supplementary output that feeds G_2. One may consider this decomposition technique as the input/output bus decomposition as against the usual functional decomposition of digital circuits. Nevertheless, functional decomposition, if it can be applied to the given network G, can also be used for this scheme. In order to test G_1 and G_2 independently, we must be able to isolate the module not under test from the module under test. Figure 2.23a shows the block diagram of an implementation of such a scheme. When G_1 is to be tested, for instance, G_2 is to be isolated. Figure 2.23b shows the selected data paths under such a test mode. Direct control of the supplementary inputs (SI_1) of G_1 is achieved by connecting this bus directly to the primary input bus X_2 via MUX 2. Direct observation of the supplementary outputs (SO_1) of G_1 is achieved by connecting it to the primary output bus f_2 via MUX 1 and MUX 4.

2.3.2 Generalized Routing Scheme

A generalized routing scheme can be deduced by careful examination of the 2-module routing scheme described in the last subsection. In general the following remarks are true:

(1) Suppose the network G is decomposed into n modules. Each module G_k ($k \in \{1,2,...,n\}$) receives n input buses (one primary input bus X_k and n-1 supplementary input buses L_{jk} ($j \in \{1,2,...,k-1,k+1,...,n\}$) from the other n-1 modules) and produces n output buses (Y_k and L_{kj} for all k except k=j) for the primary output and the other n-1 modules respectively (see figure 2.24).

(2) For each module G_k, there are n-1 linking multiplexers and one output multiplexer. (Figure 2.24)

(3) The width of any supplementary input or output link of any module must be less than or equal to that of the smallest primary input or output link.

Figure 2.25 depicts the block diagram of the implementation of a 3-module routing design. Nine multiplexers are used to route the external (primary) and the internal (supplementary) input and output buses. In normal operation, the linking multiplexers select all the supplementary output buses of all three modules, and the output multiplexers select the primary output buses of all three modules. In test mode (G_2 is under test, for example), the linking multiplexers and output multiplexers select the data paths drawn in thick lines in figure 2.25.

If we group all the multiplexers together, we will have created a circuit block which can be called the Liaison Network. Figure 2.26 shows a conceptual picture of the generalized routing model. The number of multiplexers in the liaison net is equal to n^2. The issue of additional I/O pins will be addressed in section 3. Figure 2.27 gives an implementation of a 1-out-of-4 bus multiplexer using pass transistors. A 1-out-of-n bus multiplexer has the same structure and can be implemented accordingly.

2.3.3 Realiability and Testability of the Liaison Network

Since the multiplexers are mainly pass-transistor switching networks, they are very reliable. However, faults do occur in this type of circuit. A common one is the bus stuck-at fault (i.e., one of the input buses is always selected (connected) to the output bus, independent of the selection address). This type of fault can be easily detected since, for any one multiplexer, at least one of the input buses is externally controllable and the output bus is externally observable; it can be tested by routing the input buses one by one to the output bus and observing them one by one. If these output vectors are all identical, we can almost conclude that the multiplexer has a bus stuck-at fault. The exception is that by some very small chance, all input buses may have the same data vector. This doubt can be eliminated by applying a different test vector to the externally controllable

input bus of the multiplexer and repeating the above test procedure once more. If the same phenomenon occurs, then we can certify our previous conclusion. Further testing may be required to identify the faulty input bus. Note that if there is more than one faulty multiplexer at any given time, the fault detection process may be difficult and time-consuming.

This multiplexed routing scheme indeed enhances circuit testability. A large network may be partitioned into smaller modules, each of which may be exhaustively tested in a reasonable amount of time. Hence the processes of fault modeling and test generation may be abandoned (see discussions in subsection 3.3 concerning the practicality of this supposition). On the other hand, if the modules contain relatively complex buried sequential nets, then fault diagnosis may be very difficult unless some other DFT technique(s) is also employed. Furthermore, as it is common in most DFT schemes, this scheme decreases the processing speed of the device due to the time delay of the multiplexers. Therefore, none of the aforementioned DFT schemes is the answer to effective circuit design for testability. In section 3 we will try to present and compare the strong and weak points of each of these schemes. Then in section 4, we will try to merge these techniques along with some other DFT techniques and test generation techniques to form a general approach to LSI/VLSI circuit design for testability.

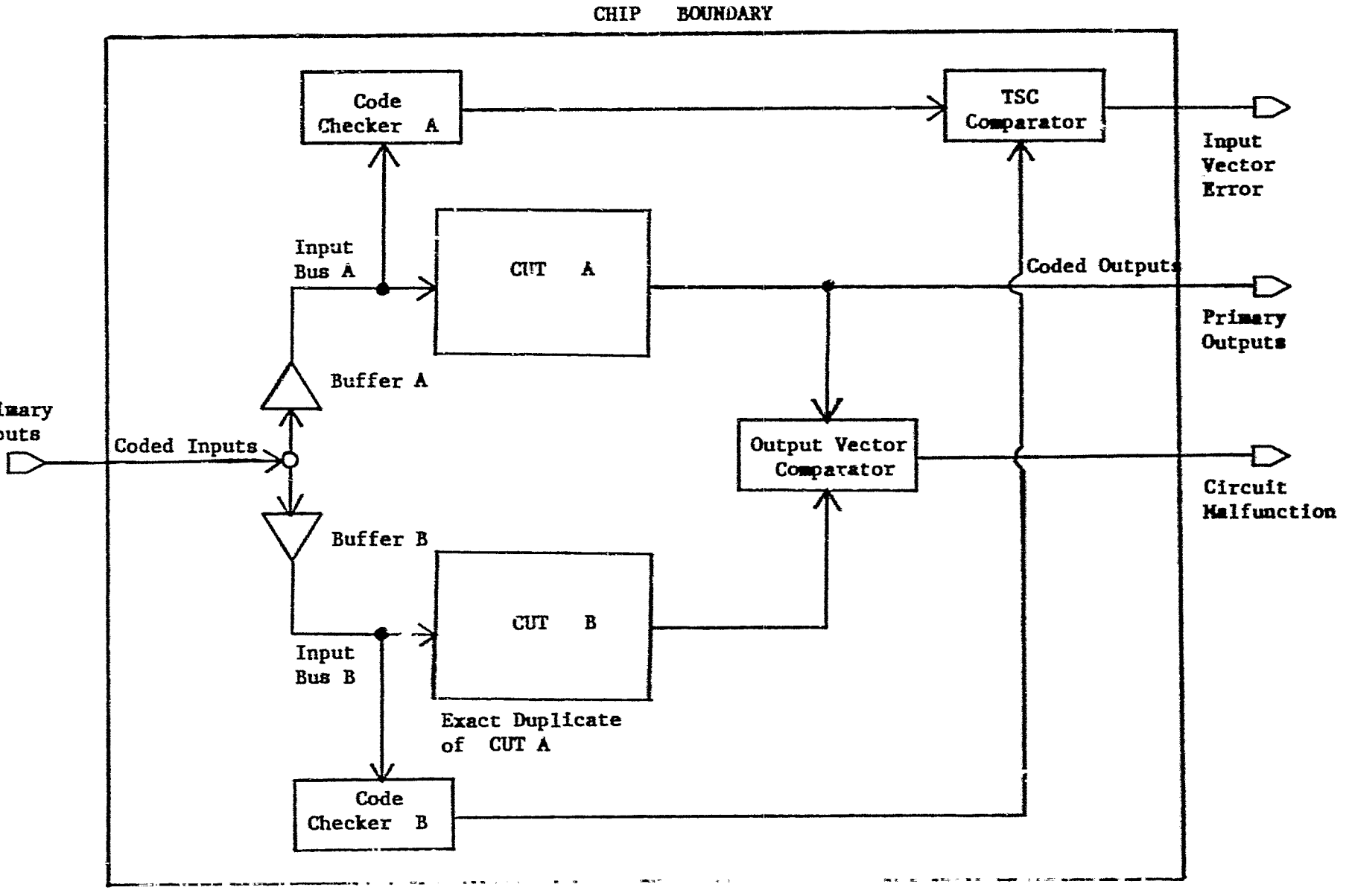


Figure 2.1 Block Diagram of TSC Circuit

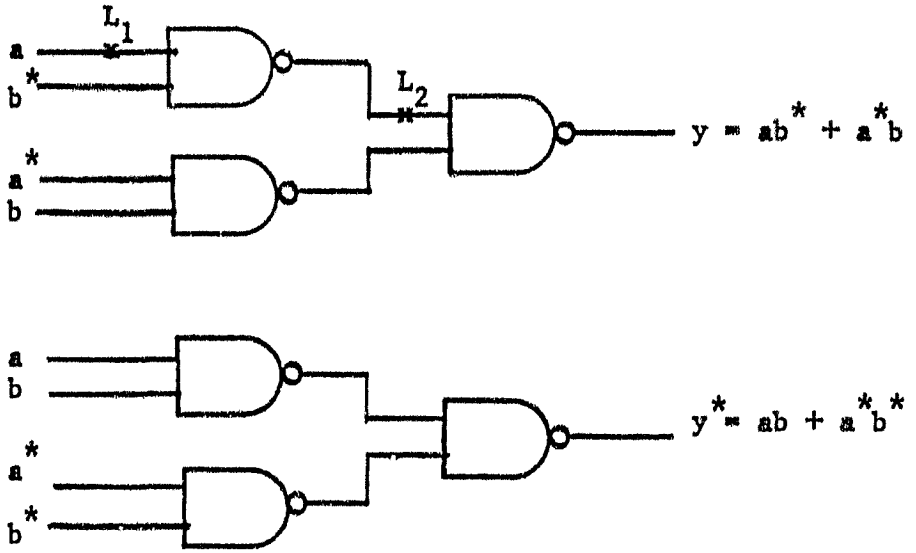


Figure 2.2 2-level NAND-NAND Logic Implementation of Morphic-AND Function

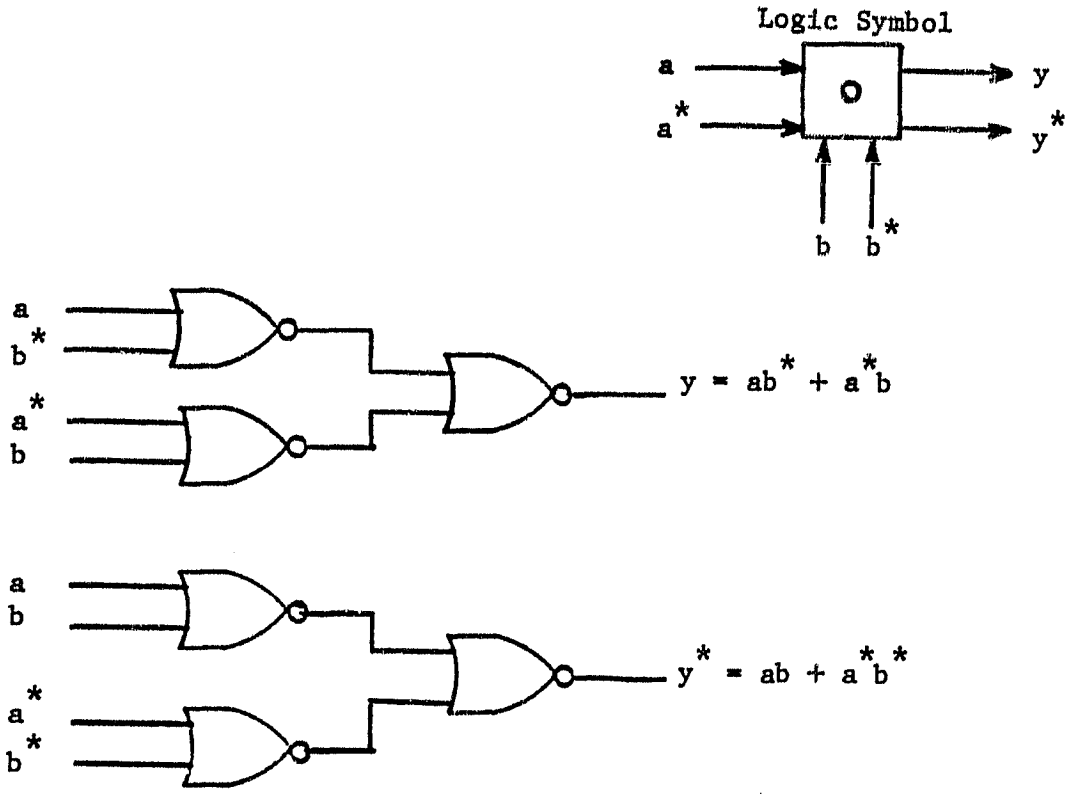


Figure 2.3 2-level NOR-NOR Logic Implementation of Morphic-AND Function

A	B	Y
0_M	0_M	0_M
0_M	1_M	0_M
1_M	0_M	0_M
1_M	1_M	1_M

	a	a^*	b	b^*	y	y^*	
Illegal Inputs (Non-codewords)	0	0	0	0	0	0	Illegal Outputs
	0	0	1	1	0	0	
	1	1	0	0	0	0	
	1	1	1	1	1	1	
	0	0	0	1	0	0	
	1	1	0	1	1	1	
	0	0	1	0	0	0	
	1	1	1	0	1	1	
	0	1	0	0	0	0	
	0	1	1	1	1	1	
	1	0	0	0	0	0	
	1	0	1	1	1	1	
Legal Inputs (1-out-of-2 Codewords)	0	1	0	1	0	1	Legal Outputs
	0	1	1	0	1	0	
	1	0	0	1	1	0	
	1	0	1	0	0	1	

Figure 2.4 Morphic-AND Function Truth Table

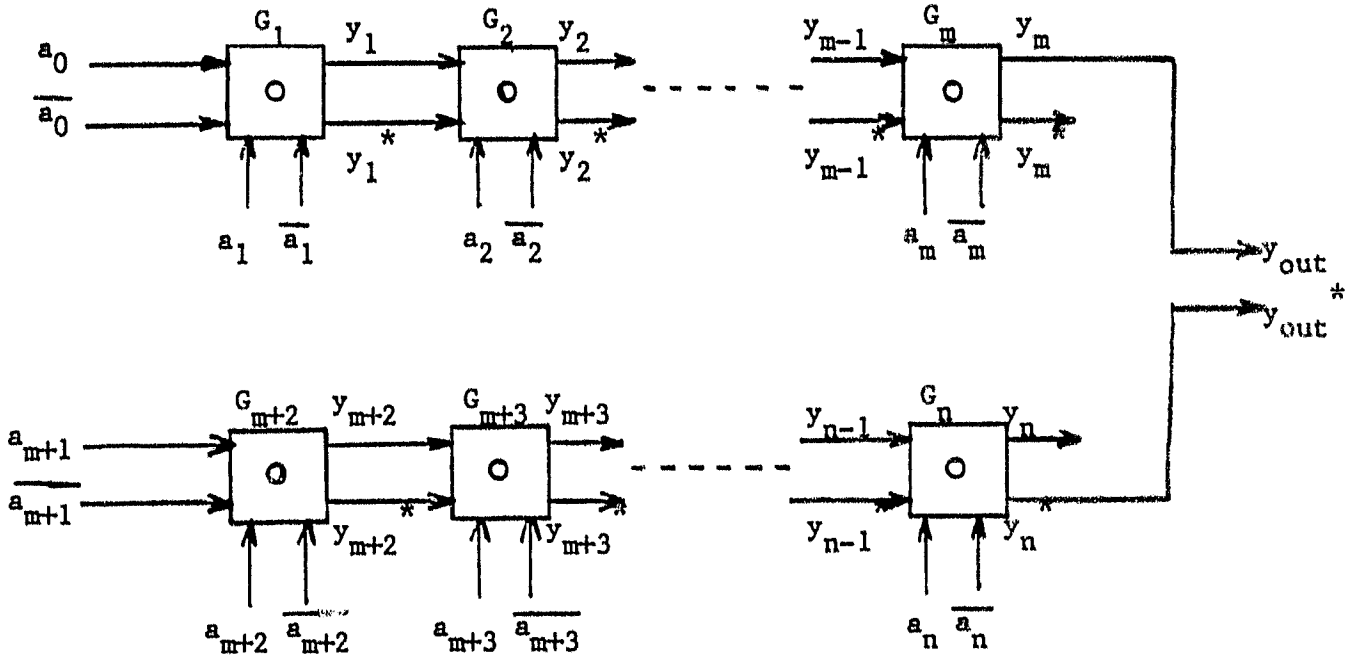


Figure 2.5 n-bit TSC Even Parity Checker

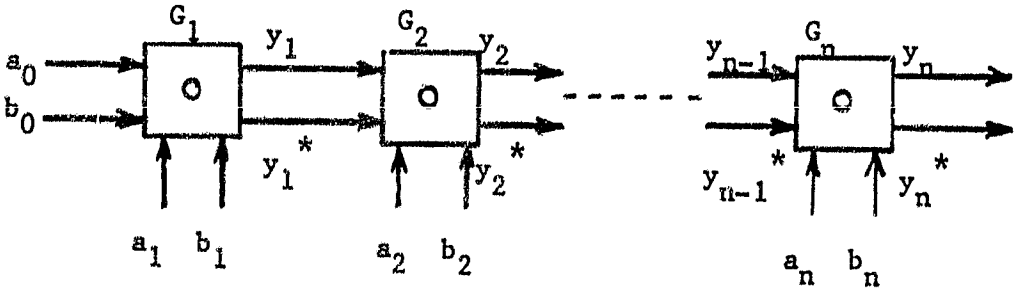


Figure 2-6 n-bit TSC Comparator

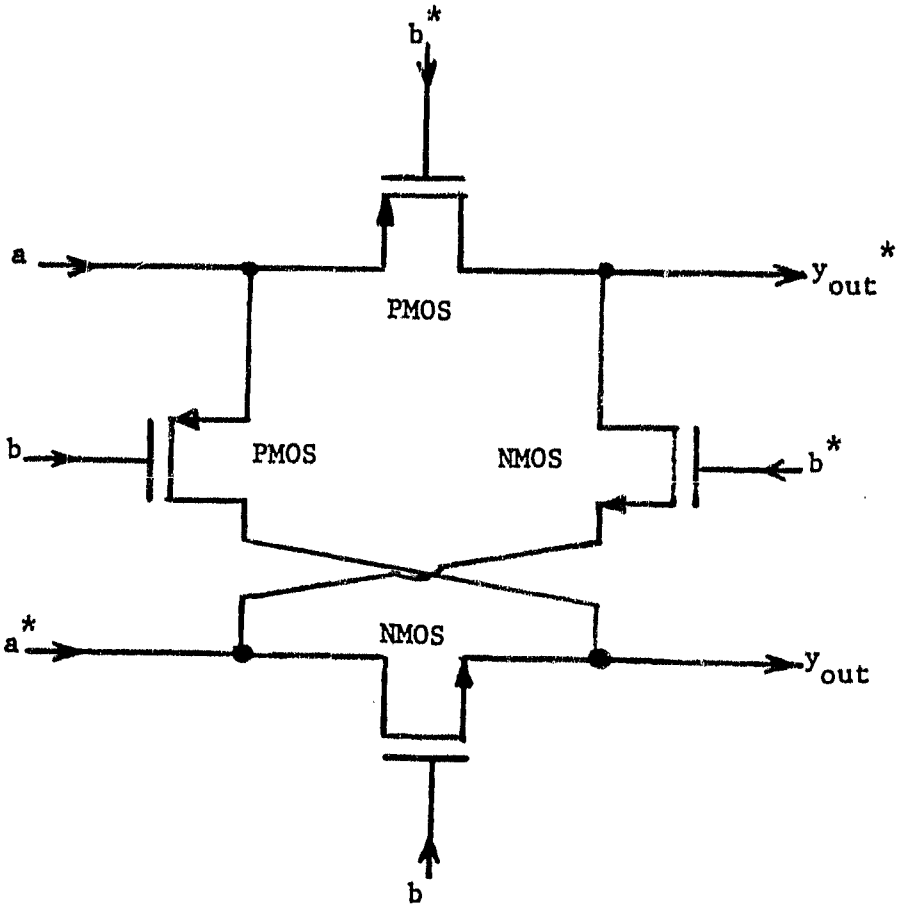


Figure 2.7 CMOS Morphic-AND Gate

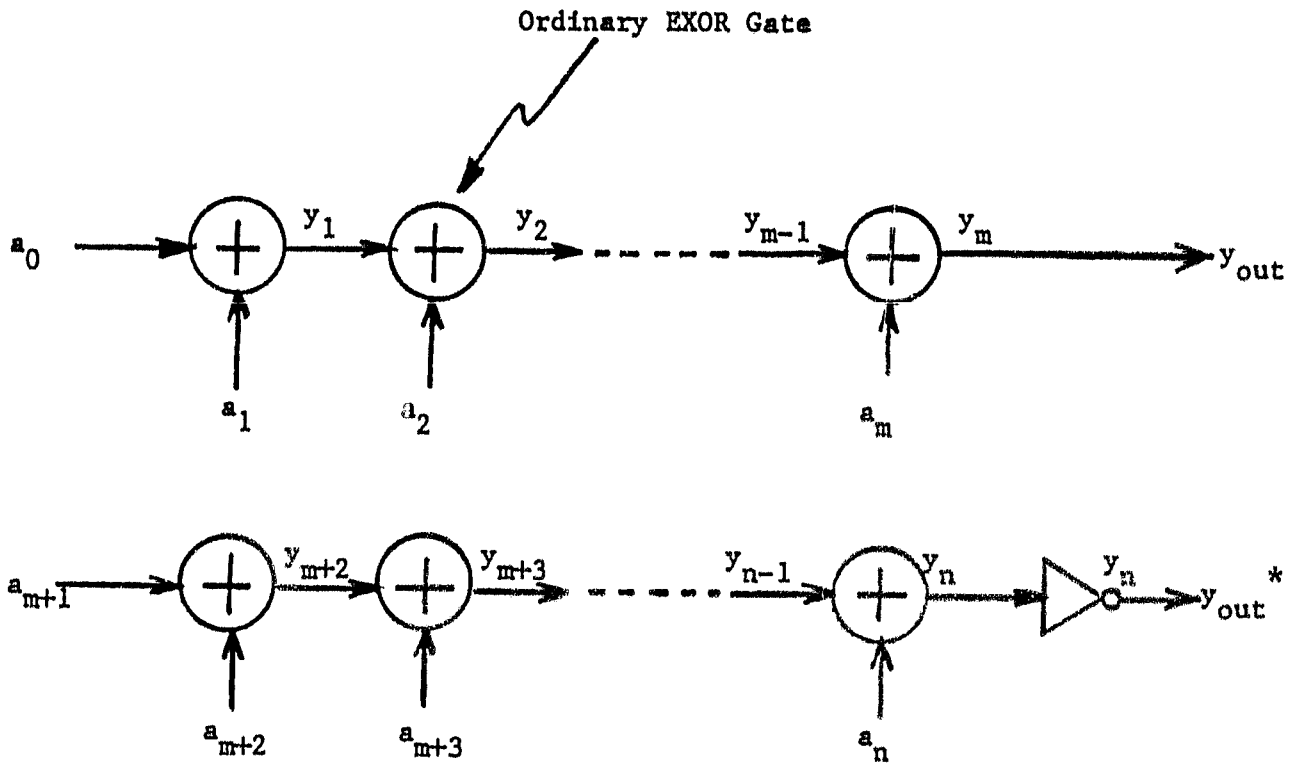
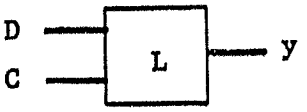


