

City Research Online

City St George's, University of London

Citation: Ludlow, M. P. (1973). The diagnosis of intermittent faults in electronic logic circuits. (Unpublished Doctoral thesis, The City University)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/38426/

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

THE DIAGNOSIS OF INTERMITTENT FAULTS IN ELECTRONIC LOGIC CIRCUITS

MICHAEL P. LUDLOW

THE CITY UNIVERSITY LIBRARY,
ST. JOHN STREET, LONDON, E.C.1.

Thesis Submitted For the Degree Of

Doctor Of Philosophy

THE CITY UNIVERSITY
1973

ABSTRACT

The automation of the design of digital system is discussed with particular reference to production testing and the maintenance of system operational integrity. The need for techniques to diagnose intermittent faults is identified.

The project is divided into three phases: investigation of the use of ordinary solid fault testing techniques for diagnosing intermittent faults; development of a new technique for constructing schedules specifically for diagnosing intermittent faults; and evaluation of the various schedules to determine and compare operational effectiveness under intermittent fault conditions.

Implementation of the various diagnostic schedule construction procedures for solid and intermittent faults is performed using computer programmed simulation routines. The table look-up and the table-driven simulation techniques are selected for this. The Fault-Matrix is used in the procedure for selecting tests for combinational circuits. The test selection procedure for sequential circuits is based on the Sequential Analyser. The types of fault, the form of fault intermittency and the class of circuit considered are limited and restricted to ease implementation problems.

The schedule evaluation is performed by simulation experiments using a two-state first-order Markov model of fault intermittency and a random number generator routine. Results from the evaluation show that the schedule designed specifically for diagnosing intermittent faults is the most effective in practice.

The work described in this thesis was carried out under the supervision of Dr. J. I. Missen in the Department of Applied Physics of The City University.

CONTENTS

	Page No.
ABSTRACT	(i)
CONTENTS	(iii)
1. GENERAL INTRODUCTION - PROJECT FORMULATION	1
1.1 THE AUTOMATION OF DIGITAL SYSTEM DESIGN	2
1.1.1 The Growth Of Design-Automation	2
1.1.2 Using D-A Systems	5
1.1.3 Future Developments In D-A	15
1.2 THE MAINTENANCE OF DIGITAL SYSTEM INTEGRITY	16
1.2.1 Component Malfunctions	16
1.2.2 Reliability Requirements Of Modern Digital Systems	19
1.2.3 Reliability Maintenance Techniques	22
1.2.4 Fault-Tolerance And System Reliability - The Future	32
1.3 INTERMITTENT MALFUNCTIONS IN DIGITAL SYSTEMS	33
1.3.1 Intermittent Faults	33
1.3.2 Present Status Of Intermittent Fault Detection And Location	36
1.3.3 The Diagnosis of Intermittent Faults - A Project	38
2. LOGIC CIRCUIT SIMULATION	41
2.1 INTRODUCTION	42
2.2 LOGIC SIMULATION PROCESSES	43
2.2.1 The General Process	43
2.2.2 Table Look-Up Simulation	45
2.2.3 Boolean Equation Simulation	46

Page No.

2.2.4 Compiled-Code Simulation 47

2.2.5 Table-Driven Simulation 50

2.2.6 A Comparison Of Simulation Processes 53

2.3 TECHNIQUES OF SIMULATOR IMPLEMENTATION 55

2.3.1 Timing 55

2.3.2 Circuit Structure Coding 57

2.3.3 Circuit Element Function Evaluation 62

2.3.4 Logic Circuit Intialization 68

2.3.5 Logic Circuit Simulation Routine 71

2.3.6 The Simulation Routine as a Fortran Subroutine 76

2.4 LOGIC FAULT SIMULATION 78

2.4.1 Circuit Testing 78

2.4.2 Fault Injection 83

2.5 SIMULATION SUBROUTINE - RESULTS, LIMITATIONS AND DEVELOPMENTS 87

2.5.1 Results 87

2.5.2 Limitations 91

2.5.3 Developments 92

3. DIAGNOSIS OF INTERMITTENT FAULTS IN COMBINATIONAL LOGIC CIRCUITS 95

3.1 INTRODUCTION 96

3.2 TEST GENERATION AND SCHEDULE CONSTRUCTION FOR THE DIAGNOSIS OF SOLID FAULTS IN COMBINATIONAL CIRCUITS 100

3.2.1 Test Generation 100

3.2.2 Formation Of Efficient Diagnostic Test Schedules 109

3.2.3 Implementation Of Schedule Formation Process 114

3.2.4 A Circuit Example 118

Page No.

3.3 INTERPRETATION OF SOLID FAULT DIAGNOSTIC TEST SCHEDULE DATA FOR DIAGNOSING INTERMITTENT FAULTS 120

3.3.1 The Effects Of Intermittent Faults On Schedule Data 120

3.3.2 Efficient Calculation Of Diagnostic-Fault-Profiles 126

3.3.3 A Circuit Example 128

3.4 TEST SCHEDULE CONSTRUCTION FOR INTERMITTENT FAULT DIAGNOSIS 134

3.4.1 A Philosophy For Efficient Testing 134

3.4.2 Implementation Of Schedule Formation Process 140

3.4.3 A Circuit Example 145

3.5 DISCUSSION 148

4. DIAGNOSIS OF INTERMITTENT FAULTS IN SEQUENTIAL LOGIC CIRCUITS 151

4.1 INTRODUCTION 152

4.2 TEST GENERATION AND SCHEDULE CONSTRUCTION FOR THE DIAGNOSIS OF SOLID FAULTS IN SEQUENTIAL CIRCUITS 155

4.2.1 Test Generation 155

4.2.2 Sequential Analyser Procedure For Solid Faults 166

4.2.3 Sequential Analyser Implementation 169

4.2.4 A Circuit Example 173

4.3 INTERPRETATION OF SOLID FAULT DIAGNOSTIC SCHEDULE DATA FOR DIAGNOSING INTERMITTENT FAULTS 181

4.3.1 The Effects Of Intermittent Faults On Schedule Data 181

4.3.2 Calculation Of Diagnostic-Fault-Profiles 185

4.3.3 A Circuit Example 189

4.4 TEST SCHEDULE CONSTRUCTION FOR INTERMITTENT FAULT DIAGNOSIS 194

4.4.1 A Philosophy For Efficient Testing 194

4.4.2 Implementation Of Schedule Formation Process 196

4.4.3 A Circuit Example 200

Page No.

4.5 DISCUSSION 205

5. EVALUATION OF INTERMITTENT FAULT DIAGNOSTIC TESTING SCHEDULES 211

5.1 INTRODUCTION 212

5.2 A STATISTICAL MODEL OF INTERMITTENT FAULT OCCURRENCE 214

5.2.1 Markov Processes 214

5.2.2 Breuer's Model 216

5.2.3 Implementation Of The Model 223

5.3 EVALUATION OF COMBINATIONAL CIRCUIT INTERMITTENT FAULT DIAGNOSTIC TESTING SCHEDULES 225

5.3.1 Evaluation Parameters 225

5.3.2 A Procedure For D.F.P. Schedule Evaluation 229

5.3.3 A Procedure For Intermittent Fault Schedule Evaluation 232

5.3.4 Implementation Of The Schedule Evaluation Procedures 233

5.3.5 A Circuit Example 236

5.4 EVALUATION OF SEQUENTIAL CIRCUIT INTERMITTENT FAULT DIAGNOSTIC TESTING SCHEDULES 246

5.4.1 Evaluation Parameters 246

5.4.2 Procedures For Schedule Evaluation 246

5.4.3 Implementation Of The Schedule Evaluation Procedures 247

5.4.4 A Circuit Example 248

5.5 AN EXTENSION TO THE USE OF THE FAULT PROBABILITY MODEL 258

5.6 DISCUSSION 263

Page No.