Figure 2.8 n-bit TSC Parity Checker



Logic Symbol

S \ CD	00	01	11	10	L
A	A	A	B	A	0
B	B	B	B	A	1

Figure 2.9a Flow Table of Level-Sensitive Polarity-Hold Latch

y \ CD	00	01	11	10
0	0	0	1	0
1	1	1*	1**	0

$Y = CD + y\overline{C}$

where y = present state

Y = next state

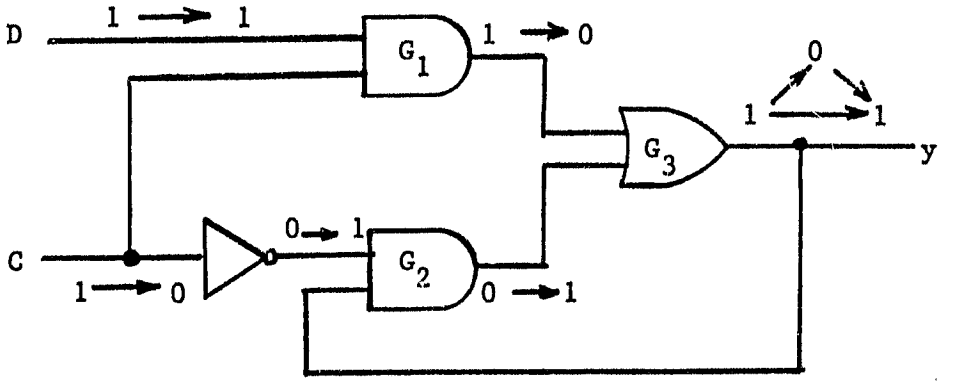
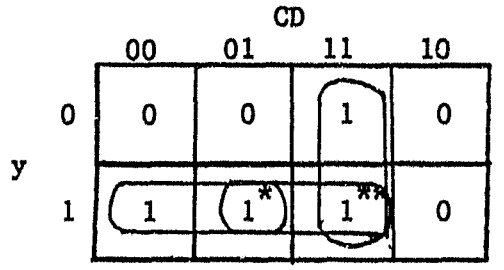


Figure 2.9b Excitation Table, Excitation Equation, and Logic Implementation of a Level-Sensitive Latch



$$Y = CD + y\overline{C} + yD$$

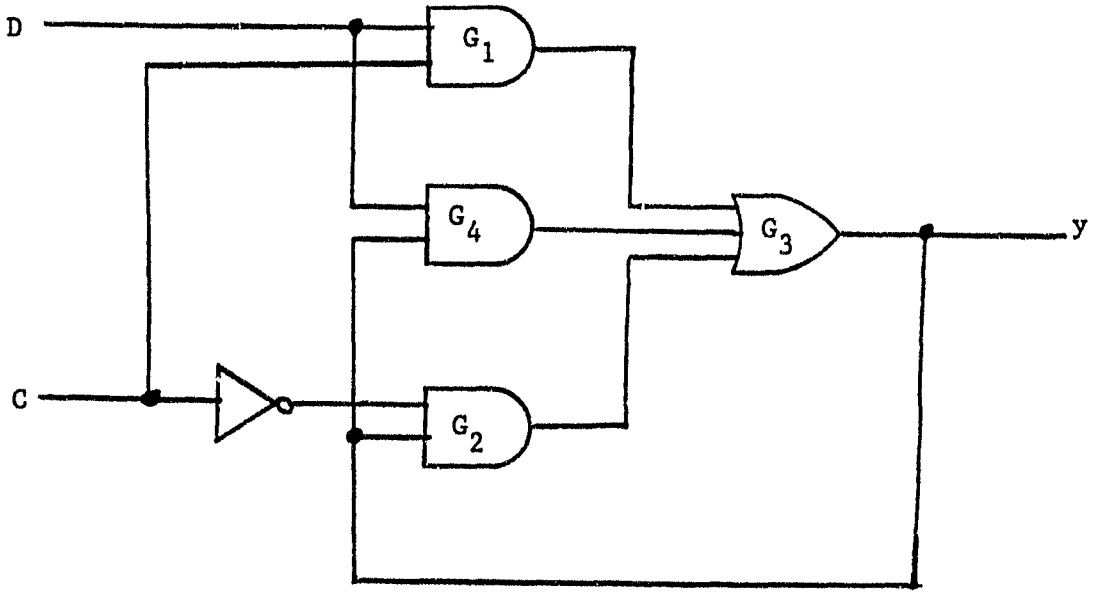
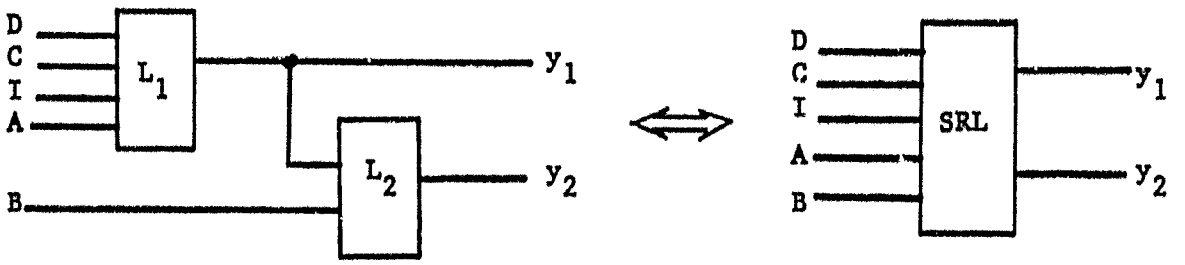


Figure 2.9c Excitation Table, Excitation Equation, and Logic Implementation of an HFPH Latch



$$Y_1 = CD + AI + y_1(\overline{C} + D)(\overline{A} + I)$$

$$Y_2 = \overline{\overline{y_1}B + \overline{y_1}y_2B}$$

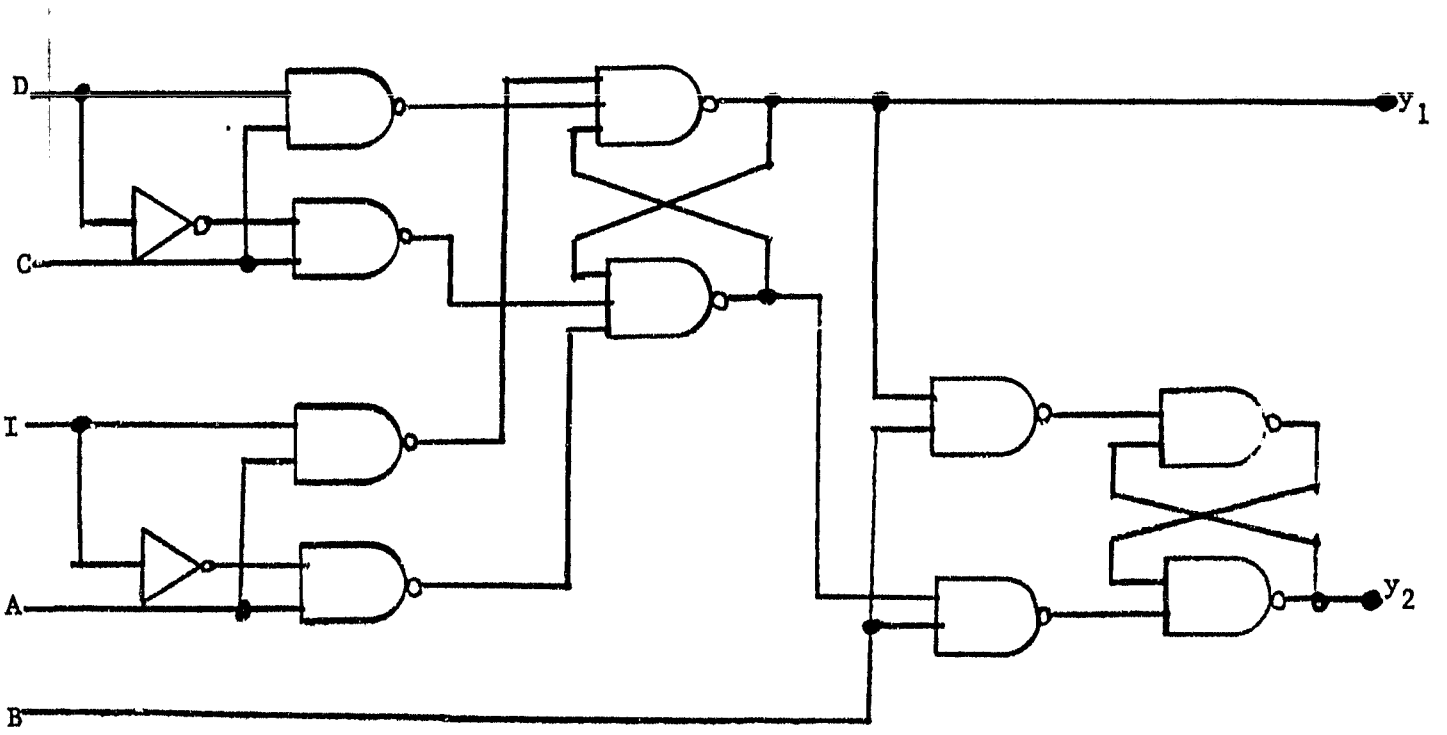


Figure 2.10 Symbolic Representation, Excitation Equation, and Logic Implementation of a Shift Register Latch

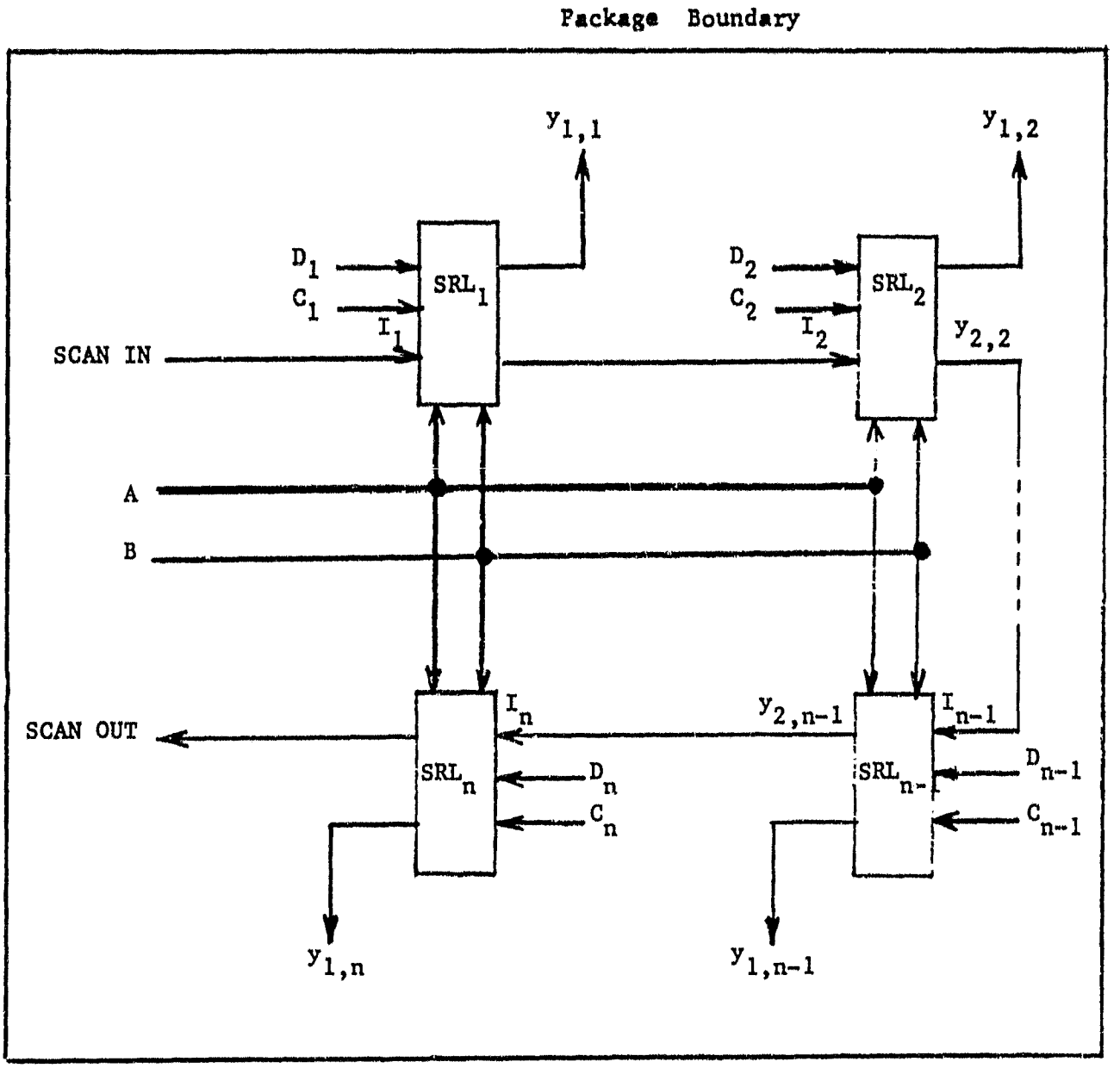


Figure 2.11 Interconnection of SRLs for Scan-in/Scan-out

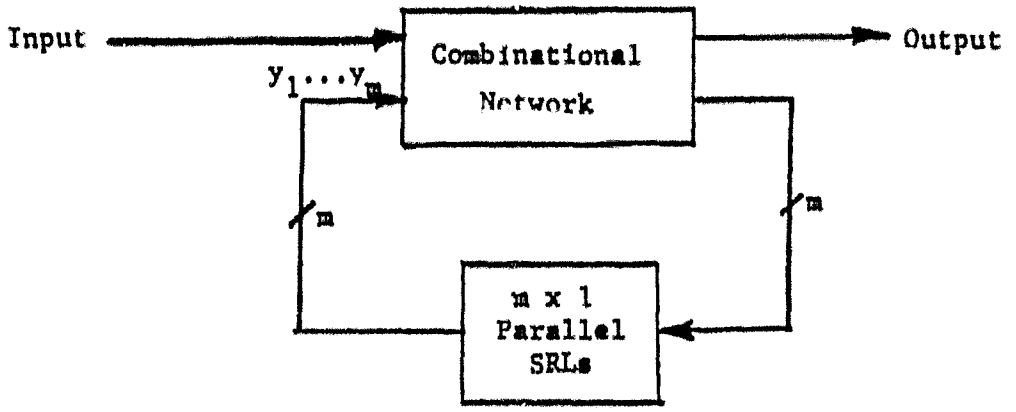


Figure 2.12a Basic Sequential Network Structure: Type 1

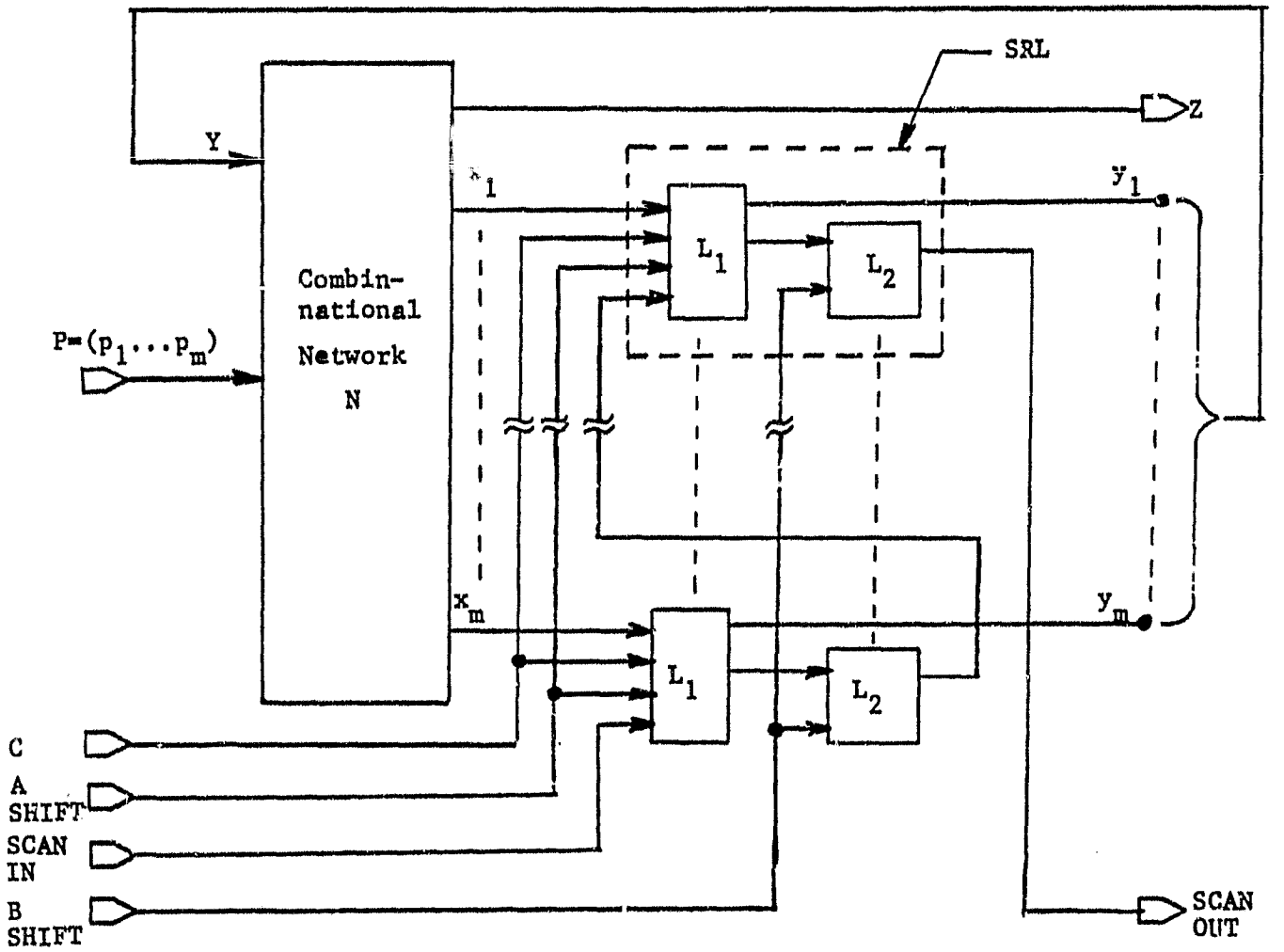


Figure 2.12b Block Diagram of LSSD Single Latch Design Structure

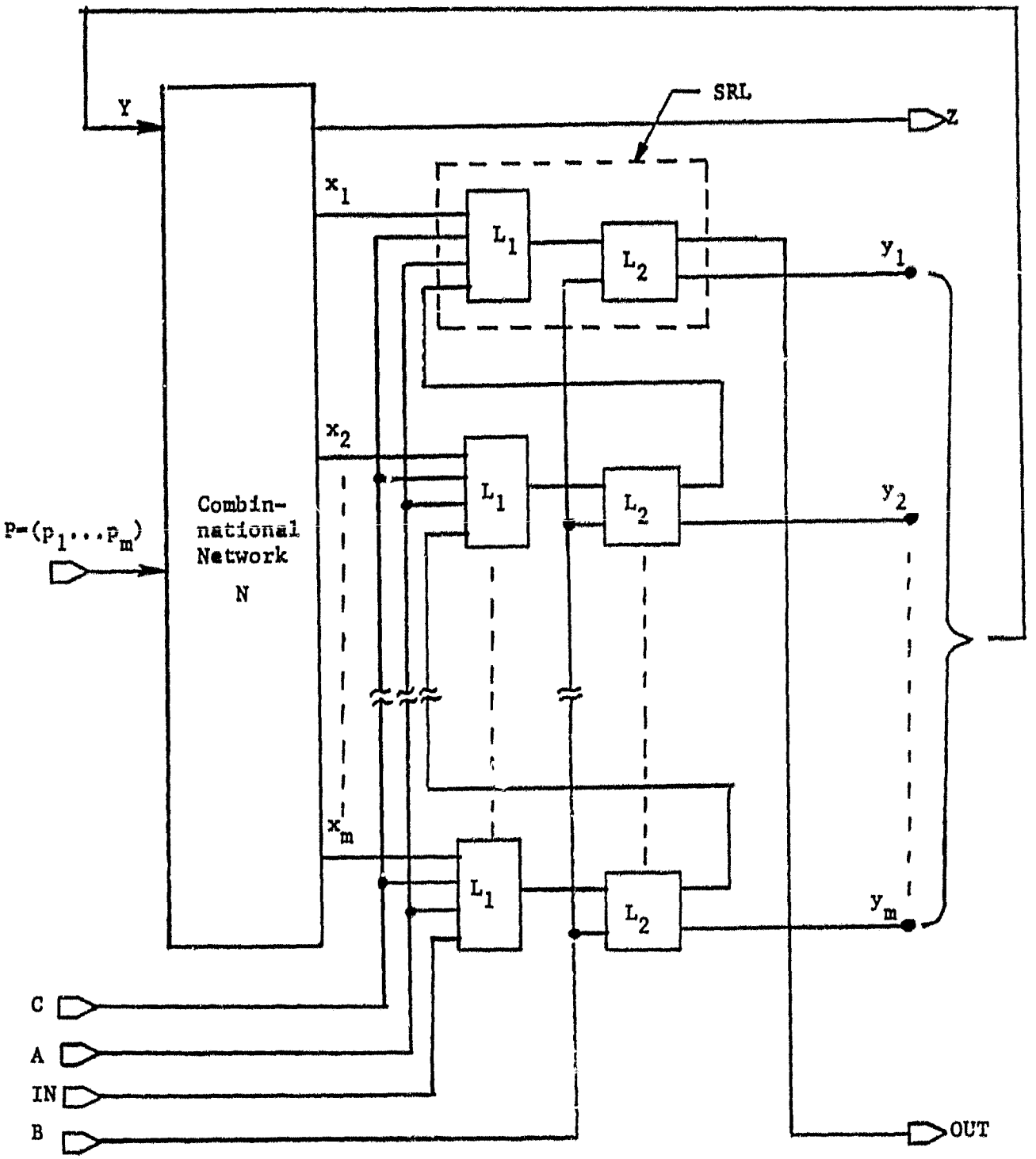


Figure 2.12c Block Diagram of LSSD Double Latch Design Structure

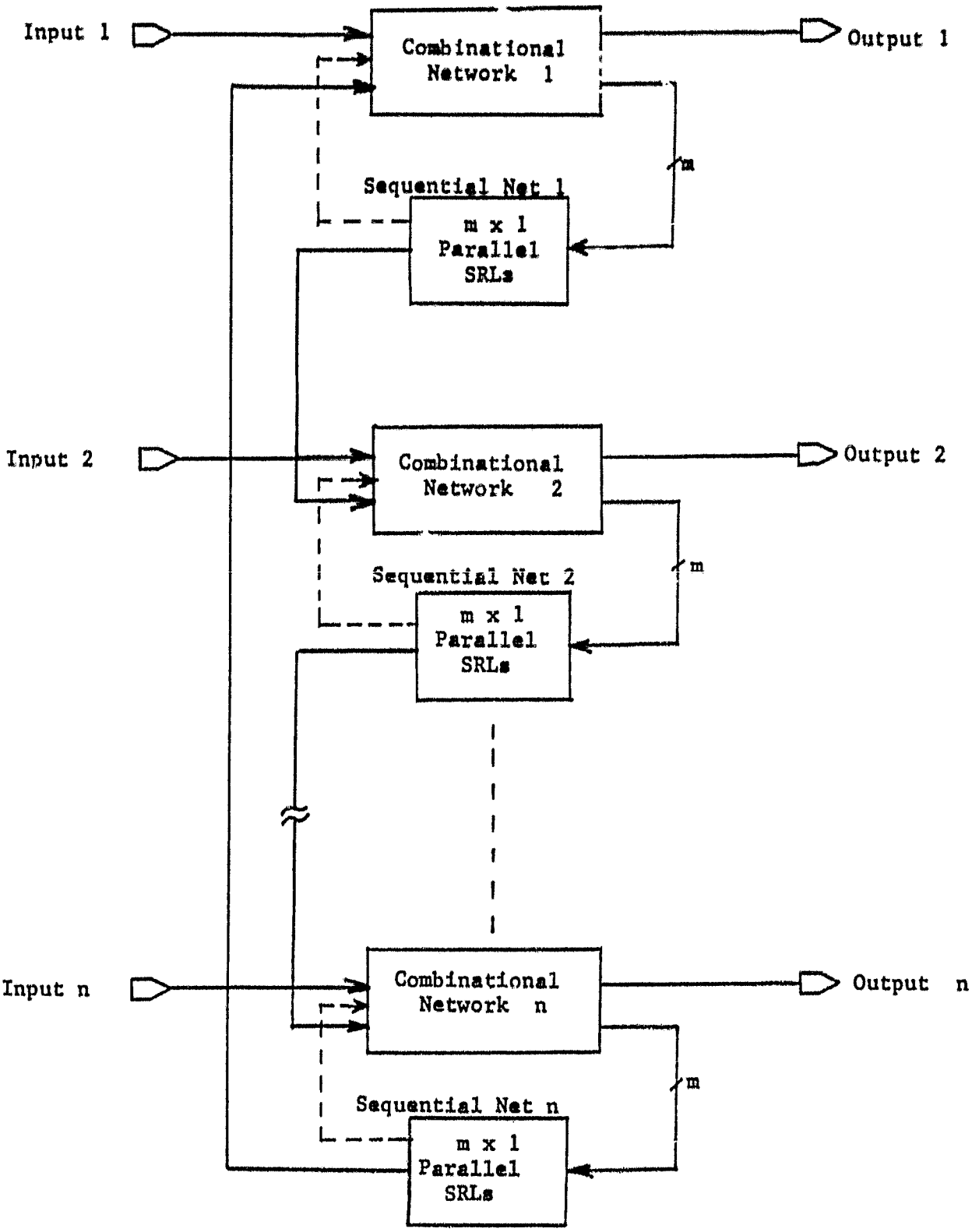


Figure 2.13 Basic Sequential Network Structure: Type 2

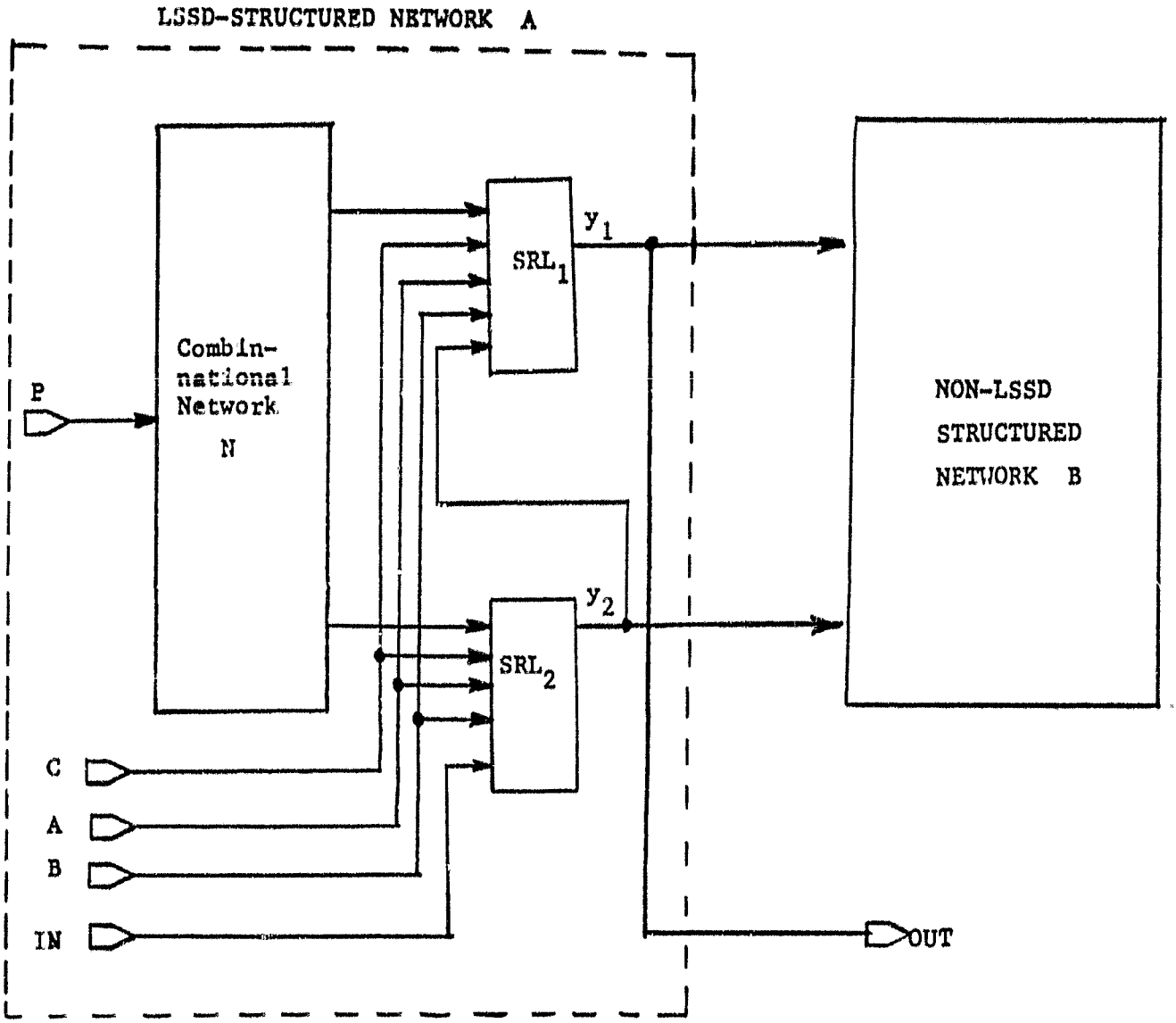


Figure 2.14 Block Diagram of LSSD to Non-LSSD Network Interface