6. GENERAL DISCUSSION AND CONCLUSIONS 266

6.1 PROJECT FORMULATION 266

6.2 PROJECT RESTRICTIONS 267

6.3 PROJECT IMPLEMENTATION 268

6.4 PROJECT RESULTS 269

6.5 CONCLUSIONS 271

REFERENCES 274

ACKNOWLEDGEMENTS 281

1. GENERAL INTRODUCTION - PROJECT FORMULATION

The problems of designing, constructing and maintaining modern complex digital systems has led many computer and digital electronics manufacturers to utilise the power of computers as a means of reducing these burdens. This is usually termed Design-Automation (D-A). Of particular interest to both manufacturer and user of digital systems is the maintenance of system integrity. Various techniques exist for increasing system reliability. Automatic generation of testing procedures for detecting or locating malfunctions is included in many D-A systems. Although much work has been done in deriving techniques for solid faults, one area that has been neglected is that of generating testing procedures for diagnosing intermittent faults.

1.1 THE AUTOMATION OF DIGITAL SYSTEM DESIGN

1.1.1 The Growth Of Design-Automation

Design-Automation (D-A) techniques originally appeared as in-house aids developed by the computer and electronic logic manufacturers for their own use, to help solve the problems of implementing transistor circuit technology. The manufacturers were faced with the new problems (1) of small components having greater switching speeds, increased packaging densities and interconnections, and inductive noise resulting from the closeness of interconnecting wires. The computer offered a way of reducing these problems by performing the complex and vital tasks of laying out wiring routes, checking on circuit loading and interwire noise and the preparation of wiring diagrams and listings.

The initial success of these aids to design were an incentive for a greater commitment to the development and utilization of other new techniques in the field of computer aided design and Design-Automation. This was reflected by the publication of research papers on this subject from 1956 onwards {(1),(2) give comprehensive bibliographies}.

One of the first published descriptions of a D-A system was by Cray and Kisch (3). This contained several extensions to the normal routeing, layout and drawing features of the (unpublished) in-house systems. The first was the checking of logical equations of circuits for logic and timing errors and contravention of logic element fan-in and fan-out restrictions. The second, and perhaps

the most important extension,was in the use of simulation techniques to test the Boolean equation description of the circuit.

Simulation became, and is still, the most powerful and useful tool of the system designer. The ability to fully test and eliminate design errors from a proposed system without recourse to bread-boarding, and with (near) certainty of success, prior to any production commitment, reduces much of the cost and time waste associated with re-work at production time.

The development and use of standard circuit boards and packages for such functions as adders, registers, decoders etc., allowed D-A to be used in the economic control aspects of the design process. For example, the control of inventories and recording the availability of different types of board and the ordering and restocking in economic quantites was now possible.

The detection and location of faults is essential at all levels of production and operation. This represented a serious drawback in optimising the efficiency of the D-A process, due to the extensive time and effort required for manual formulation of testing schedules. Although these problems have as yet not been fully solved, partial solutions, particularly at the hardware gate level, have been developed through the use of simulation techniques. If a simulator can be used to correctly model a proposed system, sub-system or circuit, it can also model malfunctions within them and so allow the generation of

testing sequences. The main limitation to the automation of such procedures for moderm complex systems lies in the large number of possible fault conditions and in the selection of an efficient testing set from the vast number of possible tests.

The characterisation and modelling of a proposed system is only fully realistic if the characteristics of the particular technology of implementation is taken into account. For example, the delay, fan-in/fan-out, etc., characteristics vary between circuit elements of the D.T.L., T.T.L. and E.C.L. technologies and so affect the design constraints. This information is therefore included in a data-base. Such an information store can also contain data on circuit diagrams, Boolean equations, etc., that are required at different points in the design process and which are perhaps common to other D-A projects.

For any D-A system to be considered really comprehensive it should include sections for dealing with the mechanical engineering problems of mainframe layout and construction, cooling and physical design of components. Techniques exist for computer-aided design of individual circuits (4) (at the transistor, resistor, capacitor level) and this can also be included as a further D-A function. These topics are not of direct concern to the systems and logic design engineers, although they must work within the limitations imposed by these aspects of the overall design process.

The growth and development of D-A was initiated through the recognition of possible economic gains of its

use. Continued growth is somewhat stunted by the security restrictions maintained by manufacturers on their own in-house D-A systems. This protects their investment and so maintains or gains commercial advantage over rivals. This restriction in the dissemination of ideas or inovations limits the fertilization of new concepts and techniques. A further restriction of growth potential of D-A is the use of different computer systems for implementation of the D-A programs by the various manufacturers. Since high level languages are usually inefficient for most D-A functions the use of low level programming has made D-A systems machine dependent (1). Furthermore, D-A systems are constructed to suit the particular circuit technology, design philosophy and needs of the individual manufacturers and as a result are usually incompatible with each other.

An attempt to unify the major concepts of D-A has been made by the recent publication of a book (5) on this subject under the sponsorship of the I.E.E.E. Computer Group Executive Committee. The idea is to review the relevant work on the subject and put it in perspective as a base for others to develop new and better automation techniques.

1.1.2 Using D-A Systems

All new computers or logic systems start off as concepts that are developed to fulfil either a users requirements or a gap in the market that is recognized by the manufacturer as being in need of development. The concept may be initially stated in terms of satisfying the

requirements of miniturization, power, versatility, speed, ruggedness, cost, etc. The formalisation and definition of the concept into operating specifications is usually performed as a joint exercise between user and maufacturer. The full design process is outlined in Fig.1.1.

The next stage in the project development is to evaluate various possible schemes of implementation, working of course within the limitations of current technology and the system specifications. The proposed system can be divided into various high level logical blocks and certain aspects of the system operations are evaluated, such as operational speed, throughput etc., using high level simulation techniques such as those provided by SIMSCRIPT (6) and GPSS (7). Each of the logical blocks, corresponding to such sections as main memory, instruction elements, arithmetic units, interconnection busses etc., can be characterised by parameters relating to response time, capacity and cost. The parameters can be altered until a satisfactory solution is found.

The logical blocks are then sub-divided to a level which an individual designer can cope. The designers are then responsible for the physical design and implementation of their own sub-block. Consistency of the sub-blocks in the overall design specification can be evaluated by a Register-Transfer level data flow simulation (8). The sub-blocks are visualised as various types of interconnected registers. In more comprehensive high level Register Transfer languages it is even possible to simulate

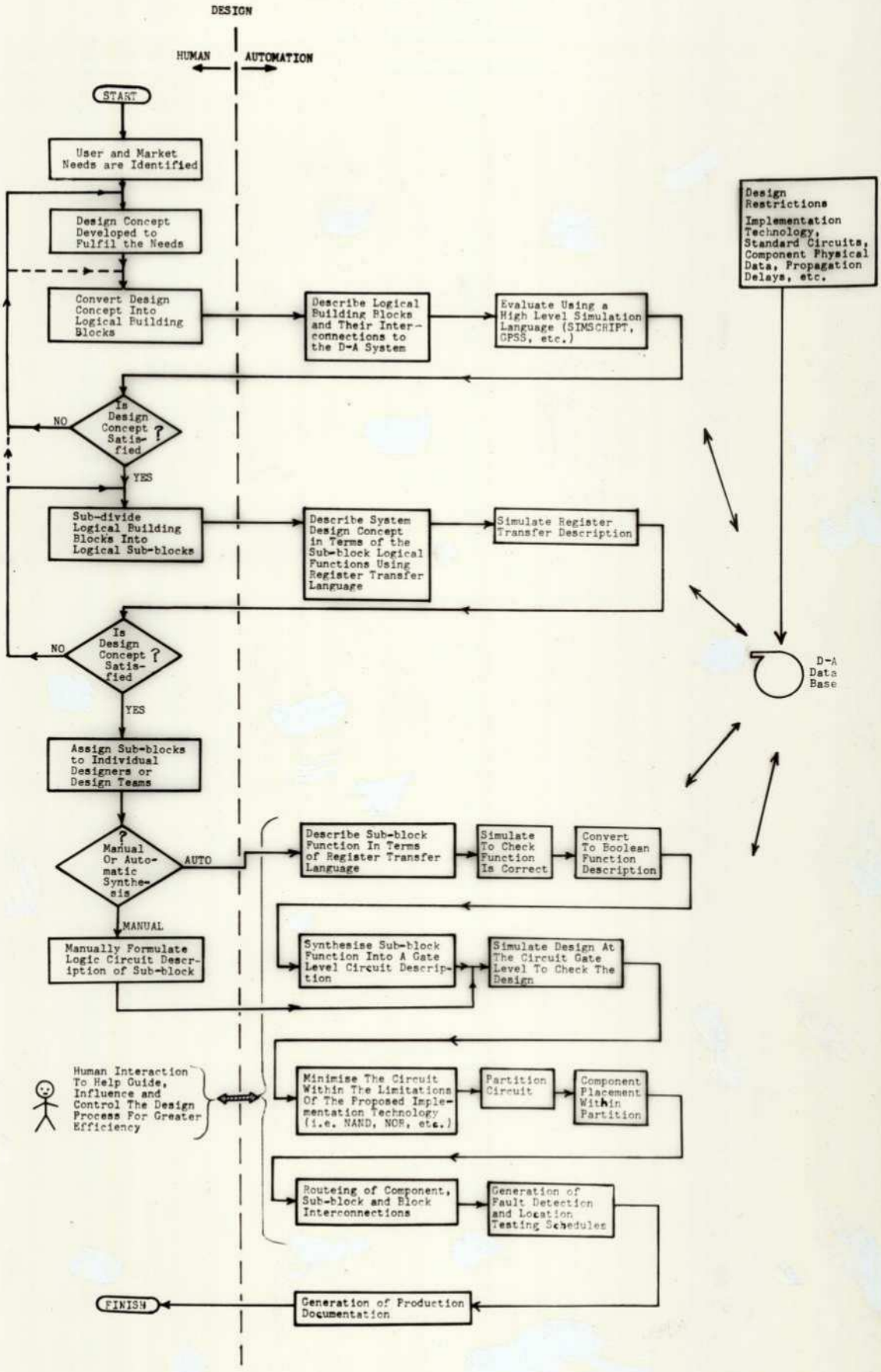


FIG. 1.1. DESIGN-AUTOMATION PROCESS

the execution of actual programs in the proposed system, as has been done for the I.B.M. 360 series of computer using the APL descriptive language (9).

Up to this point the design process has been divided between manual design and automated evaluation by simulation. The designers have been dominant in this man-machine interaction, taking responsibilities for all the decision making and control functions. From here on the D-A system takes on more and more of these responsibilities. However, except perhaps for the very lowest level, the man-machine interaction still plays an important role in optimising the design to meet the stated specifications.

Information on the circuit technology and the types of standardized circuit packages available for use in the system are manually defined and stored in the D-A data-base. This information store plays a most important role throughout the whole project, being the location for storing all the data required and produced at different times by the various D-A functions.

The task of system design now goes on at two levels, some designers working on the total-system aspects and others on the individual sub-blocks. Those on the total-system have the job of producing detailed interconnection specifications between the sub-blocks, defining timing conventions, allocating space within the main frame and finding methods of testing the system (at a high level) once it has been built. Some of these functions can be automated. For example, the production of comprehensive

wiring schedules and diagrams is a relatively simple task. At present most of the testing aspects are performed manually and usually consist of defining special software test routines or sequences of micro-instructions which will be used to detect operational malfunctions in working systems.

The defined function of each sub-block has now to be turned into a solid design, described in terms of Boolean equations or logic block diagrams. This situation can be reached either manually or using automatic aids. The choice of approach is usually determined by the manufacturers philosophy or the designers individual wishes. Manual design follows the normal intuition/experience approach familiar to designers in all fields of work. Automation is provided by detailing the sub-block in a Register-Transfer (or Boolean equation) functional description. This can be converted into Boolean equations or logic diagrams using various synthesising techniques (10),(11) and, perhaps at the same time, produce minimal covers for switching functions (12),(2),(13),(14), satisfying fan-in and fan-out restraints (15) and other relevant restrictions (5).

Once a solid design has been obtained, the designer can use the simulation function of the D-A system to test out the logic circuitry of the sub-block. This is an important phase of the process, particularly if an inituitive design approach has been made. Checks on operational delays, logic errors, hazards etc., can be

performed. Circuit descriptions are stored in the data base for future reference.

The next stage in the design process, having found a possible physical and logical implementation for the sub-blocks, is to determine solutions for fitting this design to the various mechanical and total-system restrictions. The restrictions are imposed by the size, weight, number and size of boards, pin connections, signal propagation times, and the ability to test and maintain the system. This can be divided into the three problems of partitioning, placement and interconnections.

Breuer (2) defines partitioning as dividing a system into disjoint parts (each usually to fit on one board or in one module) so that one or more of the following objectives are met:

(1) each partition contains a required number of elements and uses less than a defined maximum of connectors.

(2) the total number of interconnections between partitions is minimised to reduce wiring effort.

(3) the total number of partitions is minimised (while satisfying (1) and (2) above) so making maximum use of space.

(4) recognising standard subsystems that can be implemented by standard function boards.

(5) minimize the number of different subsystems of (4) above to reduce the number of different types of spares.

(6) signal propagation times are satisfied.

(7) maintenance and testing abilities are maximised.

Not all of these objectives are compatible and any automatic implementation could effectively take objective consideration only a few of them at any one time. For example, good testing ability is not consistent with a requirement for a small number of interconnections between partitions.

Ramamoorthy (16) has developed algorithms suitable for D-A systems for the partitioning of circuits based on directed graphs. This produces good partitioning and optimises the placement of testing points for maximum system testing ability and diagnostic resolution A solution to the problem of satisfying signal propagation times has been developed by Lawler, et al (17), by determining a partition which minimises the maximum delay through the network. The classical partitioning techniques have almost exclusively worked on minimising interconnections by finding maximum conjunction and minimal disjunction between elements to be partitioned, such as the heurstic algorithms described by Kodres (18). Another method, due to Kernighan and Lin (19), works by initially defining the system as two arbitary partitions which have (say) N interconnections. Pairwise interchange of elements between them is initiated in an attempt to minimise N. Kodres (18) has investigated techniques for implementing a circuit partition with a family of standard circuit modules using a minimum number, given that each element of the partition can be implemented by a portion of at least one of the modules. A method of

determing whether or not a new circuit should be included in the standard family has been proposed by Notz, et al, (20).

Another of the mechanical problems is that of placement of elements on boards or modules, modules in subsystems and subsystems in systems so that some restriction, such as total interconnection distance, is satisfied. Hanan and Kurtzberg (21) have published a review of various placement algorithms. Data generated by such routines could be used for controlling automatic assembly machines.

Having formed the circuit partitions, allotted given sets of elements to boards or given boards to subsystems and placed these in appropriate positions within the system, the design engineer is faced wtih the third mechanical problem of forming and routeing the optimum interconnection paths. The solution may be required to satisfy one or more of a number of constraints imposed by such factors as the number of layers on printed circuit boards; the existence of holes for connecting one layer to another; etched line widths; horizontal and vertical polarisation of interconnections; minimum interconnection path lengths and electrical cross-talk interference between parallel connection paths. These problems exist at all system levels from the etched printed circuit connections to the main-frame subsystem interconnections. Akers (22) has comprehensively reviewed this aspect of D-A. Interconnection data, in conjunction with the wiring listings and allocation information stored in the data-base, can

be used for controlling automatic back-plane wiring machines, generating masks for the production of printed circuits and creating wiring schedules and diagrams for manual production use.