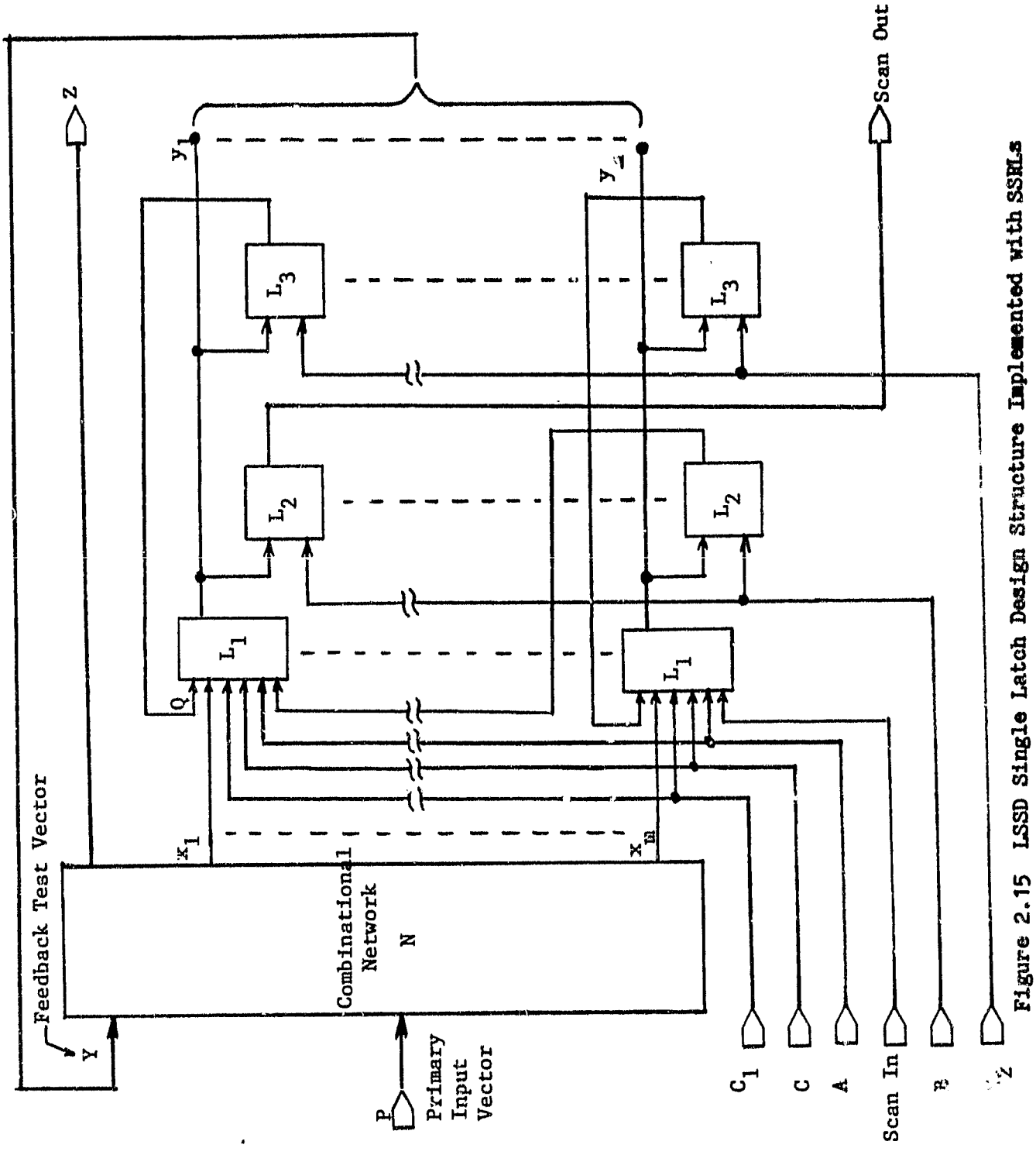


Figure 2.15 LSSD Single Latch Design Structure Implemented with SSRLs

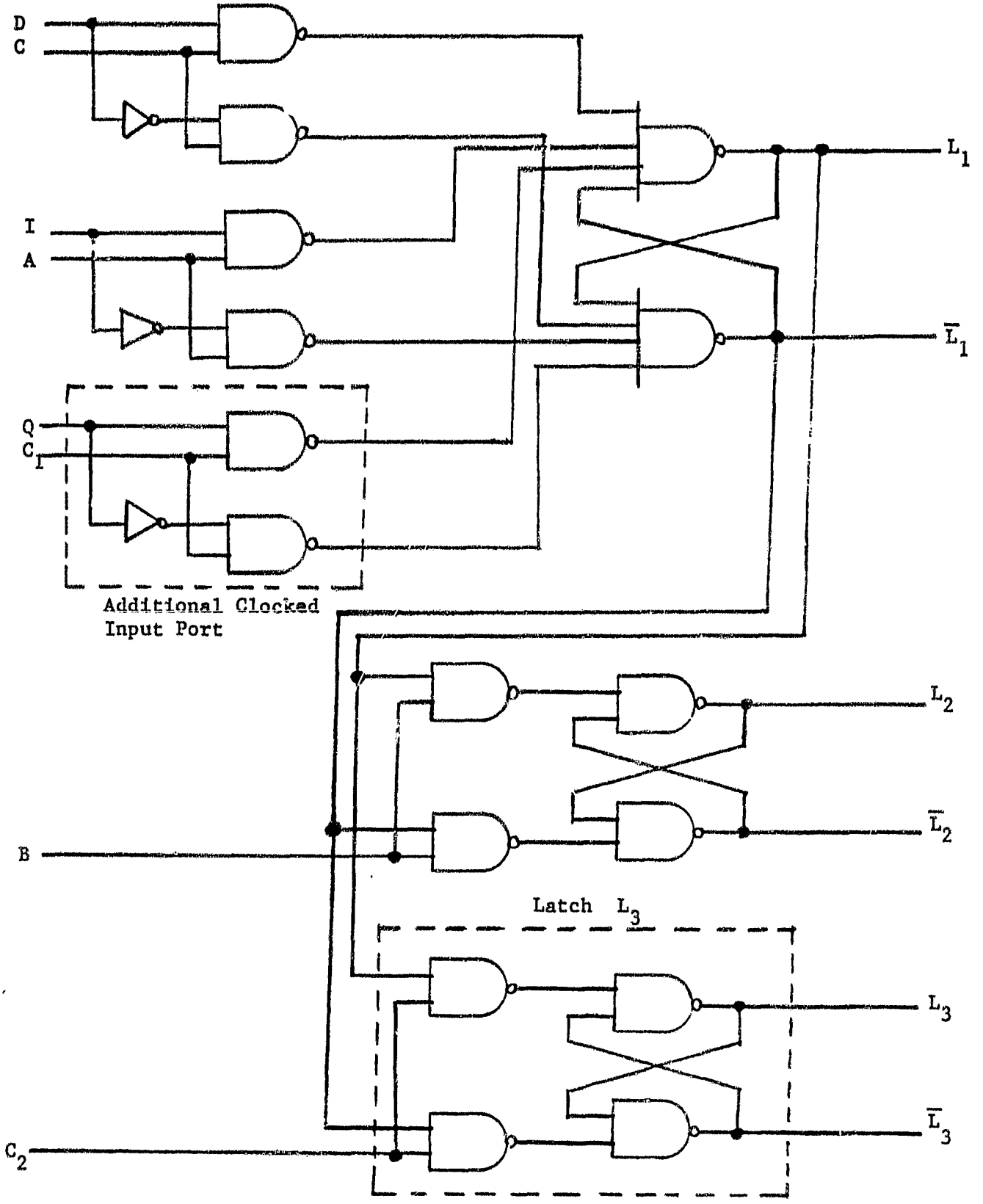


Figure 2.16 Logic Implementation of a Stable Shift Register Latch

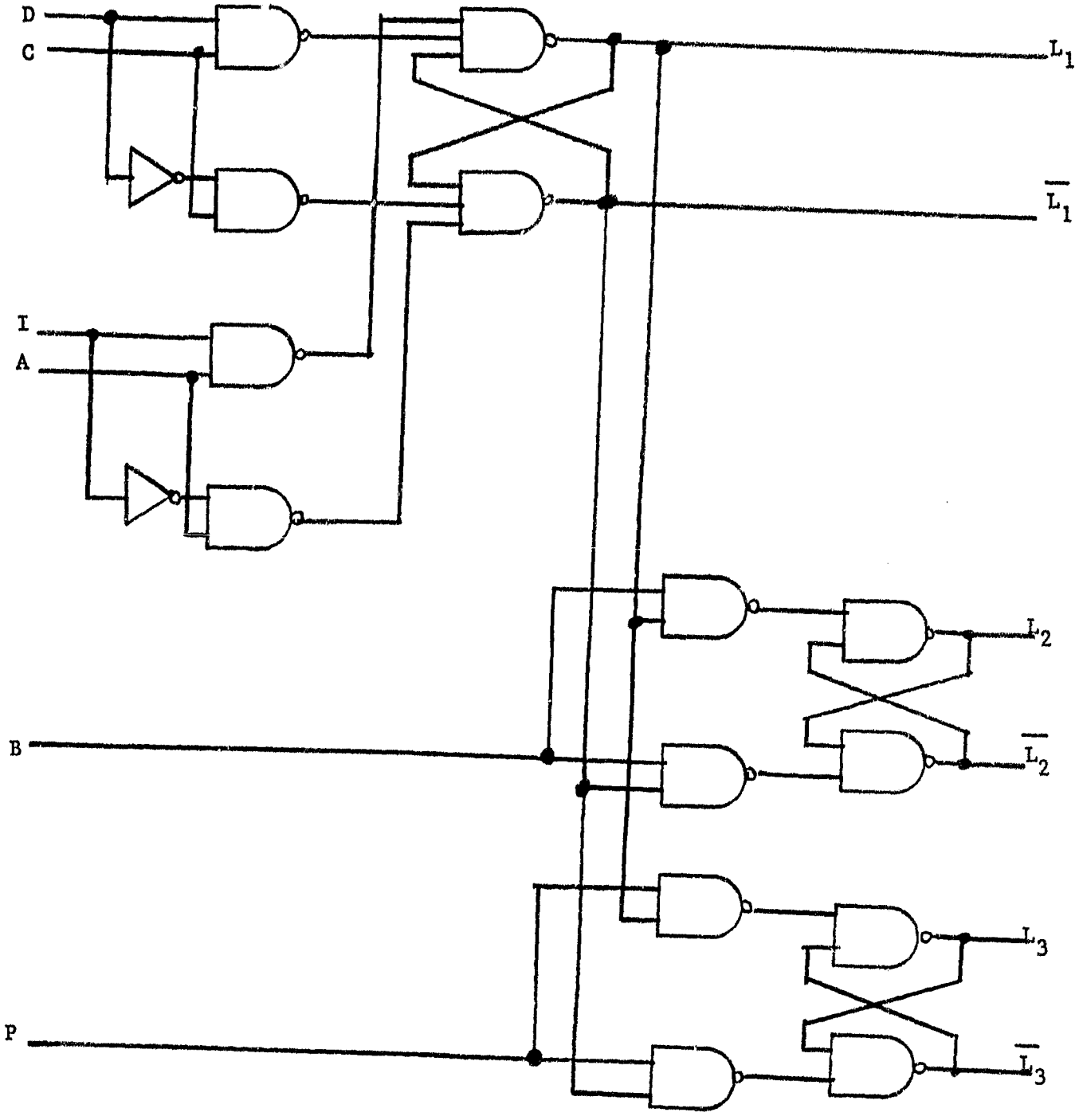


Figure 2.17 Original SSRL Implemenation

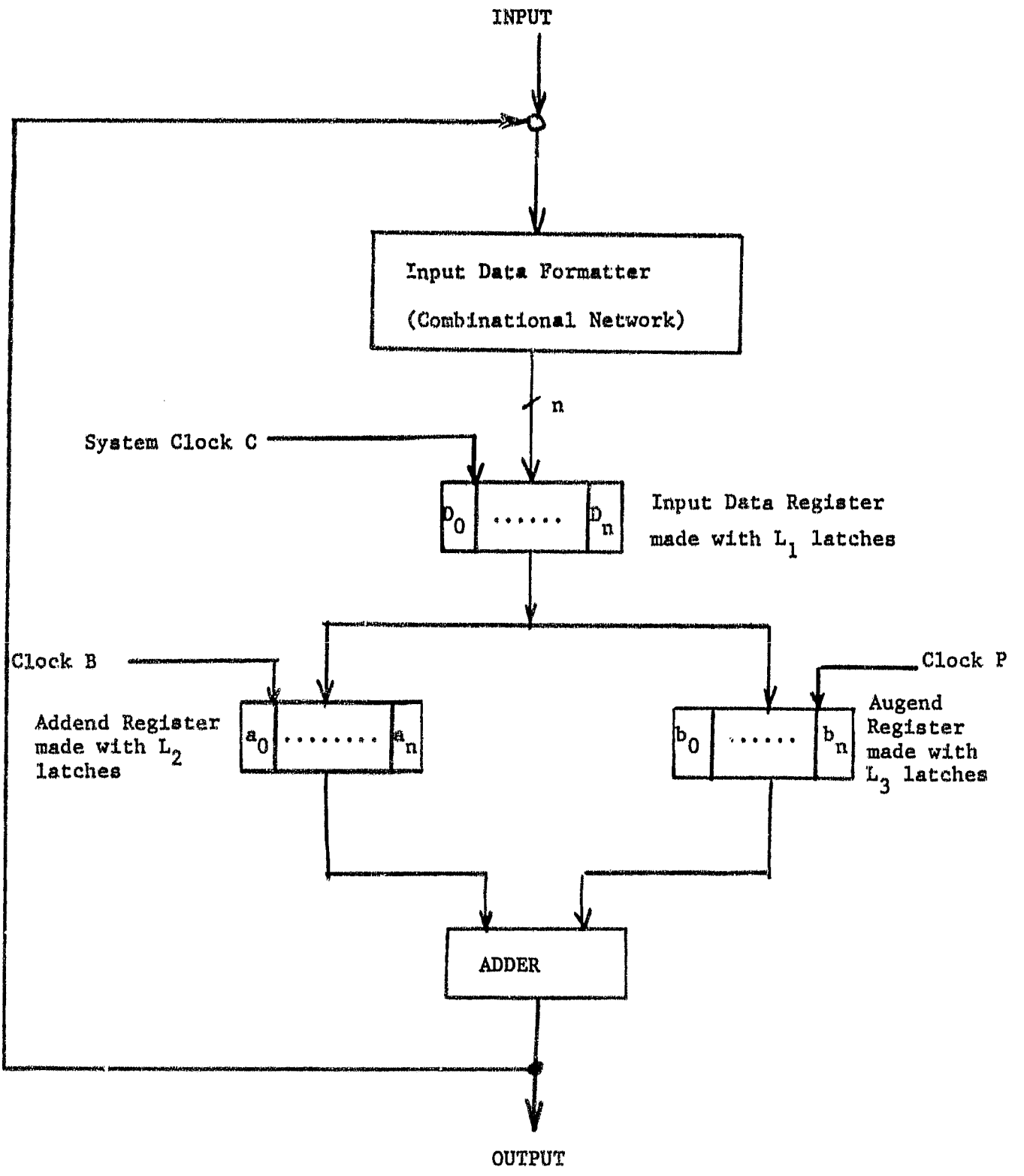


Figure 2.18 Example of Use of SSRL in LSSD Logic

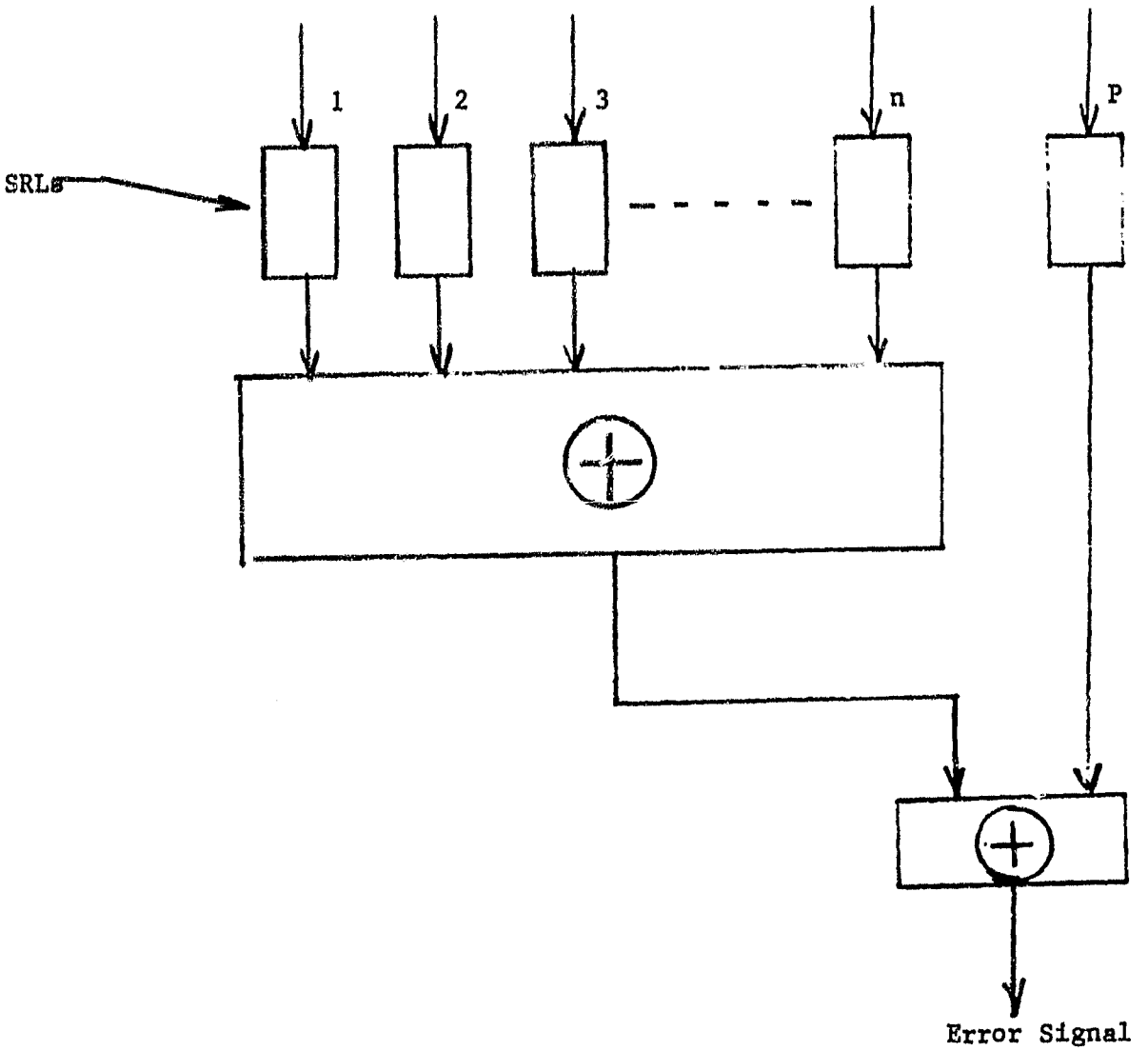


Figure 2.19 LSSD Parity Checker Implementation

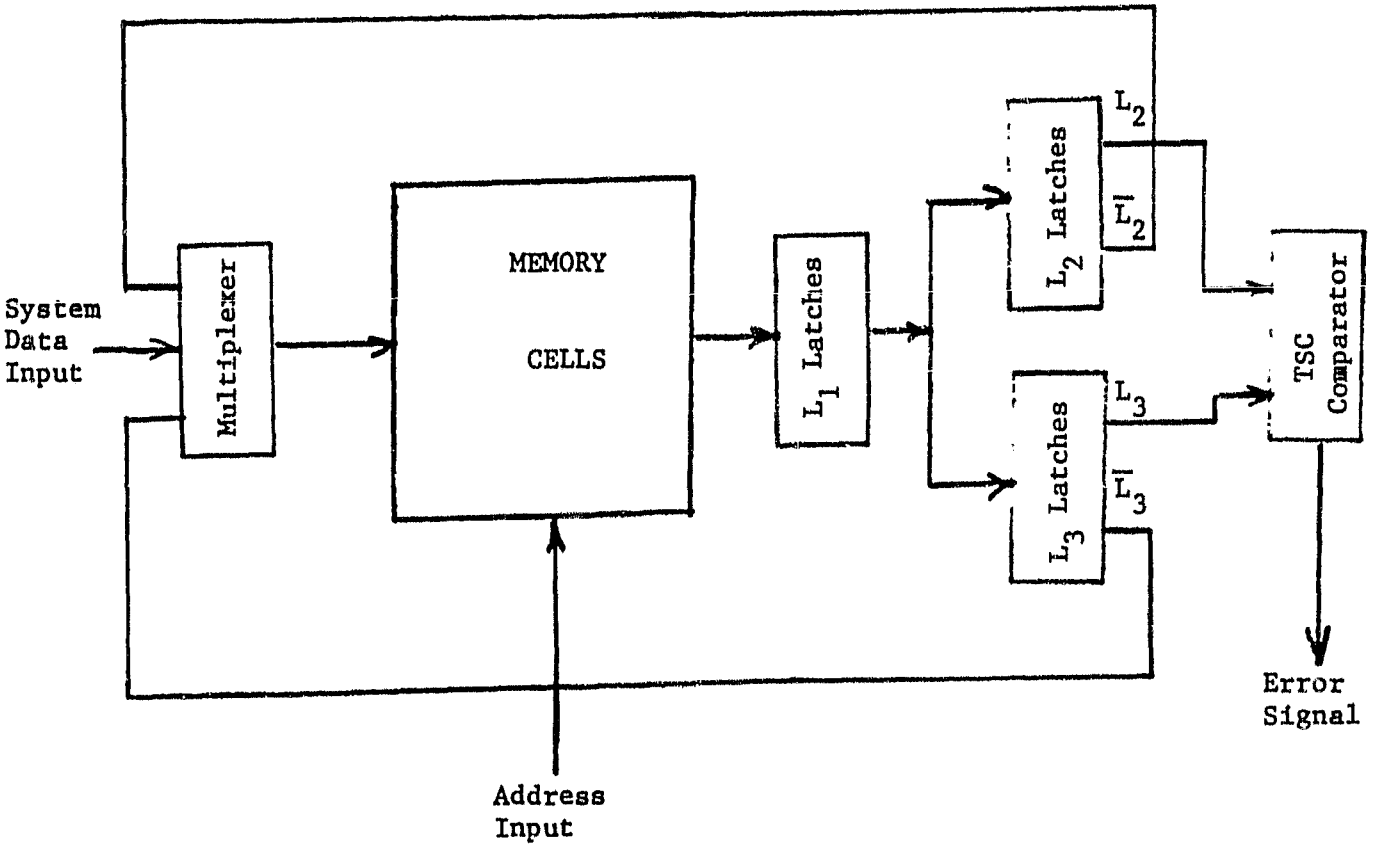


Figure 2.20 LSSD on-Line Memory Error Detection

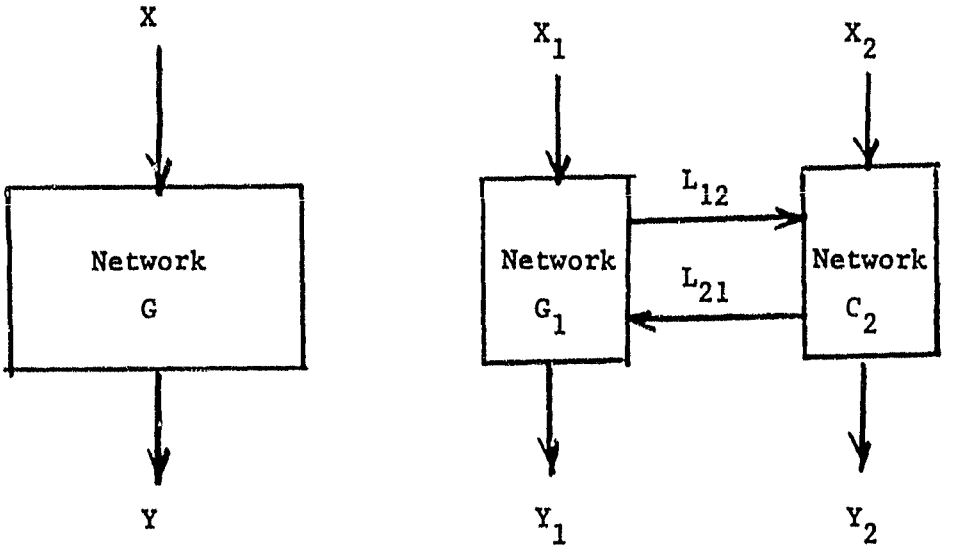
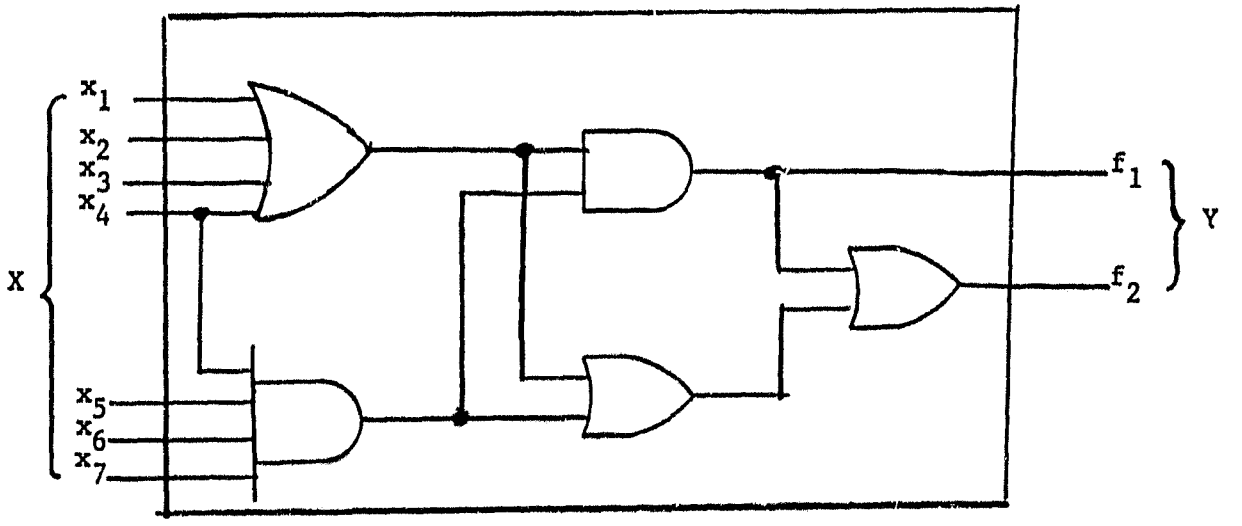


Figure 2.21 Basic Network Decomposition 2-Module Partitioning



Network G

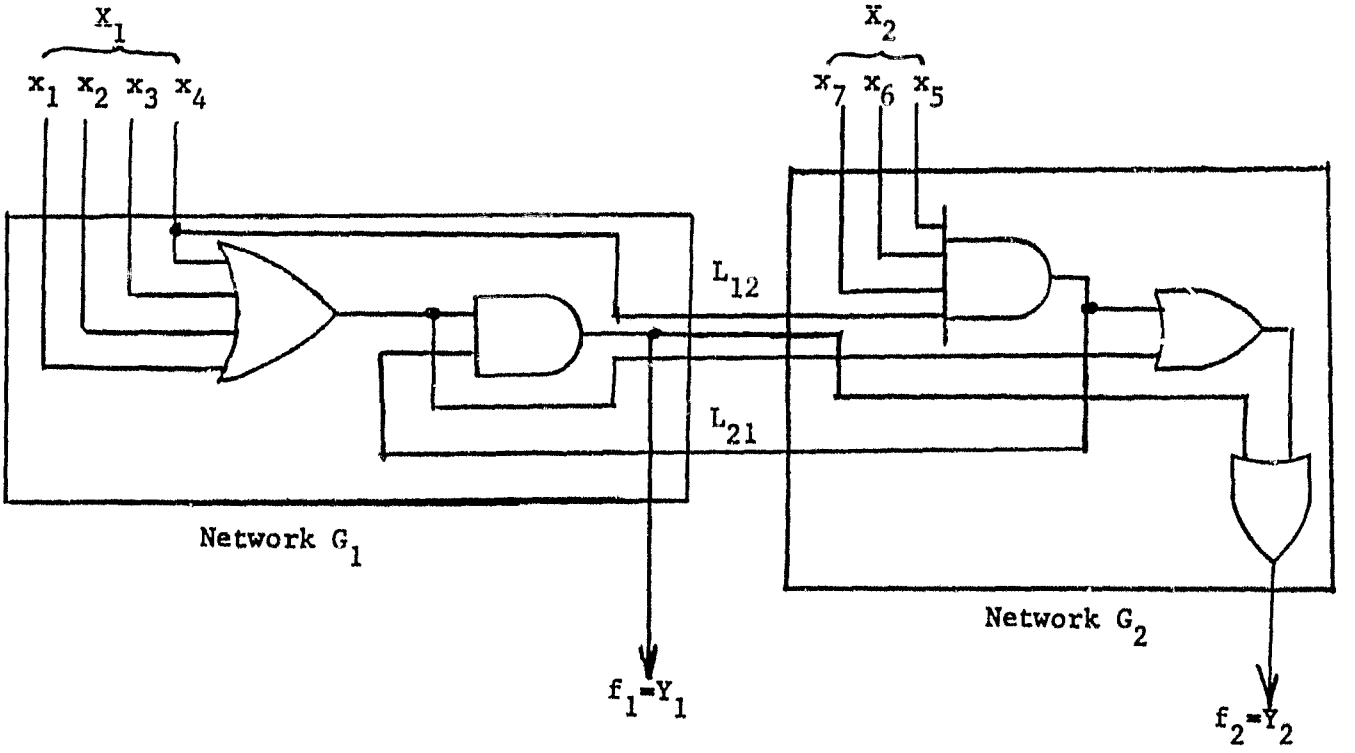


Figure 2.22 An Example of Network Partition

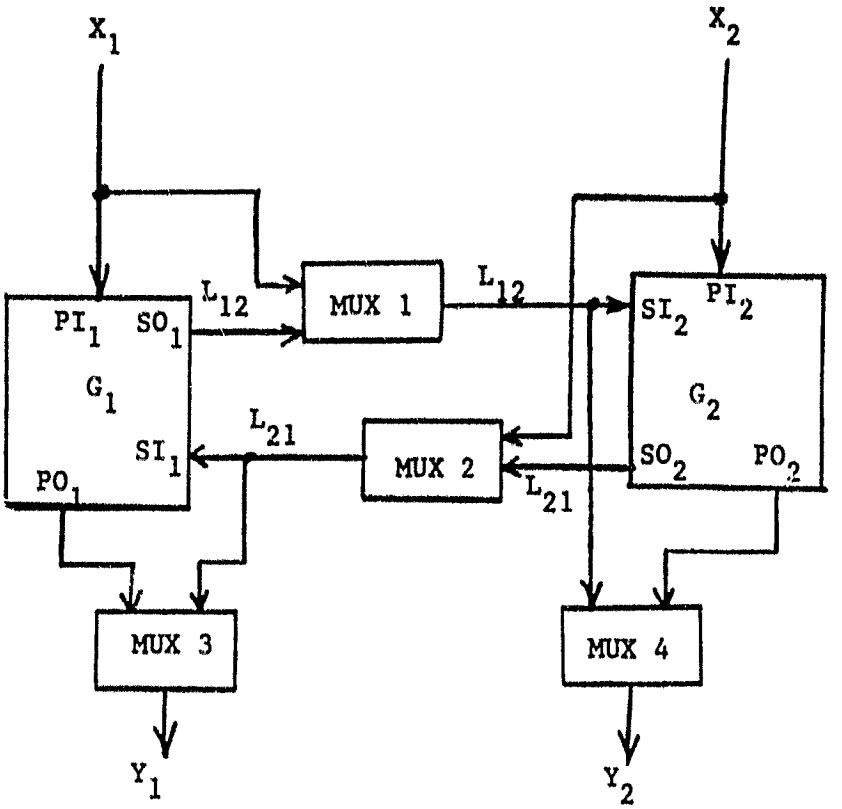


Figure 2.23a A 2-Module Routing Scheme

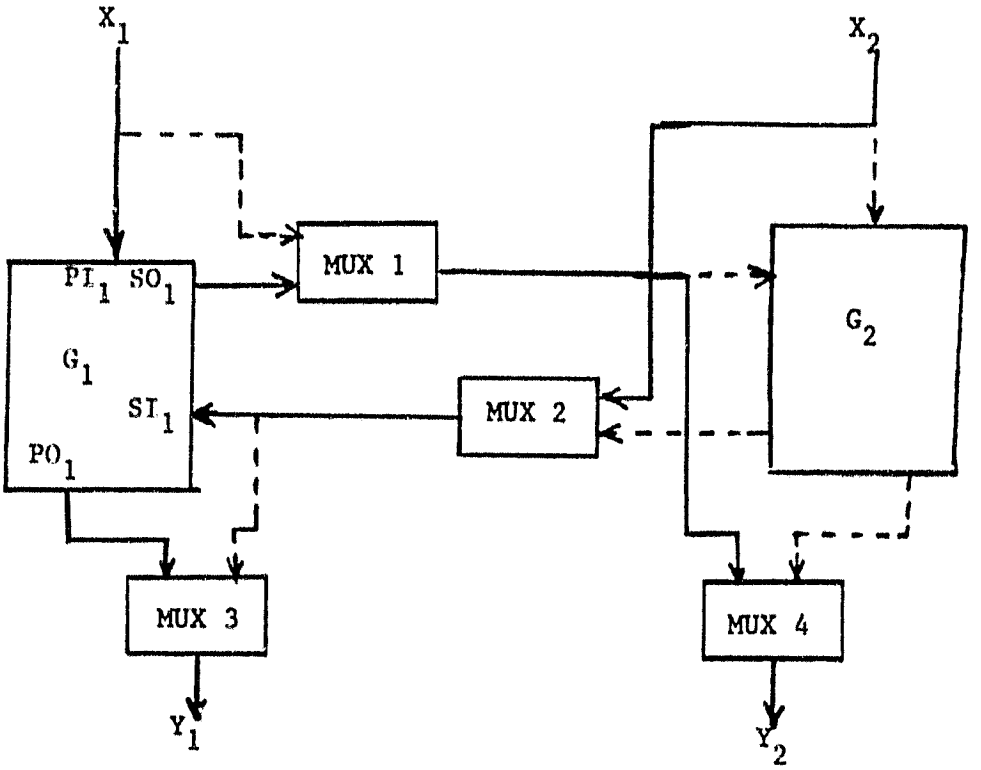


Figure 2.23b Block Diagram of Input/Output Buses Routing - G_1 under Test

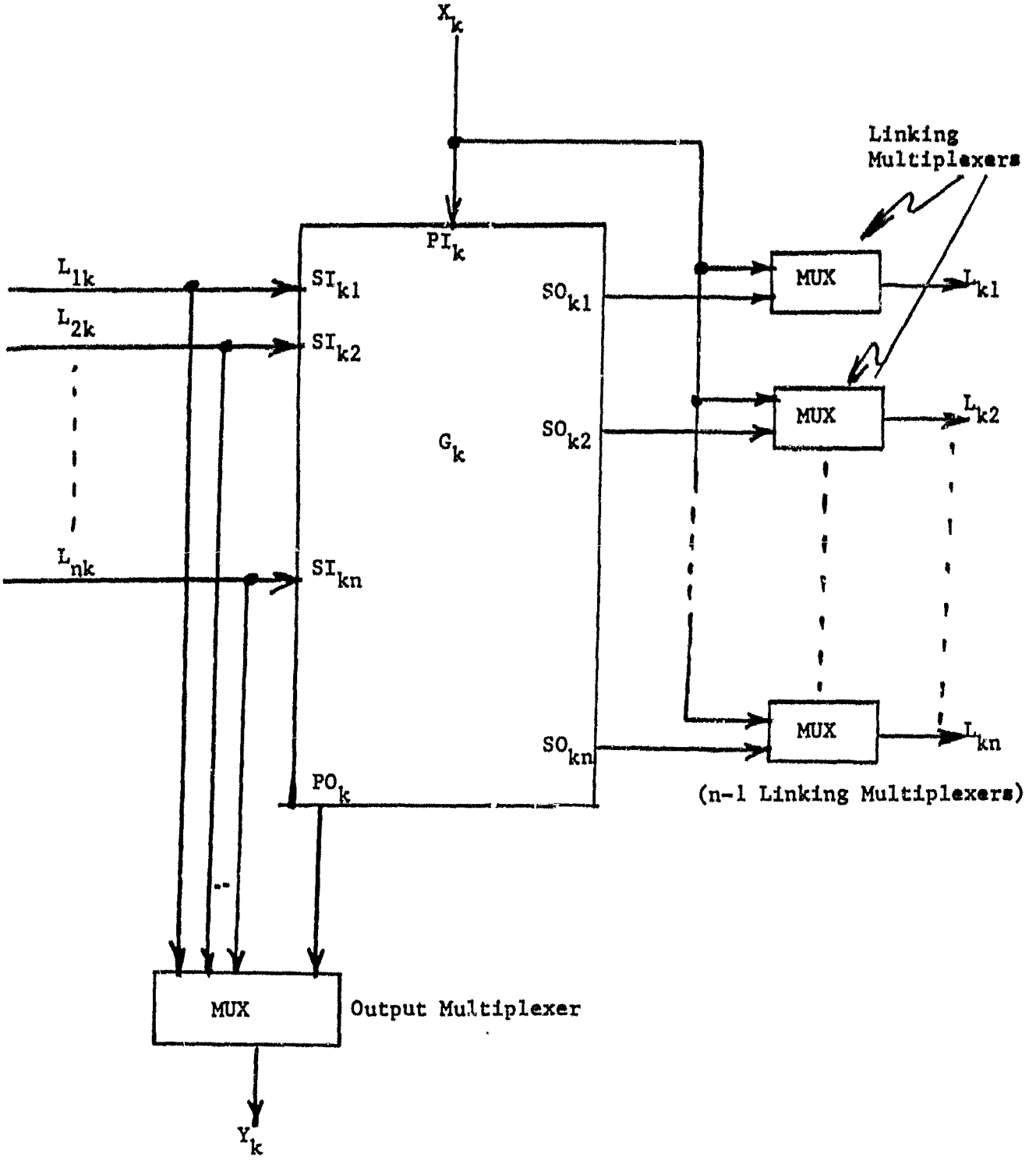


Figure 2.24 Protocol of a Routing Circuit of a Single Module

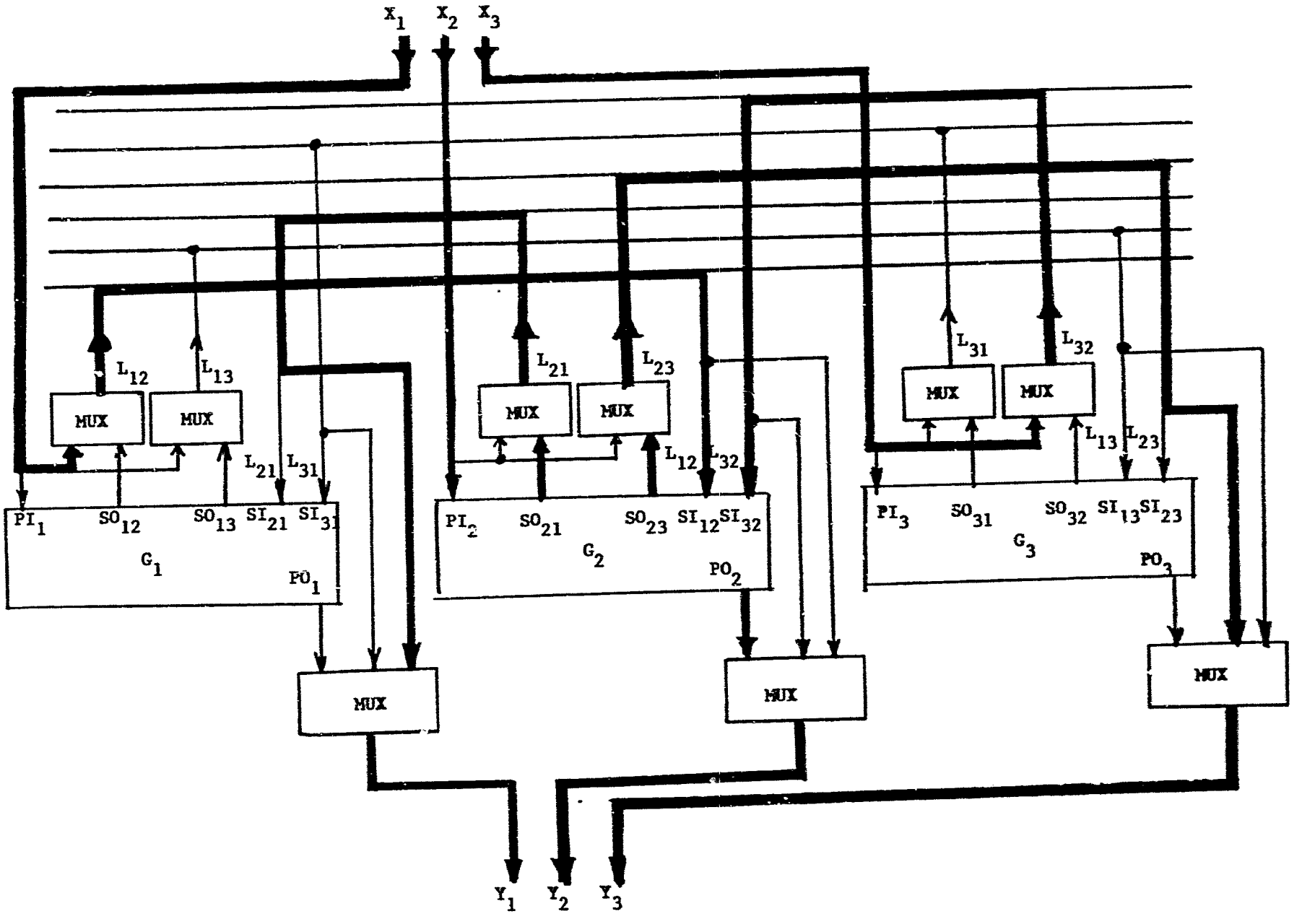


Figure 2.25 Block Diagram of a 3-Module Routing Design

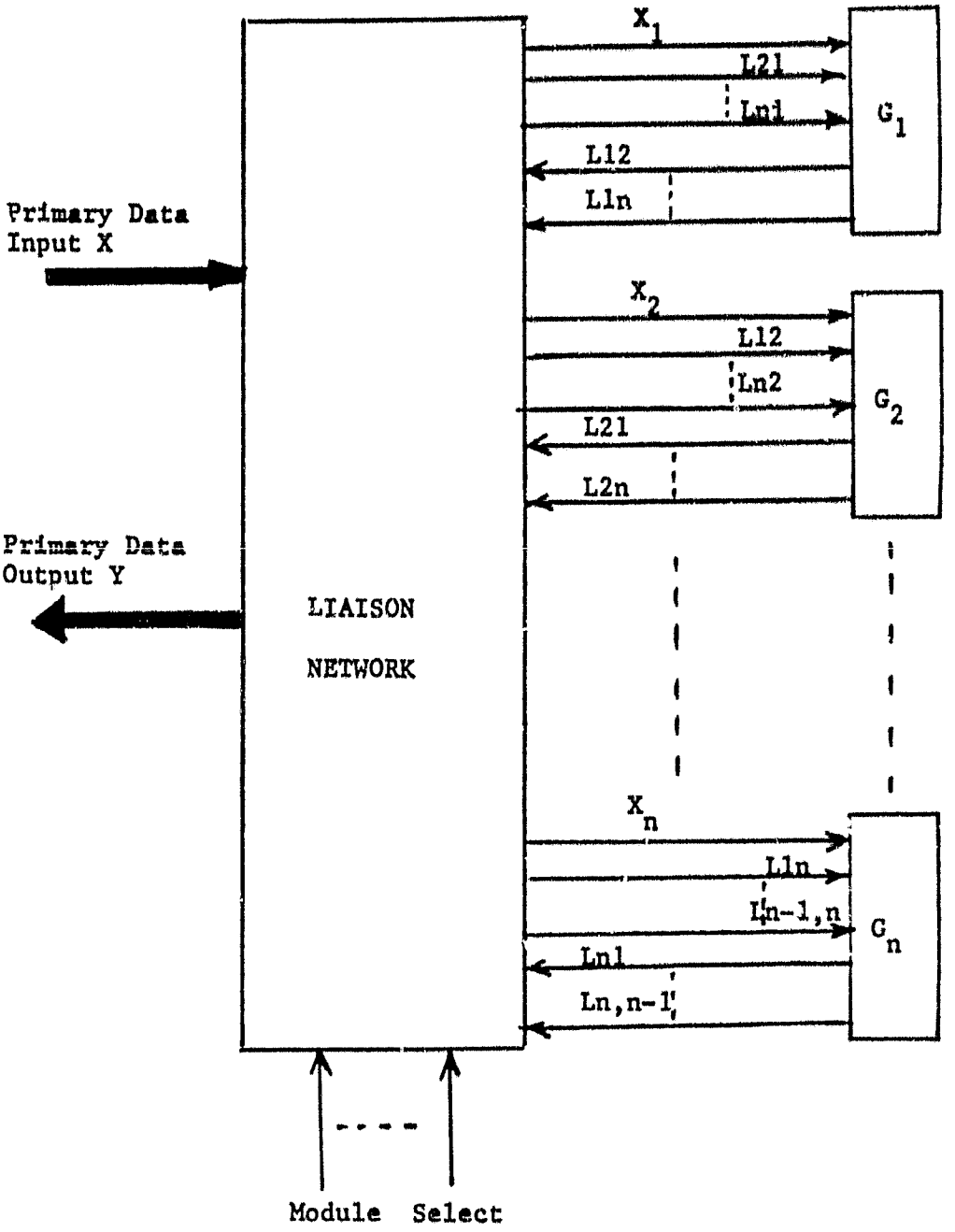


Figure 2.26 A Generalized Routing Model

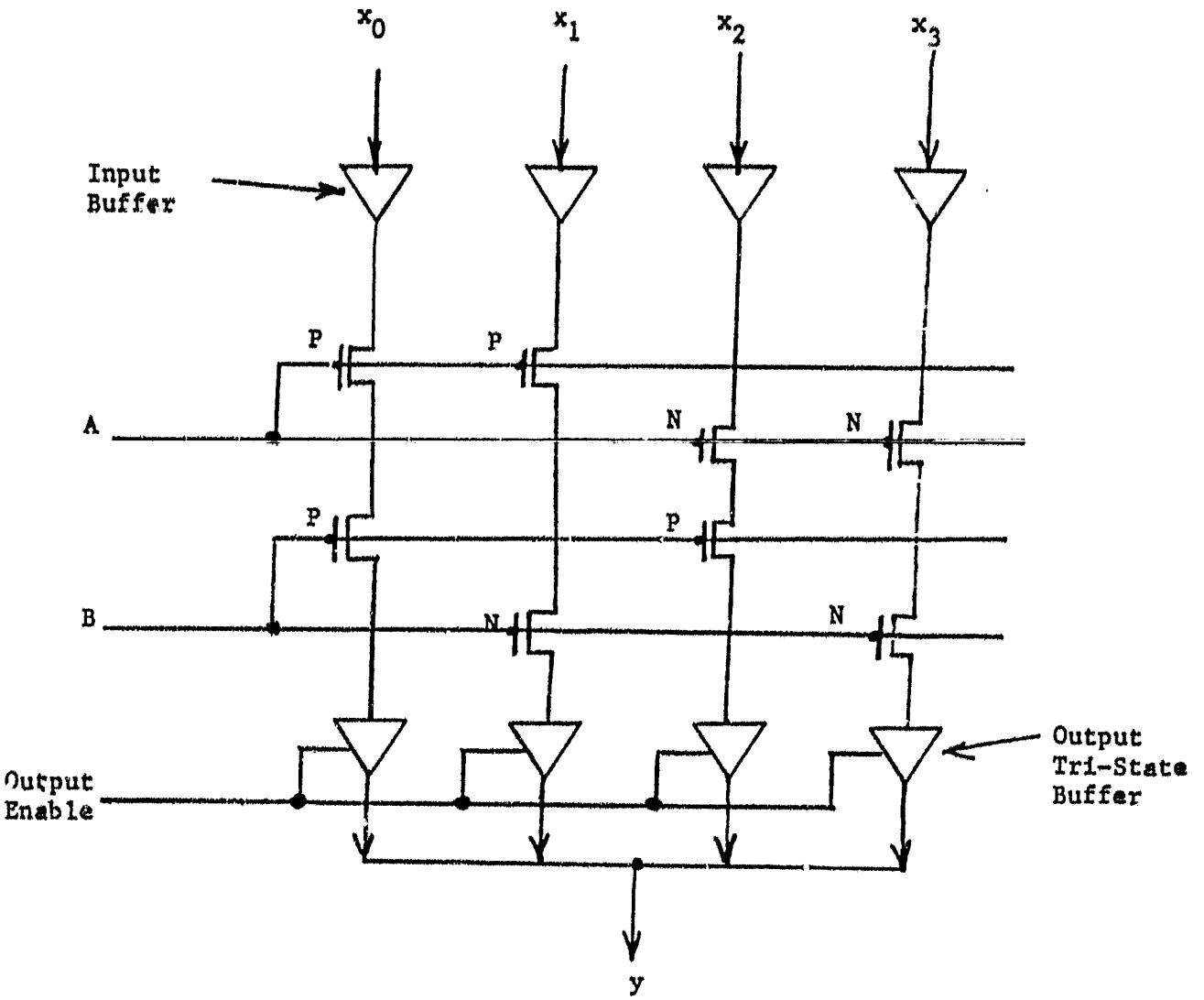


Figure 2.27 An Implementation of a 1-out-of-4 Multiplexer

SECTION 3

COMPARISON OF DFT TECHNIQUES

The viability and effectiveness of a DFT technique depends mainly on the following factors (please also refer to subsection 1.1 on "Good Testability Measures"):

(1) Effects on test generation, fault modeling, and fault diagnosis;

(2) Fault coverage;

(3) Hardware overhead;

(4) Additional I/O requirements;

(5) Effect on processing speed;

(6) Ease of implementation and utilization of added hardware for functional operations.

In the following subsections we will try to analyze the DFT schemes described in section 2 with respect to the above factors.

3.1 SELF-CHECKING TECHNIQUE

This scheme mainly uses TSC checkers and comparators to monitor the operations of various modules and check signal flows among the modules. A group of researchers at the IBM Thomas J. Watson Research Center have completed a study [28] of the cost effectiveness of employing this self-testing technique in a paper re-design of the processing unit (PU) of an S/360 computer for testability. Since this study is one of the very few similar

studies that are general enough to be considered as representative, we will use its results to support the following analysis:

(1) Effects on test generation, fault modeling, and fault diagnosis:

Since every module and every bus in the device is monitored by some checkers and comparators, circuit fault location and diagnosis can be achieved to the register-transfer level. As in the IBM study, the modified processing unit of the S/360 computer consists of ten modules (figure3.1). Each module is again partitioned at the register-transfer level such that the registers, decoders, counters, buses, etc. of the module can be monitored separately as follows:

(a) Registers - additional parity bit(s); checked by TSC parity checkers. (Figure 2.18)

(b) Decoders - all output lines decoding input configuration of even parity are connected to an EXOR gate while those of odd parity are connected to another EXOR gate. If the outputs of the EXOR gates compose a morphic 1/0, then no error/error is detected. (Figure 3.2)

(c) Internal buses and storage - coded and checked by parity.

(d) Counters with decoders - use both techniques in items (a) and (b).

Since all the checkers and comparators are totally self-checking for any single faults described in section 1.3 (see appendix B), fault modeling may not be needed. However, fault

C-2

simulation is required for fault location and diagnosis at the gate level.

(2) Fault coverage:

This scheme covers all single faults. Moreover, due to the scheme's capability of checking for all single faults and byte slice organization of storage, it is reported in the IBM study that the scheme also covers 64-80% of multiple faults.

(3) Hardware overhead:

Since this technique requires all critical functional modules (e.g., the ALU(s) of a CPU) be duplicated on chip and checkers/comparators be used to monitor module operations and data flows, the estimated minimal gate level hardware overhead is about 250%. However, some of the checking circuitry needed to check a chip can be added in the same chip itself within gate and pin constraints. Thus there is a substantial reduction on chip count overhead. As reported in the IBM study, each of the ten modules is implemented in a LSI chip and at the chip level, the self-testing design of the PU only resulted in an overhead of 6.5%.

The output of any one critical functional module can be checked for correctness by another built-in hardware scheme known as the "Store and Generate" technique due to Agarwal and Cerny [78]. Instead of duplicating the module on chip for output vector comparison, the calculated response set to a prescribed test set is stored and/or generated on chip for the comparison process. In section 4 we will present several

ways to store and/or generate the prescribed test set and the calculated response set on chip.

(4) Additional I/O requirement

The additional I/O requirement of this technique can be viewed as minimal. Additional I/O pins are required only for parity inputs and error outputs. It is estimated that an average of four extra pins are added to a 40 or 64 pin package (the self-testing IBM design averaged two extra pins per chip). However, additional I/O pins may be required if external injection of erroneous or invalid signals into the device is needed to verify the functions of the TSC checkers and comparators.

(5) Effect on processing speed

The delay effect of the TSC checkers and comparators on signal propagation is relatively insignificant due to the fact that the checking is done in parallel with the functional processing. However, the time needed for error detections and corrections of memory words may be large compared to the memory cycle time. This delay effect can be minimized by the employment of a memory processor that constantly checks the memory contents.

(6) Implementation

The TSC checkers and comparators described in subsections 2.1.2 and 2.1.3 are very easy to implement on chip. They do not take up much chip real estate and their "array" type structure is well-suited for automated logic design.

Consequently, design errors can be kept minimal. Finally, the checking circuitry cannot be used for functional operations.

3.2 LEVEL-SENSITIVE SCAN DESIGN TECHNIQUE

In this structured design scheme, there are two main design concepts:

(1) Correct operation of all internal storage elements does not depend on signal rise/fall times and on circuit delays.

(2) All internal storage elements are externally controllable and observable.

Following these design concepts, a trial redesign [29] of the Texas Instruments 74S481 four-bit slice microprocessor component used in the B-52 AP101-C Offensive Avionics Subsystem Computer was carried out by Rawlings, Rosenbluth, and Groves of the AFWAL Avionics Lab and of IBM to study the viability and cost-effectiveness of this LSSD technique. In the following we will use the results of this study to substantiate our discussions:

(1) Effects on test generation, fault modeling, and fault diagnosis:

Since in an LSSD design all internal storage elements (the SRLs and/or the SSRLs) are externally controllable and observable, test generation is greatly simplified. In view of testing, modules with buried sequential nets are converted into pure combinational nets. Thus, automated test generation time can be made to increase almost linearly with gate count instead of exponentially with sequential complexity. More-

over, ac testing and manual intervention of test generation can be avoided. The problem of fault modeling is at the present restricted to that of dc stuck-at fault modeling due to the fact that the effects of all other types of circuit faults on the behavior of a complex buried sequential net is not yet well-understood. Further research is to be done in this area.

As mentioned previously, fault diagnosis can be achieved to within a combinational net of a module (sub-module level). The boundary of a module can be defined by the SRLs and/or SSRLs of the module itself.

(2) Fault coverage

This scheme could achieve 100% of dc stuck-at fault coverage if all logical redundancy in a design is eliminated. In the redesigned version of the 74S481 (designated the SCS481), there are only 171 out of 7046 (2.5%) possible dc stuck-at faults that were determined to be undetectable, and 126 out of the 171 undetectable faults are due to logical redundancy. After the redundant circuits were eliminated from the design, a 99.35% of dc stuck-at fault coverage was obtained.