The design problems of partitioning, placement and interconnection are usually solved using the D-A system in an interactive mode, with the designer aiding the algorithms by suggesting obvious alternatives or short-cuts. The optimum implementation is not usually found because of the complexity of the problem, and so the designer has to compromise between the use and cost of the D-A computer time and the savings resulting from actually finding a truly optimum solution.

The last major function of the D-A system is the production of testing schedules to enable check-out of the design at various stages of production and for field operation. At the lowest level the correctness of the electrical characteristics of individual components is of vital importance (23) and must be tested prior to insertion on boards. When the components (say AND, OR gates etc.) have been assembled on circuit boards, the functional characteristics of the assembly as a whole becomes important. Testing sequences, either for detection or location of faults, can be generated using simulation techniques. At higher levels, system testing requires special software test routines or micro-program sequences (24). At present these are usually manually generated. Some computer systems are designed to be fault-tolerant

using techniques of hardware redundancy or automatic system reconfiguration. This uses special built-in circuitry or software control routines which are generally functions defined by the original design concepts.

The process of test schedule formation consumes large amounts of computer time which could be wasted if minor system changes have to be implemented (2). Great care has to be taken in choosing the point at which testing schedules are generated in the overall D-A process. Fowler (25) has considered this problem. Diagnosis at the lower systems level will be discussed in detail later.

Much detailed information is required for correctly and efficiently implementing the designed system. Throughout the design process comprehensive documentation must be maintained. Much of this can reside in the D-A data-base in the form of wiring lists, panel and circuit location charts, printed circuit layouts, edge connection lists and circuit and system diagrams. Whenever a design change is made, it must be recorded and reflected through all parts of the system affected by it. Wiring lists are updated, circuit diagrams altered, printed circuits corrected, etc. This process can be automated so that whenever such a change is made new sets of lists and drawings are produced and issued to the designers and others affected by the update.

Finally, when the design process has finished and the original concept has been satisfactorily modelled by the D-A system, comprehensive implementation and testing

documentation is generated ready for the production work to proceed. This leads (hopefully) to the construction of a system that is efficient in terms of design economics, electrical and logical functioning and mechanical layout and construction.

1.1.3 Future Developments In D-A

In the past, commercial secrecy hindered the development of D-A techniques. This was a result of the potential economic advantage a particular D-A system may have given one manufacturer over another. The need to develop certain aspects of D-A has resulted in much greater communication of new ideas and techniques over recent years.

One present development trend is an increase in emphasis on interactive communication with the design process. This allows the computer to aid the designer by enhancing his creative and inductive powers by supplying quick and powerful algorithmic procedures for aspects of the design process that are more easily automated (26). It is this balance of man-machine interaction that, it is hoped, will produce efficient design in all aspects.

Breuer (2) has pointed out a number of areas in D-A which need investigation:

(1) the development of synthesising techniques to cope with the larger functional units associated with L.S.I. technology, such as decoders, multiplexers, registers, etc.

(2) inclusion of circuit reliability and testing ability

features in the synthesising processes.

(3) development of algorithms for efficiently dealing with the partitioning problems of large circuits.

(4) testing of functional units such as adders, shift registers etc.; testing of cellular arrays; generating self-test routines for complete systems; identification of redundant logic and indistinguishable faults; and generation of tests for detecting and locating intermittent faults.

(5) developing D-A techniques for automating the production of system software.

(6) incorporation of design changes or developments with the minimum disruption of the already completed D-A work in the fault diagnostic data, wiring lists,etc.

As mentioned previously, Fowler (25) has considered the problem of the effects of design changes on diagnostic data. Ramamoorthy and Chang (27) have recently investigated techniques for automating testing at the system level for micro-programmed computers using the micro-instructions and micro-operations as the testing source.

1.2 THE MAINTENANCE OF DIGITAL SYSTEM INTEGRITY

1.2.1 Component Malfunctions

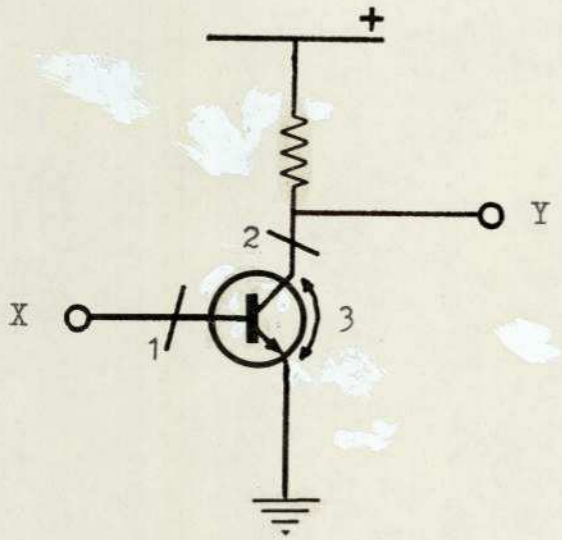
Component malfunctions in digital systems may show themselves as voltage changes, current changes, variations in pulse shapes or circuit delays, or as incorrect logic levels. Much interest has been shown in past research in the detection and diagnosis of logic level faults, it

usually being assumed that techniques are available for dealing with the other types of faults (28),(29).

Logic level faults are generally represented by the input or output of a circuit logic element being stuck-at-0 or stuck-at-1 (s-a-0 or s-a-1). Such faults are considered to be logical, that is, the malfunctioning circuit still acts in a logical (but different) and predictable manner. As an example of a logical fault consider the simple logic inverter circuit in Fig.1.2. Y_o represents the circuit output under normal conditions; Y_1 and Y_2 are the outputs resulting from the base and collector being open-circuit, giving Y s-a-1 (assuming +v = 1, 0v = 0) independant of the input state X; and Y_3 corresponds to a short circuit between collector and emitter, giving Y s-a-0 independant of input state X.

The physical causes of such faults arise due to one (or a combination) of the three factors 1) component ageing; 2) manufacturing defect; or 3) design error, (30), (23). Ageing effects can result in component failures a) in storage prior to insertion into the system; b) while the system is in storage prior to operation; and c) when the system is in operation. Such faults are caused typically by changes in component parameters due to long term chemical (oxidation) and electrolytic reactions.

During the manufacturing process typical faults are due to open or cracked bonds in integrated circuit or transistor packages, high resistance interconnections (dry joints) and short circuits caused by scratches or holes in component substrates. At the macroscopic level in the



Fault Condition
Truth Table

X	Y_0	Y_1	Y_2	Y_3
0	1	1	1	0
1	0	1	1	0

1 = + volts
0 = 0 volts

FIG. 1.2. LOGIC LEVEL INVERTING CIRCUIT

manufacturing process, for example component assembly, faults have been identified (23) as resulting from the use of incorrect or non-functioning components (representing a failure in the goods-in screening process); mis-orientation of components caused perhaps by incorrect markings or operator error; mechanical or thermal over-stressing of components by the manufacturing process, indicating perhaps that the process has not been designed or specified correctly.

Failure to adequately assign tolerances and define the conditions under which components must work, at the design stage, can result in the appearance of faults during system operation. An example would be choosing a component with temperature tolerances $0^{o}C$-$50^{o}C$ for equipment to be used under Arctic conditions. Similarly, a given packaging technique for integrated circuits should be evaluated for the effects of a saline atmosphere if it is to be included in marine equipment. Manufacturers should also recognise that the production line is different to the design workshop environment and that components could be maltreated unless the production and manufacturing processes are adequately designed to cope with the physical and electrical limitations of the components.