(3) Hardware overhead

Hardware overhead at the gate level is highly dependent on the complexity of the sequential networks. It takes six gates to implement an edge-triggered data flip flop, 10 gates to implement an HFPH SRL, and 16 gates to implement an SSRL. Therefore, if every internal storge element (memory excluded)

of a design is implemented in SSRL, then a (16-6)/6 = 170% gate overhead results. However, many of the SSRLs can be utilized, with proper circuit reconfiguration by clock controls, as functional processing elements such as data registers. Moreover, the gate count of the combinational nets of the design has a decreasing effect on the net gate overhead figure. Therefore, the average logic gate overhead is about 4 to 20% [12]. Amazingly, the figure on gate overhead reported in the AFWAL study is a mere 2.7%.

Quite clearly, the above small gate overhead figures suggest that the chip overhead would be close to 0%.

(4) Additional I/O requirement

At the chip level, at least six additional I/O pins for clocks C_1, C_2, A, and B, and for the scan-in and scan-out) are required in an SSRL design. For a 64 pin package, this figure translates to a 10% I/O increase. At the board or system level, however, these additional I/O pins can also take the roles of the system I/O pads required for standard operator/CE console interface. In the AFWAL study, the additional I/O figure at the board level is 6.5% (46 pins on the original design vs 49 pins on the revised design).

(5) Effect on processing speed

Under normal operation and with the single latch design structure (figure 2.12b), the processing speed of the device is not changed because only the L_1 latches are used for internal data storage. Using the cycle stealing technique,

scan-in/scan-out may be done without affecting the normal operation speed. On the other hand, this DFT scheme may even improve the processing speed if dynamic diagnosis is required to ensure proper system operations at all times. This is due to the fact that good diagnosability and serviceability are inherent in the design structure. The AFWAL study reported no significant decrease in processing speed of the SCS481 as compared to that of the 74S481.

(6) Implementation and hardware utilization

The design structures of this technique (section 2.2) are well-defined. All internal storage elements are implemented in SRLs and/or SSRLs, and the boundary of every module can be defined by the SRLs and/or the SSRLs. Therefore, this technique actually aids the automated design process. Finally, as mentioned earlier, careful and thorough design consideration can lead to very good utilization of added logic in the device.

3.3 MULTIPLEXED ROUTING TECHNIQUE

The main objective of this scheme is to partition the functional hardware into independent modules under the test mode such that each module can be exhaustively tested. As the device gets more complex, there is a clear-cut advantage of circuit partitioning for testing over other conventional testing techniques. Recent reports [30] pointed out that the circuit partitioning technique can reduce testing time up to 100 fold. For example, Motorola

Inc. has been highly successful in reducing production test time (from several minutes to several seconds per chip test) on its 68000 16-bit microprocessor through the use of an extra 2% of chip area for diagnostic logic and a unique partitioning test method that separates the control part from the data handling part of the chip. Other manufacturers such as Zilog Inc., Intel Corp., and Texas Instruments Inc. are also in favor of this technique [30]. McCluskey demonstrated the viability of his scheme by applying it in a redesign of the TTL74181 ALU/FUNCTION GENERATOR. We will use his results in the following discussions:

(1) Effects on test generation, fault modeling, and fault diagnosis

Realistically, test generation and fault modeling may not be totally eliminated by the employment of this technique in DFT circuit design because in large and complex LSI/VLSI devices, due to chip real-estate and power constraints, large scale modularization may not be possible. However, test generation and fault simulation run time can indeed be drastically reduced even if the modularization is done on a moderate scale. It has been observed that, in general, if a network is partitioned into m modules which can be independently tested, then the test generation and fault simulation time is reduced by a factor of m [31]. Further, test time is also reduced. In McCluskey's study, the 74181 is partitioned into five modules, and a saving of 16 times in the number of exhaustive test vectors is achieved.

(2) Fault coverage

The fault covering capability of this scheme is highly dependent on the degree of partitioning. If the device is so partitioned that exhaustive testing of each of the partitioned modules becomes feasible, then the fault coverage issue becomes unimportant because any undetected faults would be considered to have occurred in some redundant circuit of the device, and thus, such faults can be eliminated. But if exhaustive testing of the individual modules is not practical, then this scheme will have little contribution to fault coverage. In some extreme cases, this scheme may increase fault coverage if due to the routing capability of the liaison network, some originally non-sensitizable internal lines become externally controllable/observable.

(3) Hardware overhead

This scheme requires the most hardware overhead at the gate level. In general, a 100% area overhead (which may be much larger than the gate count overhead) may result due to the large number of extra routing multiplexers and decoders and the complex network of extra signal routes. In McCluskey's design, the gate count overhead is about 30% (28 additional gates vs 92 original gates). It is estimated one or two extra chips are required to house the liaison networks at the board level, which amounts to about 5% of chip count overhead.

(4) Additional I/O requirement

The only additional I/O pins are those required for the input of selection codes to the multiplexers for test module selection. The number of such additional I/O pins is clearly a function of n, the number of partitioned modules, and of circuit configuration of the decomposed circuit. It is expected to have an upper bound of $(1+n)\log_2 n$. This is because if there are n modules, each module will have n-1 linking multiplexers and one output multiplexer (figure 2.24). Each linking multiplexer has two input buses (the right and the left input buses). Only one selection line is required to select the right or the left input buses of all n-1 linking multiplexers that are associated with a module. Thus we need n such selection lines because we have n groups of n-1 multiplexers to control. With encoding, we only need $\log_2 n$ primary signal lines to set or reset any of the n selection lines. Now, for the n output multiplexers, each of them has n input buses. Therefore, for each output multiplexer, we need $\log_2 n$ selection lines. Hence for all n output multiplexers, we need $n\log_2 n$ selection lines. Therefore, we need in total $\log_2 n + n\log_2 n = (1+n)\log_2 n$ primary input lines to control all the multiplexers. For example, if a design is partitioned into 4 modules, at the most 10 additional I/O pins are required for the control of the multiplexers. However, substantial reduction of additional I/O pins can be achieved if the circuit is

well-partitioned. McCluskey has shown in his redesign of the TTL74181 ALU/FUNCTION GENERATOR that only three additional I/O pins are required to partition the circuit into five independent modules. On the other hand, circuit configuration may jeopardize any attempt to reduce additional I/O pins, and thus, the number may reach the upper bound value.

(5) Effect on processing speed

The delay effect is mainly due to the time delays of the routing multiplexers. It can be minimized if the multiplexers are implemented with pass transistors as shown in figure 2.27.

(6) Implementation

The only implementation problem is the difficulty in routing the large number of extra signal lines on chip. With the present LSI/VLSI implementation technology, the technique of orthogonal interweaving of diffusion and metal lines may be used to solve this problem. Also, total reconfiguration of the network modules may be necessary.

We have tried to analyze the effectiveness of the Self-Testing, the LSSD and the Partitioning DFT techniques with respect to the six pertinent factors. The table in figure 3.3 lists the results of the analysis. From there we can see that none of the three techniques can be considered as superior to the others. A logical conclusion may be that a good DFT technique may be one that

combines all three of these techniques together. In the following section, we will try to present a DFT design procedure that uses all three techniques.

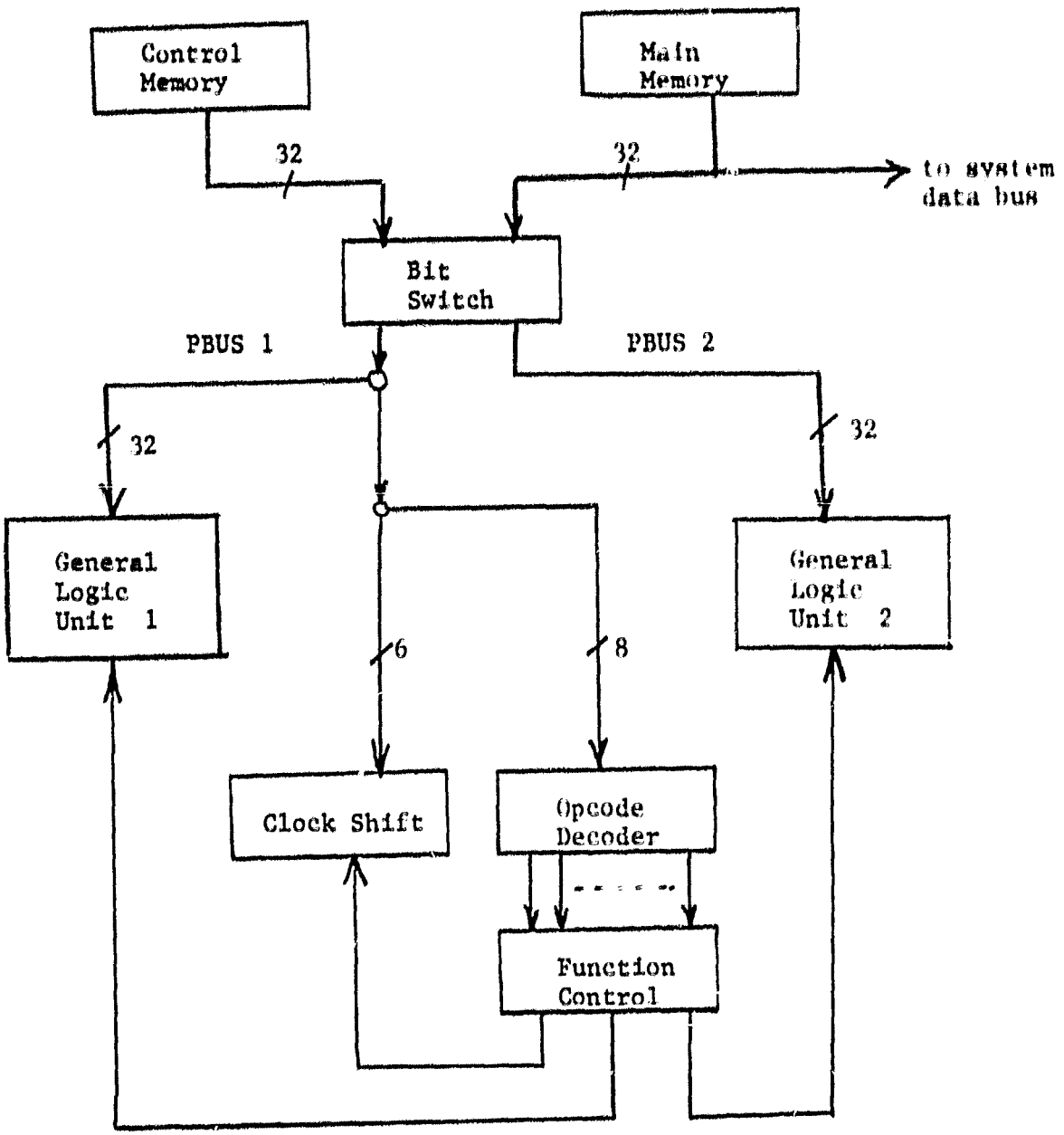


Figure 3.1. Block Diagram of IBM S/360 Processing Unit

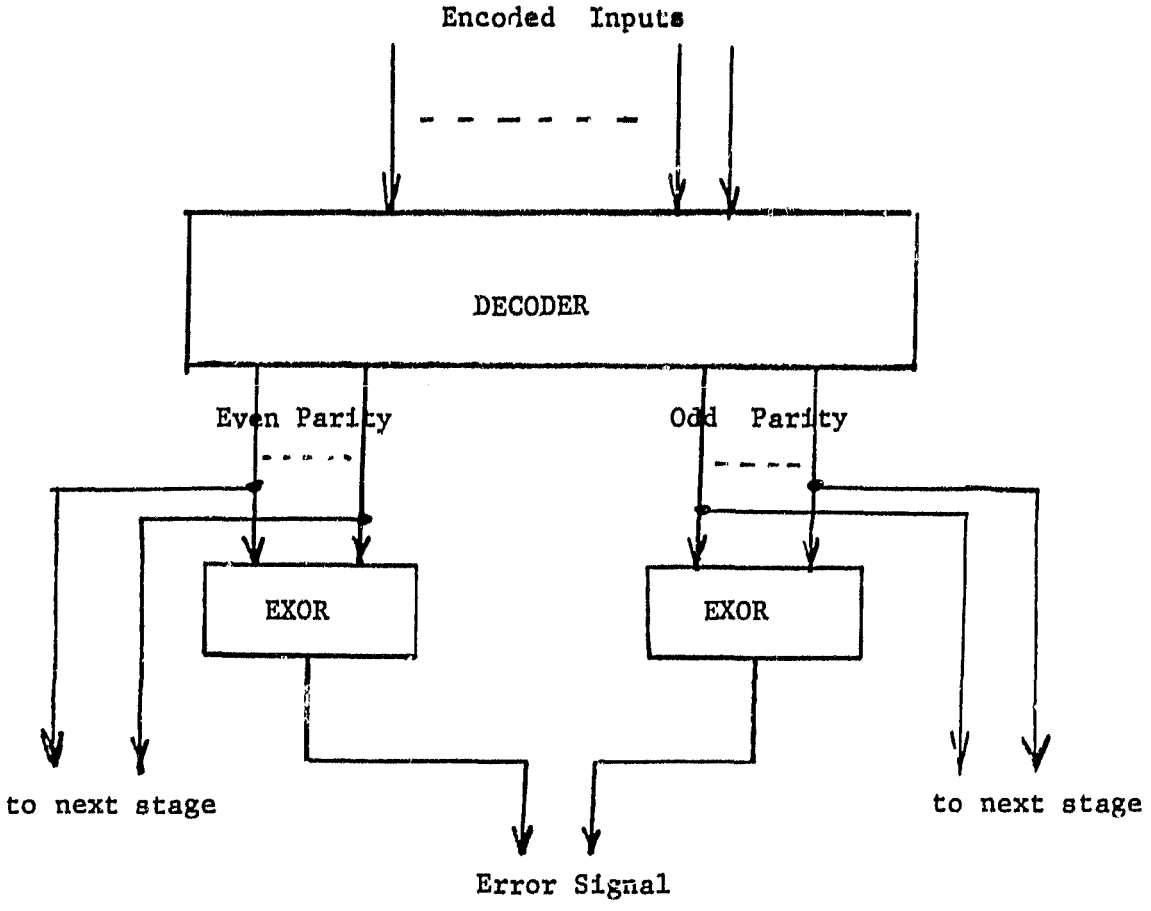


Figure 3.2 Scheme for Decoder Checking

	Self - Testing	LSSD	Partitioning
Effect on Test Generation, Fault Modeling and Fault Diagnosis	Register level testing and fault diagnosis; Little effect on test generation; Fault modeling not necessary.	Sub-network level testing and fault diagnosis; Greatly simplifies test generation; DC stuck-at fault model only.	Module level tesing and fault diagnosis; Reduces test generation time; Test generation not necessary if can be exhaustively tested.
Fault Coverage	Near 100% single faults; 64-80% multiple faults.	Near 100% single stuck-at faults.	Optimal fault coverage if exhaustive testing is allowed; Otherwise no significant effect on fault coverage.
Hardware Overhead	About 250% at gate level; About 10% at board level.	4-20% at gate level; Near 0% at board level.	About 100% at gate level; Less than 5% at board level.
Additional I/O Requirement	Averaged 4 extra pins per package.	About 10% extra I/O pins.	Number of extra pins = $(1+n)\log_2 n$, where n is the number of modules.
Effect on Processing Speed	Relatively insignificant.	Basically no effect.	Small delay due to time delay of routing multiplexers.
Implementation and Hardware Utilization	Highly suited for LSI/VLSI implementation; Zero utilization of added hardware.	Easy to implement; Good utilization of added hardware.	Routing of extra signal lines may be difficult; Network module reconfiguration may be necessary.

Figure 3.3 DFT Techniques Comparison Table

SECTION 4

A GENERAL APPROACH TO DESIGN FOR TESTABILITY

From the discussions in the previous three sections, we can draw some conclusions on the testability requirements of LSI/VLSI logic design :

(1) High degree of modularization, in both the "Operation" and "Test" modes;

(2) Inputs and outputs of each module are externally accessible;

(3) Circuit operations are independent of timing properties of the device;

(4) The state of all internal storage elements (except memory arrays) are directly controllable and observable;

(5) Operations of all critical modules are monitored in real time;

(6) Built-in memory error detections and corrections if viable;

(7) All internal and external data/address buses are monitored in real time, built-in signal propagation error corrections if possible.

It is clear that in order to incorporate all of the above seven DFT requirements into a logic design, all three of the DFT techniques, namely, self-checking, LSSD, and partitioning, have to be employed in the design. The following describes a fundamental design approach to such a design objective. In subsection 4.1 we present a set of general DFT guidelines, and in subsection 4.2 we propose a general design structure for the implementation of the

guidelines. An example based on the Intel 8748 general purpose microcomputer chip is also included for illustration.

4.1 GENERAL DESIGN FOR TESTABILITY GUIDELINES

G1 : With the multiplexed routing technique, partition the CUT into smaller modules. In general, the size of any one module should not be bigger than that of a macrocell ($\leq$10k gates, see section 1.4.5, page 1-15.

G2 : Implement all internal storage elements of each module with SRLs/SSRLs and apply the LSSD technique to allow direct access of these storage elements from both the primary inputs and outputs.

G3 : Monitor all modules that are determined to be critical to the correct operations of the CUT with TSC code-checkers and comparators. Built-in response set generator and/or storage may be used to implement this guideline (see subsection 4.3).

G4 : Code all input and output vectors to and from every module with error detecting/correcting code(s). With slight modification, the TSC code-checker described in subsection 2.1.2 can be used as a TSC parity code generator. Furthermore, memory array module(s) should at least be coded with SEC/DEC Hamming code.

G5 : All clock signal lines should be directly controlled by primary inputs. For a more formal set of design rules on clock signal lines, please refer to design rules 2, 3, and 4

of [12].

G6 : (i) If physically and economically viable, all built-in TSC code-checkers and comparators should report, to a built-in Diagnostic Processor, the real time status of all modules and data/address buses. Once the diagnostic processor is signalled of an error, it will initiate a sequence of diagnostic operations such as

(a) critical error determination;

(b) error corrections;

(c) on-line dynamic scan (see Section 2.2.4.4, page 2-27);

(d) roll-back operations; and

(e) declaration of device malfunction with error information sent to the outside world.

(ii) If not viable to include a diagnostic processor on chip, at least a TSC "Reduction Circuit" should be built on chip instead. This reduction circuit, commonly known as RCCO (reduction circuit for checker outputs), is used to encode all outputs of the TSC code-checkers and comparators such that the internal error information of the device can be sent to the outside world through a small number of primary output pins (normally ≤ 5).

4.2 A GENERAL DFT STRUCTURE

Figure 4.1 shows a general design structure with which the above DFT guidelines can be implemented into a logic design (a device). First the device is partitioned into several modules using the multiplexed routing technique. In figure 4.1 we show a 6-module partition only for illustration purpose. It is up to the discretion of the designer(s) on how the device is to be partitioned. Most often the block diagram of the structure of the device generated at the early design phase will furnish a clear picture of module boundaries. A basic approach to this problem is to partition the device by function. For example, a general one-chip microcomputer like the Intel 8748 (figure 4.2) can be partitioned into three modules : the Control Module (CM), the Data Processing Module (PM), and the Memory Module (MM).

For each module in figure 4.1, there is a TSC I/O bus checker to monitor the data flow. The I/O buses of each module are directly connectable to the primary inputs and outputs of the device via the liaison network. The input bus to each module consists of (a) the coded input data bus, (b) the scan-in line and (c) the system clock bus. The output bus from each module consists of (a) the coded output data bus, (b) the scan-out line and (c) internal error signal output lines*.

---

* For every module, additional code-checkers and comparators might be built in to monitor its operations. For example, the PM of Intel 8748 may have its ALU duplicated on chip, and the

Note that all internal error signal output lines from each module go to the liaison network which routes them to either the Diagnostic Processor, DP, (dotted block) or the primary output of the device or both.

Now, in order to implement the LSSD and the partitioning techniques, we need two extra sets of primary inputs : one for the input of LSSD control signals (clocks A, B, C_1, and C_2) and one for the input of control signals to the liaison network. Here in figure 4.1 we see that we have used a bus multiplexer (controlled by a line called "LSSD/PARTITION") and a register array (called the "Liaison Network Control Registers (LNCR)") at the front end of the device to implement the additional input requirements. This way the two extra sets of primary inputs can be sent into the device through a single bus only. The size of this bus is often determined by the number of control lines required for the liaison network (normally ≤ 5). This scheme works because of the fact that under the "test" mode, often only one module is being tested, and the control lines to the liaison network are kept constant throughout the test period of that module. Therefore, the liaison network control vector is first sent into the LNCR to select the test module. Then the "LSSD/

---

two identical ALUs are compared by a TSC comparator (figure 4.3). All registers (PC, AC, FR, DMAR, and I/O buffers) may also be monitorred by TSC code-checkers. Therefore, a internal error signal bus is present at the output PM.

PARTITION" line to the bus multiplexer is reset from logic 1 to logic 0, enabling the bus multiplexer to route its input bus to the LSSD Control Signal bus. At this time LSSD operations can begin.

Finally, note that if there is a diagnostic processor or some functionally equivalent hardware built into the device, we may build in some test set and response set generator(s) and/or storage as one or several of the modules to periodically check any critical module under test. In many cases, real time monitoring can be substituted with periodical checking of critical modules to guarantee reliable operation of the device. The following subsection presents some of the techniques for test set and response set generation and storage.

4.3 TEST SET AND RESPONSE SET GENERATION AND STORAGE

The block diagram of figure 4.4 depicts the general "store/generate and compare" technique. First the calculated test set T and the calculated response set R are either stored in and/or generated by the Input Test Set Complex (ITSC) shown by the dotted block and the Output Test Set Complex (OTSC) respectively. Note that the input test vectors can be sent to the CUT externally through the primary input pins of the device, thereby eliminating the ITSC. For each test vector passing through the CUT generating a true response vector, the corresponding calculated response vector is sent from the OTSC. These two output vectors are then stored in the two data

registers separately. A clocking signal "CC" is then used to clock the vectors into the TSC comparator for equality comparison.