1.2.2 Reliability Requirements Of Modern Digital Systems

Before a digital system is delivered to a user, the manufacturer must be able to guarantee that it will work correctly. The most efficient way of meeting this guarantee is to detect and correct faulty components or

subsystems and design errors prior to insertion into a more complex assembly on the production line. This is because the total cost of the failure and the diagnostic complexity increase with the complexity of the system (23). The cost of a failure at a given level of production complexity can be listed as the sum of:

1) Cost of replacement component.
2) Cost of diagnosis.
3) Cost of rectification work.
4) Cost of retest and check.
5) Cost of production delay.
6) Cost of any design changes that may be needed in wiring, circuit diagrams, wiring lists, component schedules, etc.

At high levels this total can be many hundreds of times greater than the actual component cost. It therefore pays to catch faults at low levels.

Once a system has been delivered to a user for operational use certain reliability guarantees must be maintained. From operational considerations modern digital systems can roughly be divided into two classes:

(i) off-line, batch processing - for general scientific, mathematical and business computation.

(ii) on-line, real-time processing - for industrial process control, weapon/plane/ship/rocket guidance and control, telecommunication switching, etc.

In either case the occurrence of a component malfunction is undesirable and the system manufacturer must supply

methods for minimising the effects of such occurrences and for maximising reliability. In the class (i) system, normal operation ceases when a fault has been detected and the diagnostic testing procedures are applied, either from another testing computer or through a self-test technique, until the fault or faulty module is found and rectified. The manufacturer must make sure that the system has built-in reliability and should supply efficient procedures for locating malfunctions. Those computer jobs corrupted by the fault are then reprocessed. The cost of the fault is the sum of:

1) Cost of time lost.
2) Cost of time for diagnosis.
3) Cost of replacement part.
4) Cost of rectification work.
5) Cost of job re-runs.

Unless suitable protection is supplied a component malfunction could be disasterous to a class (ii) system. For example, a rocket could go out of control or take an incorrect course, etc. For this reason redundancy and self-repair techniques are used to mask out the effects of faults and permit the maintenance of system operating integrity. The total cost of maintaining this integrity is the sum of:

1) Cost of extra design work prior to system manufacture.
2) Cost of extra components and specialised circuitry required for redundancy.

3) Cost of lost performance capabilities if system degrades itself by reconfiguration (to switch out malfunctioning unit).

4) cost of time to physically repair a fault (if this is possible, as in a telecommunications computer).

1) and 2) above represent capital costs and are included in the total system cost at purchase time.

1.2.3 Reliability Maintenance Techniques

Many techniques have been developed to aid the maintenance of digital system reliability. The term *reliability* has different meanings for different system requirements. In one case it might mean the detection, location and repair of malfunctions within certain time limits; in another case it might mean guaranteeing the system for complete integrity, through the use of some fault masking technique, over a defined period of operating time. A general model for reliability maintenance (28) is outlined in Fig.1.3. The model applies to production or pre-production component or sub-system testing as well as operational system testing.

The implementation of the reliability model depends on the particular requirements of the subject system and could be ;

(i) A special-purpose testing computer connected to the system. The test data is applied as a series of electrical stimulii at the inputs; the resulting outputs are detected and recorded. The observation

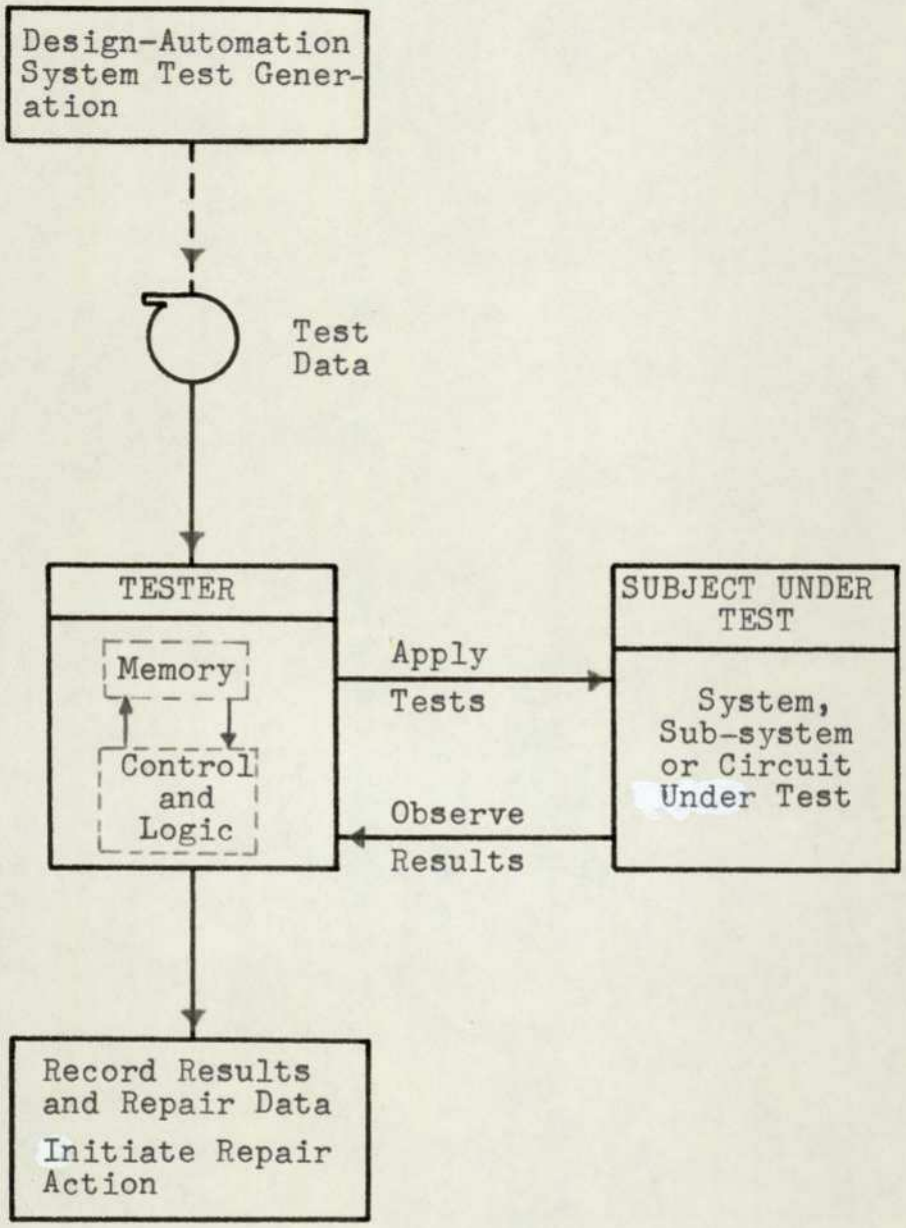


FIG. 1.3. GENERAL MODEL OF A SYSTEM FOR RELIABILITY MAINTENANCE

of a particular series of output responses indicates a particular fault. When the fault is located it is manually repaired.

(ii) A special-purpose production line testing computer which can be connected to various components, circuits and subsystems. The testing data is applied as in (i) above and can be used for fault location at various stages of manufacture.

(iii) Included as special hardware circuitry or software programs as part of a self-checking system. When a fault is located the system reports the diagnosis and it is manually repaired.

(iv) The same as (iii) above, but with the effects of faults masked out by either hardware redundancy, software programming or reconfiguration of the system.

The distinction between the different functional sections of the maintenance model becomes blurred in (iii) and (iv), the testing mechanism being taken over by the subject system itself or being inherent in the designed hardware, with the test data included as hardware functions (perhaps by calling a series of micro-instructions) or stored data.

Various methods have been developed for generating, selecting and verifying those tests which are sensitive to the occurrence of malfunctions (of the s-a-1, s-a-0 type) in digital systems, such as:

1) One-dimensional path sensitising (31).

2) D-Algorithm (32).

3) Equivalent Normal Form (31).

4) Boolean Difference (33).

5) Seshu's Heuristics (34).

The formation of a series of these tests into efficient diagnostic test schedules has been considered by many researchers (35),(36),(37),(38),(39),(40),(41). These topics will be considered in detail at a later point in this thesis. The general techniques of maintaining system reliability will be outlined here.

The first area to be discussed is that of fault-masking through redundancy. Here, the effects of one or more faults (up to some defined limit) are instantaneously masked out by additional circuitry that is itself part of the operating system. The most simple scheme to improve reliability is that of triplication and voting, Fig.1.4. The function z = F(x) is replicated three times and a vote taken on the three answers (z', z" and z'") to find the majority result using the voting circuit M. If one F unit has malfunctioned the true result (z) is still obtained due to a majority of correctly working circuits. Although the mean time between failures for such a system is smaller than in the non-redundant system because of the larger number of components, the probability of survival of the triplicated system is greater for short periods of time. The majority vote circuit is an obvious trouble point in the system since it cannot be allowed to fail if reliability is to be maintained.

More sophisticated triplication schemes have been developed where the function F is divided into n sub-

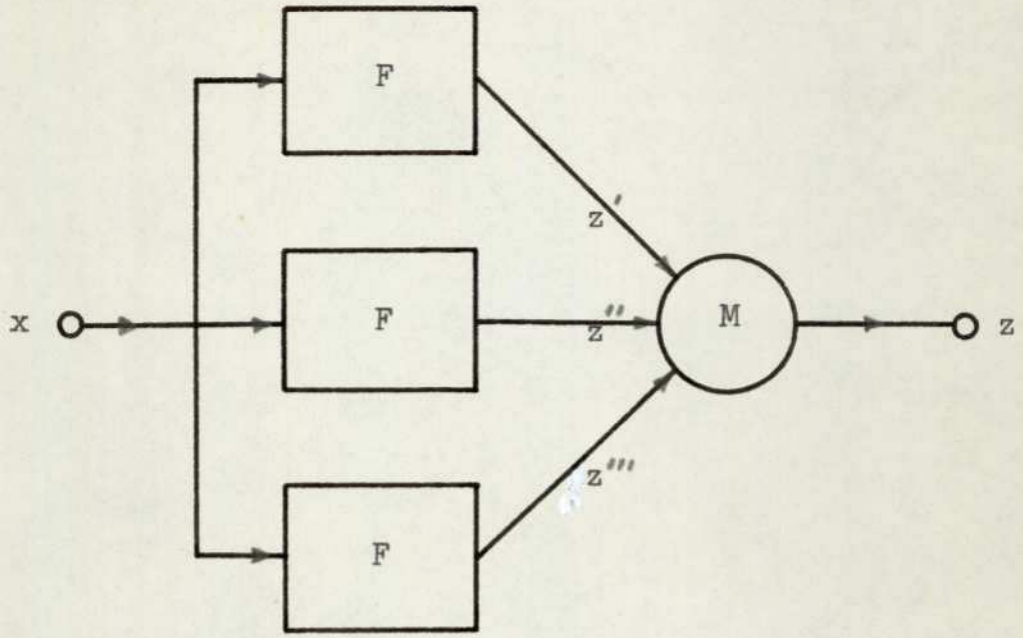


FIG. 1.4. TRIPLICATION AND VOTING SCHEME FOR INCREASED RELIABILITY

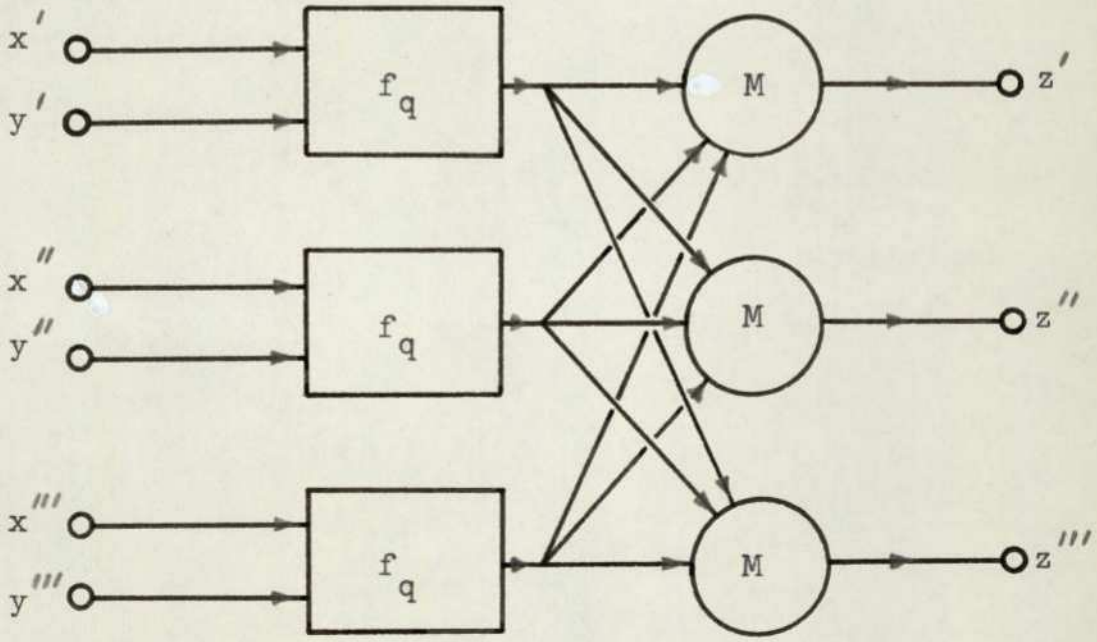


FIG. 1.5. SUB-FUNCTION TRIPLICATION AND VOTING SCHEME FOR INCREASED RELIABILITY

functions, f_i, i = 1,2,.....,n, and where each f_i is itself triplicated (42). Consider the q^{th} sub-function, f_q, of a function F, Fig.1.5. The voting circuit itself is triplicated and so any fault in it or in any f_q will be masked. The three output results z', z" and z'" form the triplicated inputs needed to drive any other connected sub-function. The majority gates in this multiplexing scheme act as correcting functions and re-generate reliable information for transference to the next sub-function.

Another technique of redundancy is that of *quadded* logic (43). In the redundant construction of the system logic elements appear in quadruplicate, malfunctions being corrected by the subsequent level of logic. The basic concepts can be illustrated by the AND gate elements shown in Fig.1.6(a) and (b). The faults that appear at the inputs of circuit (a) can be divided into the two classes of:

(i) those which result in erroneous 1 outputs.

(ii) those which result in erroneous 0 outputs.

Consider circuit (a) modified to the form shown in (b), where x_1 and x_2 normally have the same value (except perhaps under a fault condition) but are obtained from independant sources, and the same with y_1 and y_2. In this form no single input fault can result in an incorrect 1 output appearing at z. The circuit (b) therefore corrects for class (ii) types of faults, but not for class (1). The implementation of a similar extension of redundancy to include class (i) faults results in a fourfold increase in the number of logic elements over the non-redundant

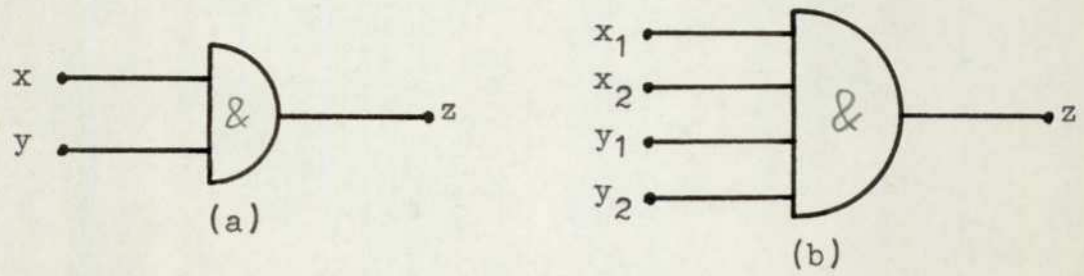


FIG. 1.6. 'QUADDED' LOGIC EXAMPLE

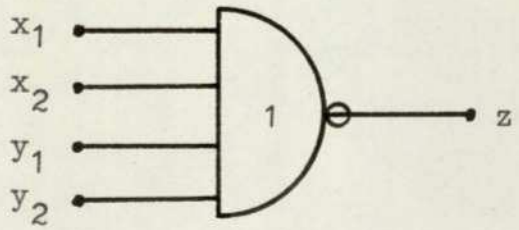


FIG. 1.7. 'RADIAL' LOGIC EXAMPLE

implementation.