The main problem of this scheme is the large memory space required by the ITSC and/or the OTSC. An improved version of this scheme was proposed by Agarwal and Cerny, which they termed as the "Store and Generate Built-in Testing" approach [18]. The block diagram in figure 4.5 depicts this scheme, which was designed to reduce the storage requirement. There we see that the ITSC is composed of a ROM memory storage and a linear feedback shift register. The ROM simply contains a set of vectors $A=[a_1, a_2,..., a_r]$. Upon the control of the counter and for each vector a_i sent from the ROM, the linear feedback shift register can generate another set of vectors $S_i=[a_{i1}, a_{i2},..., a_{is}]$. Therefore a set of r s vecotrs $[a_{ij}]$, i$\in$ [1,2,...,r] and j$\in$ [1,2,...,s], can be generated by the ITSC. In order to use the set $[a_{ij}]$ to detect all the faults in the prescribed fault set F, it remains a problem of calculating the set A and determining the structure of the linear feedback shift register such that the calculated test set T is equal to or a large subset of the set $[a_{ij}]$. There is no unique solution to this problem. The following describes a primitive approach to this problem:

Let the test vectors of the test set T be n tuples and let the ROM have bit locations $b_1, b_2, b_3,..., b_k$, $k = 2^n$. First the test set T is rearranged such that the test vectors are in an ascending order (i.e. $T=\{t_1, t_2,..., t_m\}$, $t_1<t_2<...<t_m$

and $t_1 \geq \vec{0}$). Starting with the zero vector, do the following:

(1) Set i and j to 1;

(2) Set $\vec{v}$ to $\vec{0}$;

(3) If $t_i = \vec{v}$, then set b_j to 1 and increment i

else set b_j to 0;

(4) Increment $\vec{v}$ and j ;

(5) If $i \leq m$, then go to (3) else stop.

For example, suppose the following are the calculated test set (left column) and the rearranged test set (right column) with n=4:

Test Vector	Calculated	Rearranged
t_1	0100	0010
t_2	1011	0100
t_3	1000	0101
t_4	0111	0111
t_5	1110	1000
t_6	0010	1010
t_7	0101	1011
t_8	1010	1110

then following the algorithm presented above, the set A is found to be:

i	j	$\dot{v}$	t_i	$a_j=(b_j)$
1	1	0000	0010	0
1	2	0001	0010	0
1	3	0010	0010	1
2	4	0011	0100	0
2	5	0100	0100	1
3	6	0101	0101	1
4	7	0110	0111	0
4	8	0111	0111	1
5	9	1000	1000	1
6	10	1001	1010	0
6	11	1010	1010	1
7	12	1011	1011	1
8	13	1100	1110	0
8	14	1101	1110	0
8	15	1110	1110	1

After set A has been found and stored in the ROM, the calculated test set can be generated using a simple up-counter which starts the up counting with the zero vector, generating vectors c_1, c_2,..., c_p, p=j, and c_1=zero vector. Each bit from the ROM is used to gate the corresponding vector from the counter to the CUT. Therefore, for every a_j=1/0, the vector c_j is sent/not sent to the CUT. To reduce the testing time, the up-counter can be run in a very fast rate and is only paused whenever the output from the ROM is a

logic 1 to allow enough time for the test vector to propagate through the CUT. Then a "Resume Count" signal may be sent from the output comparator to the counter to restart the normal counting operation (figure 4.6).

Using this "Store and Generate" scheme, the required memory space, in some cases, can be substantially reduced. As illustrated in the above example, there are 4x8=32 bits in the test set while there are only 15 bits in set A.

A similar scheme can be used to "store and generate" the test response set. The following describes a method proposed by Savir [32] to reduce the storage requirement of the test response set. It is called a "Syndrome Test". This scheme is only applicable to combinational circuits. Since the combination of the LSSD and the partitioning techniques can be used to convert any logic circuit into a combinational circuit, this scheme remains an effective tool for built-in testing.

Definition : The syndrome S(F) of a Boolean function F is defined as $S(F) = K(F)/2^n$, where K(F) is the number of minterms realized by F, and n is the number of binary input lines.

Therefore, given a combinational circuit, the syndrome of each of its primary output lines is basically the number of 1s appearing at this output line after all possible input combinations have been applied to the circuit.

Now, if the combinational circuit is so designed that its fault-free and faulty (single stuck-at fault) versions have different

syndromes, then the circuit can be tested by comparing the actual syndrome generated by the circuit after all input combinations have been applied to it with the calculated fault-free circuit syndrome (figure 4.7). This special design of combinational circuits is called "Syndrome Testable Design". Detailed design principles and procedures can be found in the paper of Savir [32]. Basically, the design method is to insert extra I/O to the circuit to modify the realizations of the given function(s) such that the resulting circuit becomes syndrome-testable. The extra I/O requirement of this scheme does not pose additional problems due to the fact that the built-in LSSD structure can be utilized to furnish the required extra I/Os to the circuit through the scan-in/scan-out operations. Note that in this scheme, the ITSC is simply an up-counter which counts from zero to 2^n-1. Thus the problem of "Store and Generate" of the input test set is virtually eliminated.

There are some other similar methods proposed to solve this "Store and Generate" problem such as the transition count technique and the ones-count technique, which proved to be quite effective for some specific cases. Further research is inevitably needed to furnish more general and efficient solutions.

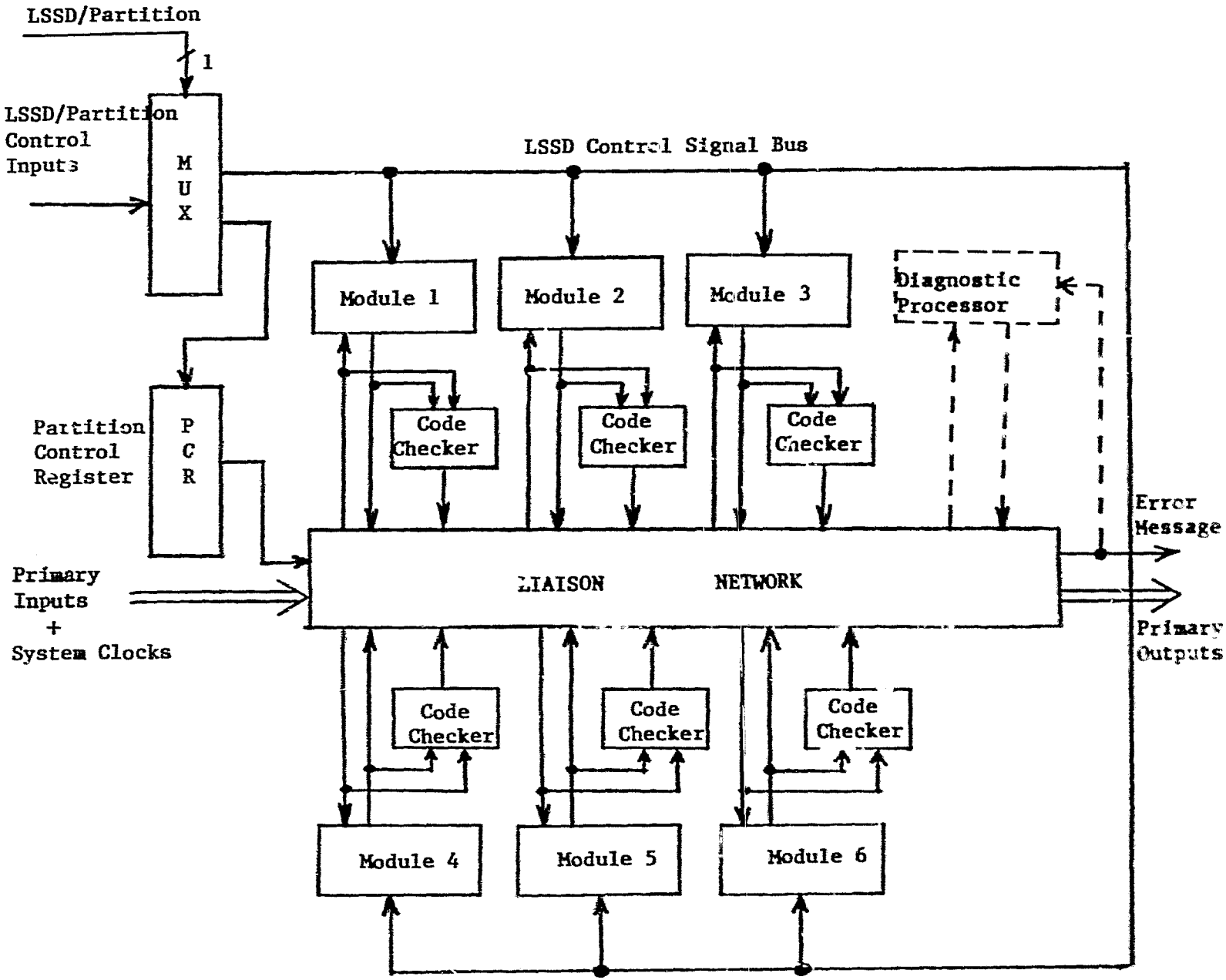


Figure 4.1 A General DFT Structure

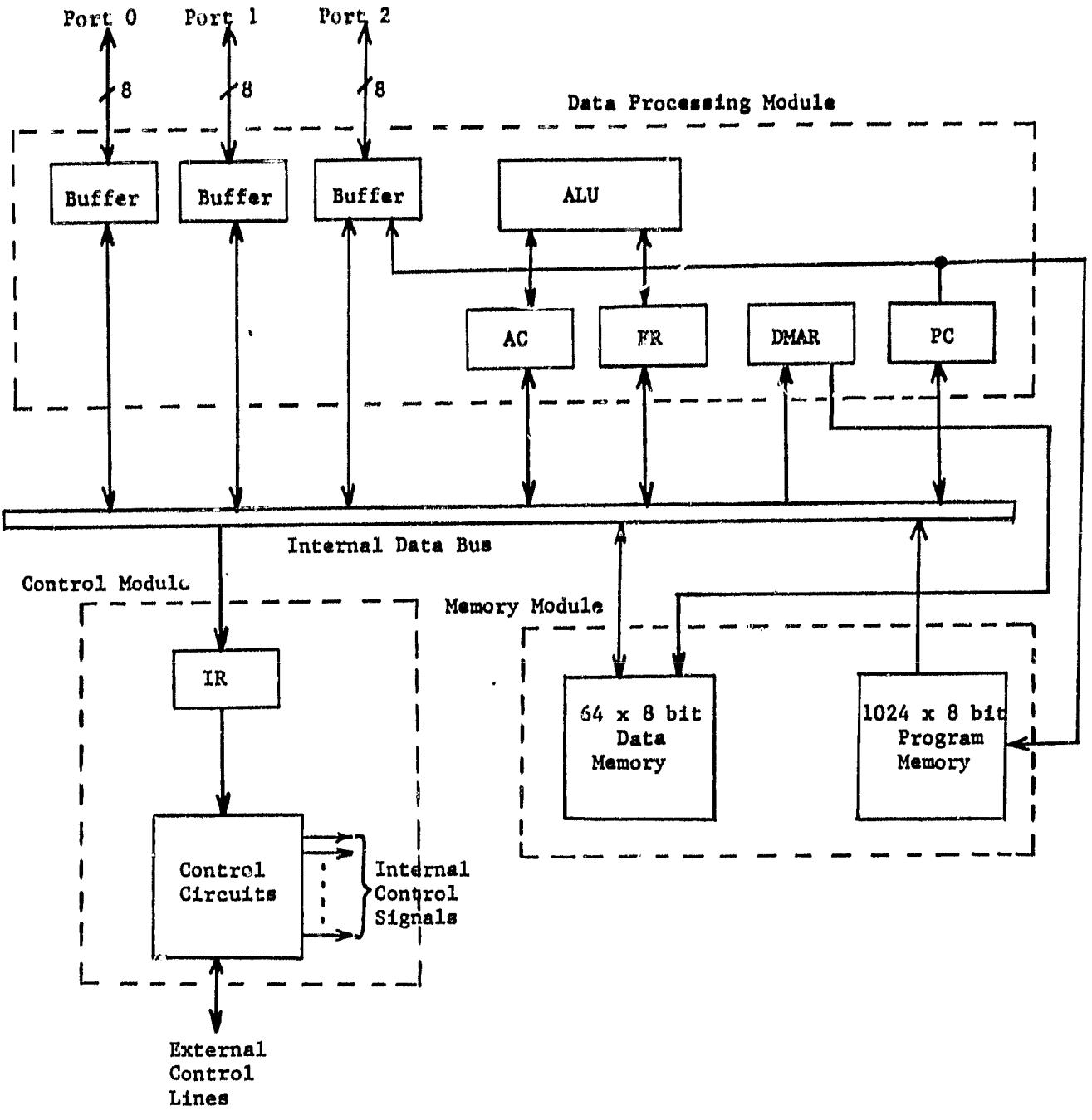


Figure 4.2 Block Diagram of the Structure of Intel 8080 Microprocessor (shown in 3 partitioned modules)

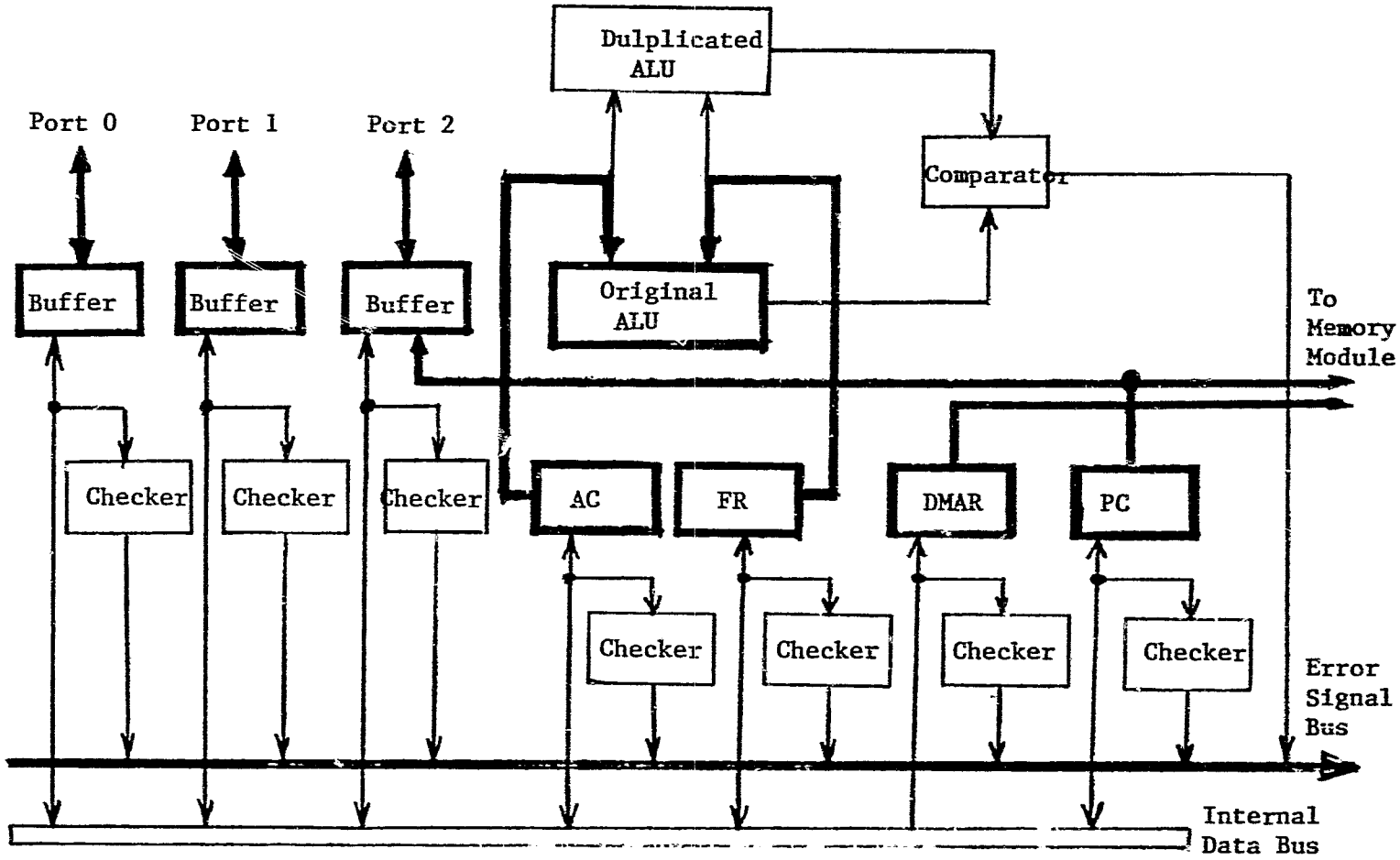


Figure 4.3 Real-Time Monitoring of Critical Module, an Example

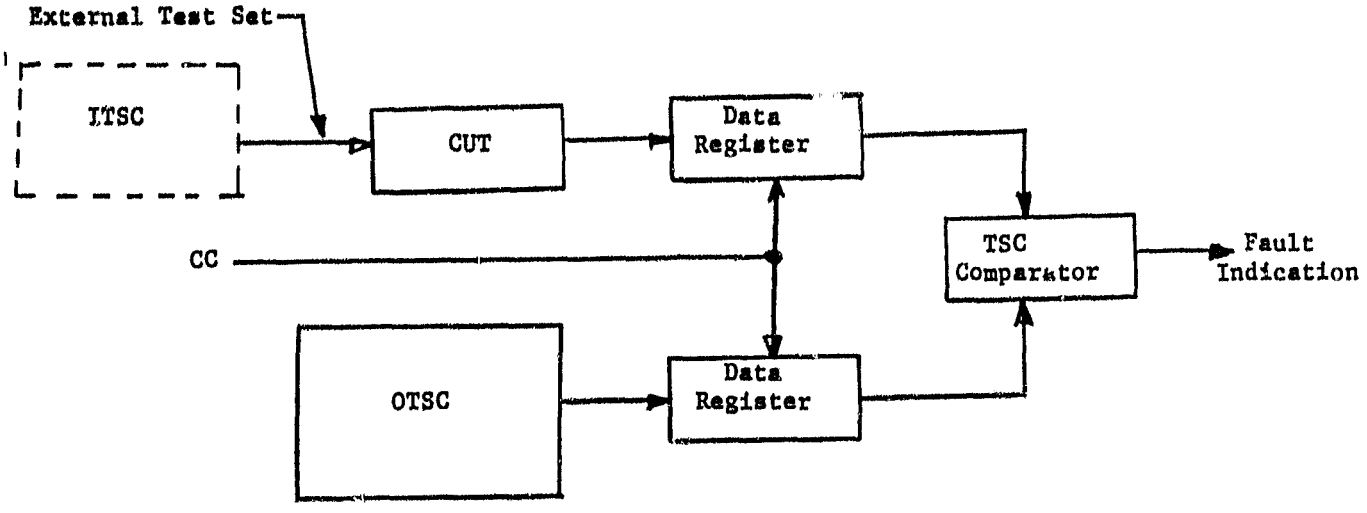


Figure 4.4 General "Store/Generate and Compare" Technique

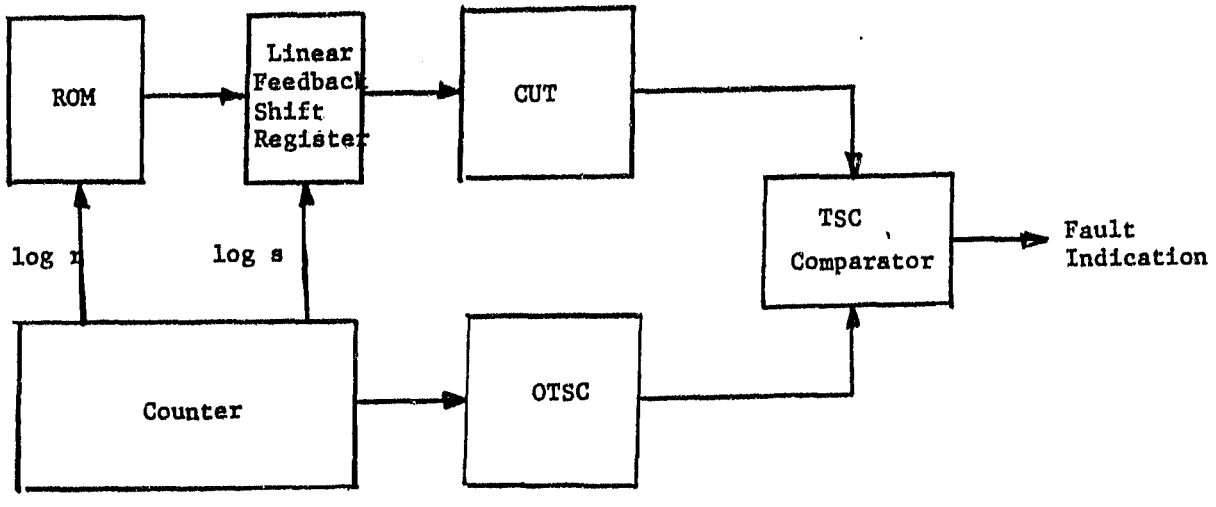


Figure 4.5 General "Store and Generate Built-in Testing"