Radial Logic (44) is another redundancy technique that can be used for masking most single element malfunctions. It utilises the fault masking properties of the NOR (or NAND) logic element having duplicated inputs, Fig.1.7. Only faulty 0 outputs can be produced by this implementation, and it is possible to correct for these at the next level of logic.

A further fault-masking technique is the use of error correcting codes. An m-input, n-output circuit can be designed to give correct outputs even in the presence of single faults by adding a certain number of check data bits for coding the error correction data, and a reliable decoder. Efficient techniques have been developed by Armstrong (45) and Ray-Chaudhuri (46) for producing minimally-redundant coding.

System survival probability gains of an order of magnitude have been predicted for techniques of dynamic redundancy. This is self-checking and repair through a process of fault detection and then corrective action by automatic replacement of failed sub-system modules, or by reconfiguration of the system (47). Two examples of systems that are highly fault-tolerant through the use of such techniques are the Bell System No.1 E.S.S. (Electronic Switching System) (48) and the STAR (Self-Testing And Repair) computer (49) developed by the Jet Propulsion Laboratory, California.

No.1 E.S.S. had the design objective of keeping the

total system *down-time* in the 40 year expected system life to less than two hours (although this did not preclude the loss of a certain percentage of calls or a reduction in service efficiency at certain times). Apart from the special techniques of fault-masking and dynamic redundancy, the system was made as reliable as possible in a conventional sense by using long life components and conservative tolerancing and designing. The special techniques consist of duplication of vital parts of the system, hardware and software checking of the various units, plus the ability to switch out faulty units and switch in functioning spares. In units such as data busses and memories, error-correcting codes are used. The central control unit is duplicated. Both work simultaneously, only one processes the real telephone calls, the other shadows its operations and continually compares the results through special check circuits. If a mismatch occurs processing temporarily ceases and both units proceed to check themselves and each other by calling on special testing sequences stored in memory. Depending on the result of the diagnostic procedure, the system automatically reconfigures itself and normal operations are continued. The malfunctioning unit can then be repaired without interfering with operations. Preventative maintenance is performed by periodically testing the checking circuits and seldom used parts of the system circuitry.

The STAR computer was developed to investigate reliability design techniques for systems to be used in

long, unmanned, deep space missions. The features of the system giving high fault tolerance are:

(i) The system is divided into a number of functional (and easily replaceable) units, each with a set of stand-bys. These spares are permanently connected to the system through the information busses, but are unpowered.

(ii) All machine words are encoded in error detecting codes. Also, fault detection is performed by other special error detection circuitry.

(iii) The *hard-core* (that part which must not fail) of the computer is a reconfiguration unit which monitors the system operations and the fault status of the various units. It switches in spares appropriately. This unit is duplicated to increase reliability.

(iv) A malfunctioning unit is replaced by power-switching. The bad unit is switched off and a spare is turned on. The use of unpowered spares reduces the effects of operational ageing.

(v) The effects of intermittent faults are minimised by repetition of program segments. Software can also aid in masking faults by recognising the malfunction and altering the computation appropriately.

(vi) The use of spares removes the problems of synchronization associated with parallel duplicated systems. Also the system can survive until all spares have been used.

To terminate this brief review of reliability

maintenance of electronic systems it is worth mentioning a few techniques of designing circuits for easy fault diagnosis. Friedman and Menon (29) have looked at the problem of designing combinational circuits such that the test set is small. Furthermore, they have considered designing such circuits so that faults are easily locatable (which is not the case for many circuit faults). Sequential circuit diagnosis is a more difficult problem, and Kohavi and Lavallee (50) have investigated procedures for adding outputs to any sequential circuit to make it definitely diagnosable through identification of distinguishing sequences. The idea of converting sequential into combinational circuits through the use of additional hardware, to simplify the diagnostic process, has been investigated by Hall (51) and Friedman and Menon (52).

The concern of manufacturers and users of digital systems for the maintenance of operational integrity is highlighted by the introduction of an annual International Symposium on Fault-Tolerant Computing in 1971 (53). The interest in this topic is further emphasised by the publication of two books on system testing and diagnosis in 1970 (28) and 1971 (29).

1.2.4 Fault-Tolerance And System Reliability - The Future

One of the main objectives of the STAR computer project was to investigate techniques for obtaining highly reliable systems. The need for such a project emphasises the lack of a general theory of fault-tolerance for understanding the degree of self-checking and self-repair that is

attainable through a particular system design.

Areas in need of investigating are:

(i) The amount of software back-up required for use in the diagnostic process; when and at what point programs should be re-run after fault detection to minimise the spread of corrupted data.

(ii) The method of restarting a system after a near-catastrophic failure, and the amount of data needed for this.

(iii) Methods of software assisted utilization of failed units to perform useful and functional processes.

(iv) Better circuit design, tolerancing and component design to increase inherent reliability of the system hard-core areas.

(v) Effective techniques for estimating reliability probabilities for particular system designs.

1.3 INTERMITTENT MALFUNCTIONS IN DIGITAL SYSTEMS

1.3.1 Intermittent Faults

Intermittent faults are probably the most troublesome type of fault that test engineers have to deal with (23). Even when the existence of such a fault has been noted diagnosis may be almost impossible unless the fault condition can be reproduced. In reviewing the field of fault diagnosis many authors have pointed to the problem of fault intermittency as an area in need of investigation (23),(28),(29),(30),(41),(54),(55).

Ball and Hardie (56), investigating the effects and detection of intermittent failures in digital systems, have determined that a large proportion of field failures in aerospace digital systems tend to be intermittent in nature. They claim that this is a result of the efficiency of current production and operational diagnostic screening techniques at detecting and locating the solid class of fault. (Note: a solid fault is defined as one which having occurred, remains until repaired.) The intermittent faults slip through the manufacturing screening process and remain undetected, reappearing later to plague operation in the field. In reviewing several computer systems, Ball and Hardie state that their investigations have indicated that:

(i) intermittent faults absorb a greater part of testing and maintenance costs in modern computers.

(ii) intermittent faults represent 30% of pre-delivery failures and 90% of operational failures.

Intermittent faults are the result of physical defects that stimulate marginal operation of components or interconnections. The defect may be purely mechanical in nature, for example:

(i) A poorly bonded component connection that is sensitive to thermal or mechanical stressing (heat cycling, vibration) resulting in an intermittent closed/open circuit connection.

(ii) A printed circuit edge connector that has poor mechanical contacts, perhaps due to dirt or a manufacturing defect.

(iii) Too close layout of printed circuit lines or metallisation layers in integrated circuits; or wires in cables running parallel for too long a distance - leading to signal pick-up.

(iv) Certain thermal conditions can cause the growth of whiskers on aluminium layers in aluminium/silica and silicon structures, resulting in the formation of temporary leakage paths through oxide layers (57),

Marginal operation can also result from electrical variations caused by component over-sensitivity to such factors as:

(1) temperature.

(2) humidity.

(3) magnetic field and radiation variations.

(4) electrical loading effects.

(5) normal power supply variations.