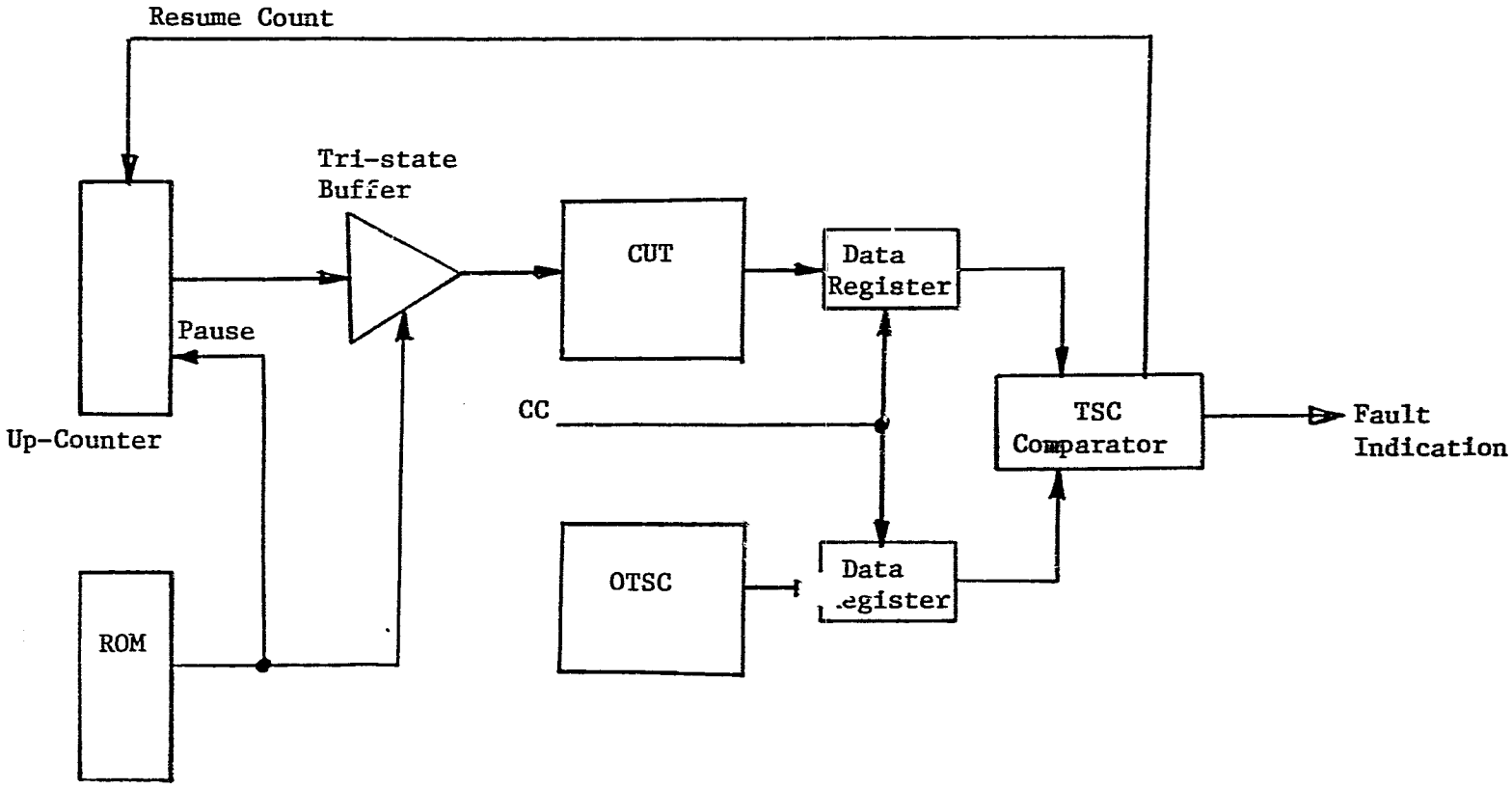


Figure 4.6 Block Diagram of Test Set "Store and Generate" Scheme

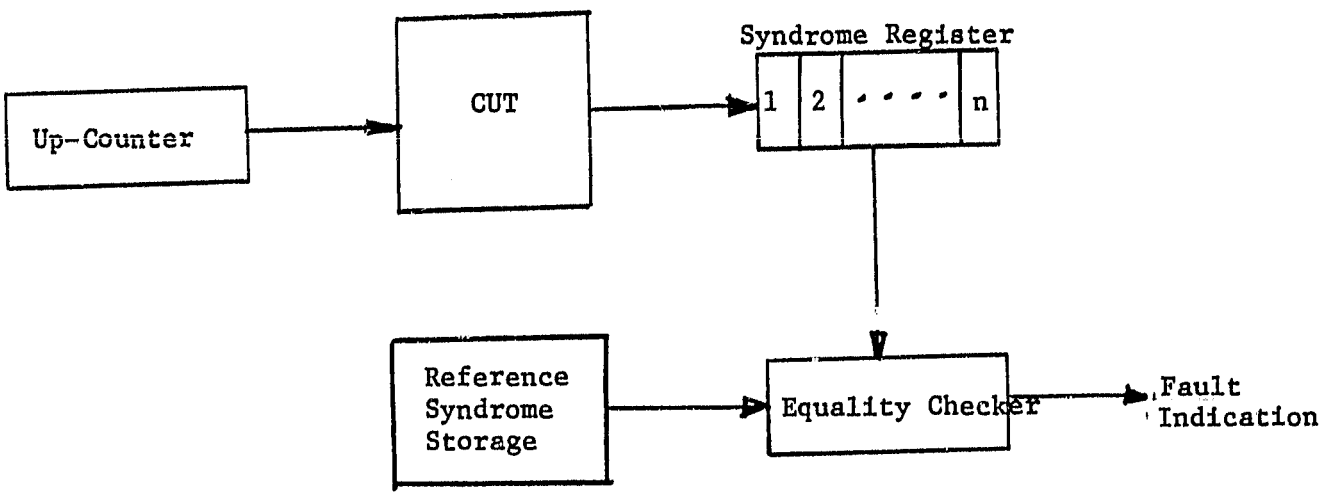


Figure 4.7 Block Diagram of Syndrome-Test Scheme

SECTION 5

CONCLUSIONS

In this report we have shown that the incorporation of testability characteristics into large scale digital design is not only necessary for but also pertinent to effective device testing and enhancement of device reliability. We have also shown that there are at least three major DFT techniques, namely, the self-checking, the LSSD, and the partitioning techniques, each of which can be incorporated into a logic design to achieve a specific set of testability and reliability requirements. Detailed analysis on the design theory, implementation, fault coverage, hardware requirements, application limitations etc. of each of these techniques are also presented. A more general DFT approach, which combines the strong points of all of the three DFT techniques, is presented in the form of a set of general DFT requirements which defines what should be incorporated into a design, a set of general DFT guidelines which tells what to do to satify the DFT requirements, and a general DFT structure which shows how to implement the set of DFT guidelines. Finally some methods are discussed on the built-in storage and generation of test and response sets, which are essential to the implementation of dynamic checking of critical modules for device reliability.

As the final note of this report, we list in the following two interesting research topics for the readers to explore :

(1) The error latency problem of the TSC code-checkers and

comparators is still one major weak point of the self-checking technique. Making the circuits from totally self-checking to totally self-correcting for single stuck-at fault may be a solution to this problem. It is expected in the realm of large scale integration technology, Quaded Logic or Interwoven Logic [33] may serve as a practical tool for such type of circuit design.

(2) As mentioned before, the problem of built-in test/response set storage and generation poses an interesting research topic. Clearly, the way the input test set is stored/generated may have an effect on the way the response set is stored/generated and vice-versa. For example, if a test set is stored/generated by the scheme described in section 4.3, pages 4-7 to 4-9, then the response set to this generated test set may be very random in order and thus may be impossible to generate with a reasonable amount of hardware and ROM space. It is suspected that something similar to the "syndrome" approach - signature analysis, for instance, may be worthwhile for further investigations.

REFERENCES

1. V.V. Nickel, "VLSI-The Inadequacy of the Stuck at Fault Model," Digest of Papers, GOMAC 80, Nov. 1980, Houston, Texas, pp. 331-334.
2. T.W. Williams and K.P. Parker, "Testing Logic Networks and Designing for Testability," Computers, Vol. 12, No. 10, October 1979, pp. 9-21.
3. J.M.H. Heines, "Built-in Test and VHSIC/VLSI Technology," Electronic Test, October 1980, pp. 60-73.
4. J.E. Stephenson and J. Grason, " A Testability Measure for Register Transfer Level Digital Circuits," Proc., Int'l Symp. on FTC, June 21-23, 1976, pp. 101-107.
5. R.L. Wadsack, "Fault Modeling and Simulation of CMOS and MOS Integrated Circuits," Bell System Technical Journal, Vol. 57, No. 5, May-June 1978, pp. 1449-1473.
6. L.J. Gallace, H.L. Pujol and G.L. Schnable, "CMOS Reliability," Proc., 27th Electronic Components Conf., May 1977, pp. 496-512.
7. A. Toth and C. Holt, " Automated Data Base-Driven Digital Testing," Computer, January 1974, pp. 13-19.
8. F.P. D'Ambra, M.A. Menezes, H.H. Muller, H.Stopper, and R.C. Yuen, "On-Chip Monitors for System Fault Isolation," in ISSCC Tech. Digest, Feb. 1978, pp. 218-219.
9. D.P. Fulghum, "Automated Self-Test of a Microprocessor system," Autotestcon, 1976, pp. 47-52.

10. K.H. Kim, "Error Detection, Reconfiguration, and Recovery of Distributed Systems," DCS 79, pp. 284-295.

11. D.A. Rennels, "Distributed Fault-Tolerant Computer Systems," IEEE Computers, March 1980, pp. 55-65.

12. E.B. Eichelberger and T.W. Williams, "A Logic Design Structure for LSI Testaility," Proc., 14th Design Automation Conf., June 1977, pp. 462-468.

13. R.A. Cliff, "Acceptable Testing of VLSI Components which Contain Error Correctors," IEEE Trans. on Computers, Vol. C-29, No. 2, February 1980, pp. 125-134.

14. M.J.Y. Williams and J.B. Angell, "Enchancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic," IEEE Trans. on Computers, Vol. C-22, No. 1, January 1973, pp. 46-59.

15. J.P. Hayes, "On Modifying Logic Networks to Improve Their Diagnosability," IEEE Trans. on Computers, Vol. C-23, No. 1, January 1974, pp. 56-62.

16. S. Bozorgui-Nesbat and E.J. McCluskey, " Structured Design for Testability to Eliminate Test Pattern Generation," Proc. of FTCS-10, October 1980, pp. 158-163.

17. H. Eiki, K. Inagaki, and S. Yajima, "Autonomous Testing and its Application to Testable Design of Logic Circuits," Proc. of FTCS-10, October 1980, pp. 173-178.

18. V.K. Agarwal and E. Cerny, "Store and Generate Built-In-Testing Approach," Proc. of FTCS-11, June 1981, pp. 35-40.

19. D.B. Armstrong, "On Finding a Nearly Minimal Set of Fault Detection Test for Combinational Nets," IEEE-TEC, EC-15, Vol. 13, No. 2, February 1966, pp. 1015-1017.

20. J.P. Roth, W.G. Bouricius and P.R. Schneider, "Programmed Algorithms to Computer Test to Detect and Distinguish Between Failures in Logic Circuits," IEEE-TEC, EC-16, October 1967, pp. 567-580.

21. S.L. Wang and A. Avizienis, "The Design of Totally Self-Checking Circuits using Programmable Logic Arrays," FTCS-9, June 1979, pp.173-180.

22. M.W. Seivers, "Computer-Aided Design and Reliability of a General Logic Structure for Custom VLSI," Ph.D. Dissertation, UCLA Computer Science Dept., June 1980.

23. C. Mead and L. Conway, "Introduction to VLSI Systems," Addison-Wesley Publishing Co., 1980, pp.150-154.

24. A.D. Freidman and P.R. Menon, "Theory and Design of Switching Circuits," Computer Science Press, Inc., 1975, pp. 379-428.

25. D.K. Pradhan and J.J. Stiffer, "Error Correcting Codes and Self-Checking Circuits," Computer, Vol. 13, No. 3, March 1980, pp. 27-37.

26. H.C. Godoy, G.B. Franklin and P.S. Bottorff, "Automatic Checking of Logic Design Structures for Compliance with Testability Ground Rules," Proc. 14th Design Automation Conf., June 1977, pp. 469-478.

27. S. DasGupta, R.G. Walther, T.W. Williams and E.B. Eichelberger, "An Enchancement to LSSD and Some Applications of LSSD in Reliability, Availability, and Serviceability," Proc. of FTCS-11, June 1981, pp. 32-34.

28. W.C. Carter, G.R. Putzolu, A.B. Wadia, W.G. Bouricius, D.C. Jessep, E.P. Hsieh and C.J. Tan, "Cost Effectiveness of Self Checking Computer Design," Proc. of FTCS-11, June 1981, pp.117-122.

29. J.B. Rawlings, W. Rosenbluth and R.D. Groves, "VLSI Design for Testability Using IBM Level Sensitive Scan Design (LSSD)," Digest of Papers, GOMAC 80, Nov. 1980, Houston, Texas, pp.347-350.

30. J.R. Lineback, "Self-Testing Processors Cut Cost," Electronics, Vol.54, No.25, December 1981, pp.110-112.

31. T.W. Williams, "Ultilization of A Structured Design for Reliability and Serviceability," Digest of Papers, GOMAC 80, Nov. 1980, Houston, Texas, pp. 441-444.

32. J. Savir, "Syndrome-Testable Design of Combinational Circuits," IEEE Trans. on Computers, Vol.C-29, No.6, June 1980, pp. 442-451.

33. W.H. Pierce, "Failure-Tolerant Computer Design," Academic Press, New York, 1965, Ch. V, pp. 80-110.

34. M.A. Breuer and A.D. Friedman, "Diagnosis & Reliable Design of Digital Systems," Computer Science Press, Maryland, 1976, Ch. 5, pp. 268-269.

APPENDIX A

PARASITIC FLIP FLOP FAULT CAUSES

For a two-input CMOS NOR gate (figure A1), there are five causes that lead to parasitic flip flop faults. Some of these causes make the fault permanent. Some make the fault occur only if certain input patterns are applied to the gate. These five causes are listed as follows [5] :

(1) Input A pull-down transistor (S1) missing or defective;

(2) Input B pull-down transistor (S2) missing or defective;

(3) P-channel transistors (S3 and/or S4) defective;

(4) Broken output wire; and

(5) Missing output contact.

All of these fault causes turn the NOR gate into a D-Flip Flop type circuit. Suppose input A to transistor S1 of figure A1 is broken due to a manufacturing defect, as indicated by an X in the figure. With input pattern A=1 and B=0, transistors S1, S2, and S3 are turned off and S4 is turned on. The output line is neither pulled to VDD nor ground. The result is that this line remains floating for as long as the time required for the fan-out capacitance to discharge to ground. (Note : In some circuit implementation, the fan-out capacitance may discharge to VDD). During the discharging time, the output line of the NOR gate will assume the present logic value stored at the inputs of its fan-out gates. This logic value is in fact the previous output value of the NOR gate prior to the application of the above input pattern to it. Therefore, a "PARASITIC FLIP FLOP" is formed and the NOR gate is

transformed into a memory element (Data Flip Flop) by this circuit fault. The effect of this PFF fault on the self-checking properties of TSC checkers and comparators will be discussed in appendix B.

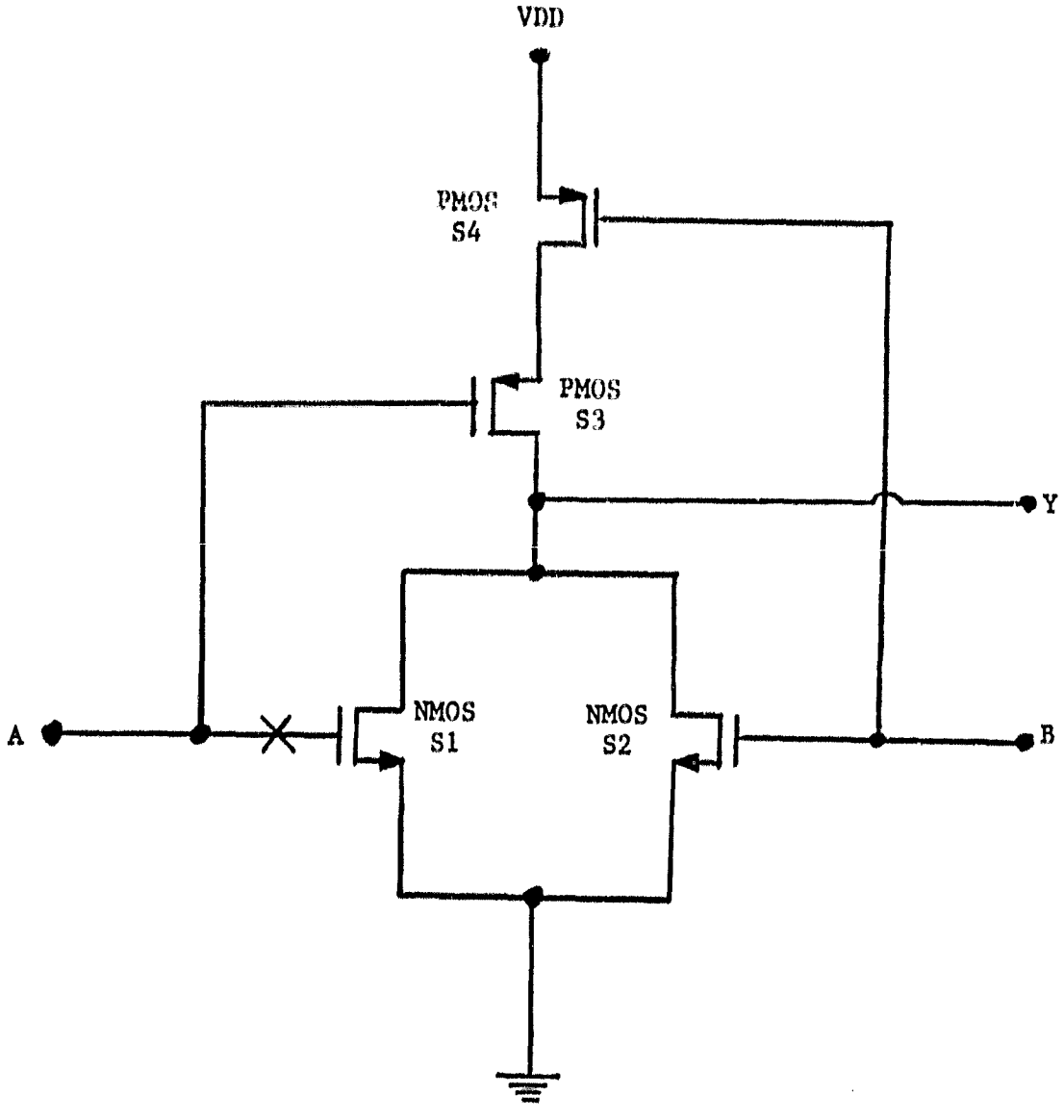


Figure A1 Two-Input CMOS NOR Gate

APPENDIX B

CIRCUIT FAULT EFFECTS ON TSC CIRCUITS

We have shown in subsection 2.1.2 that TSC circuits are totally self-checking for single stuck-at circuit faults. But what are the effects of the other types of faults on the TSC properties of these circuits? We will try to present an analysis of the effects for both of the NMOS and the CMOS cases. In both cases, a single bridging or floating fault does not affect the TSC properties of the circuits. In the case of a bridging fault presence, two neighboring wires short together. This fault will be detected by at least one of the input patterns because in a non-redundant circuit, any neighboring pair of wires must have different logic values for at least one input pattern or else this particular wire pair can be replaced by a single wire; and thus in the faulty situation, the fault can be detected by applying the particular input pattern that would normally distinguish the affected wire pair. In the case of a floating fault presence, the affected wire is floating between logic values. When it floats to the wrong value with respect to a particular input pattern, the fault is detected. If the wire floats to the correct value, the fault is not detected but it is very unlikely that the wire always float to the correct value for all of the legal input patterns. Hence we conclude that the TSC circuits remain totally self-checking in the presence of single stuck-at, single bridging and single floating faults.

In the CMOS case, as we have discussed previously, there is another type of circuit fault called the Parasitic Flip Flop Fault which is input pattern sensitive. (Note : PFF faults can also occur in NMOS circuits if implemented with pass-transistor(s).) For a two-input CMOS NOR gate, we have four distinct input patterns :

Pattern	Input A	Input B
a	0	0
b	0	1
c	1	0
d	1	1

Hence there are 4!=24 possible input pattern orderings (See table in figure B1). It is easy to see that if input A is damaged, the PFF fault is detected only by such input pattern orderings in which input pattern a is followed by input pattern c. Thus there are six input pattern orderings by which the PFF fault is detected. Similarly, if input B is damaged, only the six input pattern orderings in which input pattern a is followed by input pattern b will detect this fault (see table in figure B1). Therefore, only at the most half (12/24) of the input pattern orderings can detect a PFF fault in a two-input CMOS NOR gate and thus any of the TSC circuits implemented with CMOS is <u>not</u> strictly totally self-checking with respect to PFF faults.

In summary, in the presence of all postulated faults, the NMOS implementation of the TSC circuits remains totally self-checking

while the CMOS implementation is input order sensitive with respect to the self-checking property.

ELIMINATION OF PARASITIC FLIP FLOP FAULTS

One simple way to eliminate the undesirable effect of the PFF fault is to connect a pass-transistor between every input of each gate of the circuit and ground. Two complementary system clocks are required. One of them is used to clock the normal system operation while the other is used to discharge charges stored in every input gate capacitor. The drawback of this scheme is that the operating speed of the circuit is decreased by a factor equal to the ratio of the period of the system clock and that of the discharging clock.

INPUT PATTERN	FAULT CAUSED BY	
	S1 input A damaged	S1 input B damaged
1. abcd	-	x
2. abdc	-	x
3. acbd	x	-
4. acdb	x	-
5. adbc	-	-
6. adcb	-	-
7. bacd	x	-
8. badc	-	-
9. bcad	-	-
10. bcda	-	-
11. bdac	x	-
12. bdca	-	-
13. cabd	-	x
14. cadb	-	-
15. cbad	-	-
16. cbda	-	-
17. cdab	-	x
18. cdba	-	-
19. dabc	-	x
20. dacb	x	-
21. dbac	x	-
22. dbca	-	-
23. dcab	-	x
24. dcba	-	-

Figure B1 NOR Gate Parasitic Flip Flop Fault Detection vs. Input Patterns