Chang, et al (28), point out that component marginal operation may eventually turn into a solid malfunction through a gradual increase in the component defect. This would then allow the fault to be located by conventional solid fault diagnostic techniques. However, the time for this to occur (if it ever does) is unpredictable and so does not offer a satisfactory solution to the problems of diagnosing such faults.

1.3.2 Present Status Of Intermittent Fault Detection And Location

Three possible solutions to the problems of intermittent faults in digital systems can be identified:

(i) The design of techniques for constructing testing schedules specifically for the detection and location of intermittent faults.

(ii) The inducement of intermittent faults to appear as solid malfunctions through the use of some physical stimulus.

(iii) Designing systems to be insensitive to intermittent faults.

Solution (i) has been identified by Ball and Hardie (56) as an area in urgent need of development. The recently published work of Breuer (58) represents the first real steps in this area, at least as far as fault detection is concerned. The technique is based on finding minimal length test sequences which can detect an intermittent fault, given that it occurs in a pattern that is related to some statistical property of the fault itself. The probability of a given pattern of fault occurrence can be determined from a particular model of fault intermittency and the various possible patterns can be evaluated in their most probable order. The test sequences can be generated and evaluated for efficient detection of the faults by use of the D-Algorithm (59). The Breuer model of fault intermittency will be described later in this thesis.

The obvious limitations of this technique lie in the

large number of different fault occurrence patterns that exist and the fact that some probability data on the fault is required for efficient test sequence evaluation. For a fault detection schedule of n tests the number of different fault patterns is 2^n (given that the fault is either present or absent). As far as this author is aware no method exists for accurately predicting the probability statistics associated with such faults. Without this data Breuer's method will not be able to guarantee the construction of efficient testing sequences.

It has been suggested (60) that the repeated cycling of the test set until the intermittent fault is *caught* can be used for diagnosing such a fault. This is investigated later in the thesis.

Solution (ii), the solidification of the intermittent fault, is commonly used in practice. The faults are forced to appear by the use of vibrational and thermal stimuli. The range of techniques varies from the use of special rigs designed specifically for these purposes (56) to the simple stimulus of manually stressing plug-in circuit boards by tapping with a screwdriver (23). This technique has no real scientific basis and represents an area in need of further development.

Solution (iii) circumvents the effects of an intermittent fault by designing the system to be insensitive to its occurrence. The techniques of redundancy, using triplicated logic and voting or dynamic replacement and reconfiguration, have proved effective in increasing

system fault-tolerance. The process of repeating program segements used in systems such as No.1 E.S.S. and STAR also allows intermittent faults to be masked out. It is perhaps the development of these techniques that will in the long term solve the problems of intermittency (56).

1.3.3 The Diagnosis Of Intermittent Faults - A Project

Ball and Hardie (56) have investigated the effects of intermittent faults on digital systems and have shown the need for techniques to be developed for constructing detection and location testing schedules for such faults. Breuer (58) has taken the first steps in this direction by investigating a method of constructing detection schedules, but he points out the need for the development of location schedules. This need forms the justification for the research project that is presented in this thesis.

Such a project would normally form a new program segment in a D-A system. However, the task of building such a system is beyond the capabilities of one person working, without group support, to a limited timetable. For these reasons the project was taken out of the context of a real D-A system and viewed as a problem on its own. This, in fact, represents a considerable simplification because of the removal of the restrictions of having to communicate with and between the various parts of the D-A system and the complex format and control problems associated with such processes.

Since this investigation is research rather than development oriented, consideration of particular logic

systems, types of circuitry and technology, or methods of implementation, is ruled out. It is possible that these limitations may make the new techniques developed in the project seem rather abstract, but it is hoped that the presentation will be illustrative enough to be useful in a realistic way.

A general statement of intent for the project can be written as:

> 'the investigation of techniques for deriving efficient testing schedules for the diagnosis of intermittent faults in digital systems.'

More specifically this could be expanded to:

(i) Investigation of the effectiveness of present solid fault schedules at diagnosing intermittent faults, and the derivation of techniques for efficiently utlizing such diagnostic data for both combinational and sequential circuits.

(ii) Development of efficient testing schedules which are designed specifically for diagnosing intermittent faults in both combinational and sequential circuits.

(iii) Evaluation of the performance of the schedules described in (i) and (ii) above.

The thesis is divided into six major sections:

SECTION 1 - This places the project in perspective by reviewing the field of Design-Automation and the various techniques of maintaining system integrity.

SECTION 2 - Reviews the various techniques of logic circuit simulation and describes the program developed for this work, including a fault injection capability.

SECTION 3 - Is concerned with the diagnosis of intermittent faults in combinational circuits. It starts by reviewing the techniques developed in the past for diagnosing solid faults and goes on to examine the possibilities of using such techniques for intermittent faults. A new technique of schedule construction for intermittent faults is also described.

SECTION 4 - Sequential circuits are examined in a similar way to Section 3.

SECTION 5 - The various schedule construction techniques are evaluated using a special fault model with the simulation and fault injection process. These results are compared against the result produced by the re-cycled test set. The use of the intermittent fault model is extended for calculating fault probability distributions.

SECTION 6 - Discussion and conclusions on the work presented in the thesis together with a consideration of future developments.

2. LOGIC CIRCUIT SIMULATION

Simulation provides a technique for evaluating design proposals for complex and costly electronic logic systems prior to a manufacturing commitment. It can also be used to aid the construction of diagnostic test procedures for production and operational testing. Simulation can take place at different levels, overall system level; register level; logic gate level; and basic component level (transistor, resistor, etc.). The gate level simulation is of particular interest for this project. Four simulation techniques can be identified:

(i) Table look-up.

(ii) Boolean equation.

(iii) Compiled-code.

(iv) Table-driven.

Methods (i) and (iv) were used in this project. A method of permitting fault injection into a circuit under simulation was developed.

2.1 INTRODUCTION

The elimination of design errors and confirmation of satisfactory system operation to within specification is of great economic importance to digital system manufacturers. The cost of re-design and re-work after a commitment to production can be enormous, especially in systems of great complexity and high packing density. Simulation offers a solution to these problems.

Simulation consists of exercising a model of a proposed system with sets of input stimulii to examine its behaviour and reactions. Information from such a model can be used for evaluating various system design alternatives, verifying that operations are to within specification, obtaining operational reference data to aid in acceptance testing at production, and, most important for this project, generating circuit diagnostic test schedules.

Simulation can be performed at various system levels (see Section 1.1.2). It is the logic gate level of simulation that is of interest in this project and discussion will be limited to this aspect.

The rest of this section is divided as follows. Section 2.2 describes the various identifiable phases common to all types of simulation process. The various processes are described and evaluated. Section 2.3 considers the problems of time mechanization and the implementation of the simulation model chosen for this project. It goes on to describe the development of the overall simulation process, the flow diagrams of the simulation routines, and implementation of the process as a

computer program subroutine. Techniques for injecting solid and intermittent faults in the simulation model are described in Section 2.4. The limitations and possible developments of the implemented simulation process are discussed in Section 2.5.

2.2 LOGIC SIMULATION PROCESSES

2.2.1 The General Process

In general, two distinct phases can be identified in the simulation process (61):

(i) system logic description,

(ii) simulation operation and control.

This is illustrated in Fig.2.1. The circuit to be simulated is encoded in some form, perhaps as a set of Boolean equations or as a list of gates and interconnections, and is input to the simulation medium (i.e. the computer program and associated storage). Backed by a data reference file each function or logic element can be modelled.

The combined description of elements (or functions) and interconnections is exercised by the simulation control routine to determine the output response to some set of specified input stimulii. The initial logic states and any special conditions, such as the presence of faults or the requirement of observation at particular points in the circuit, can be specified and implemented through the control section.

Particular simulation processes are now described.

Input of circuit
connection
description

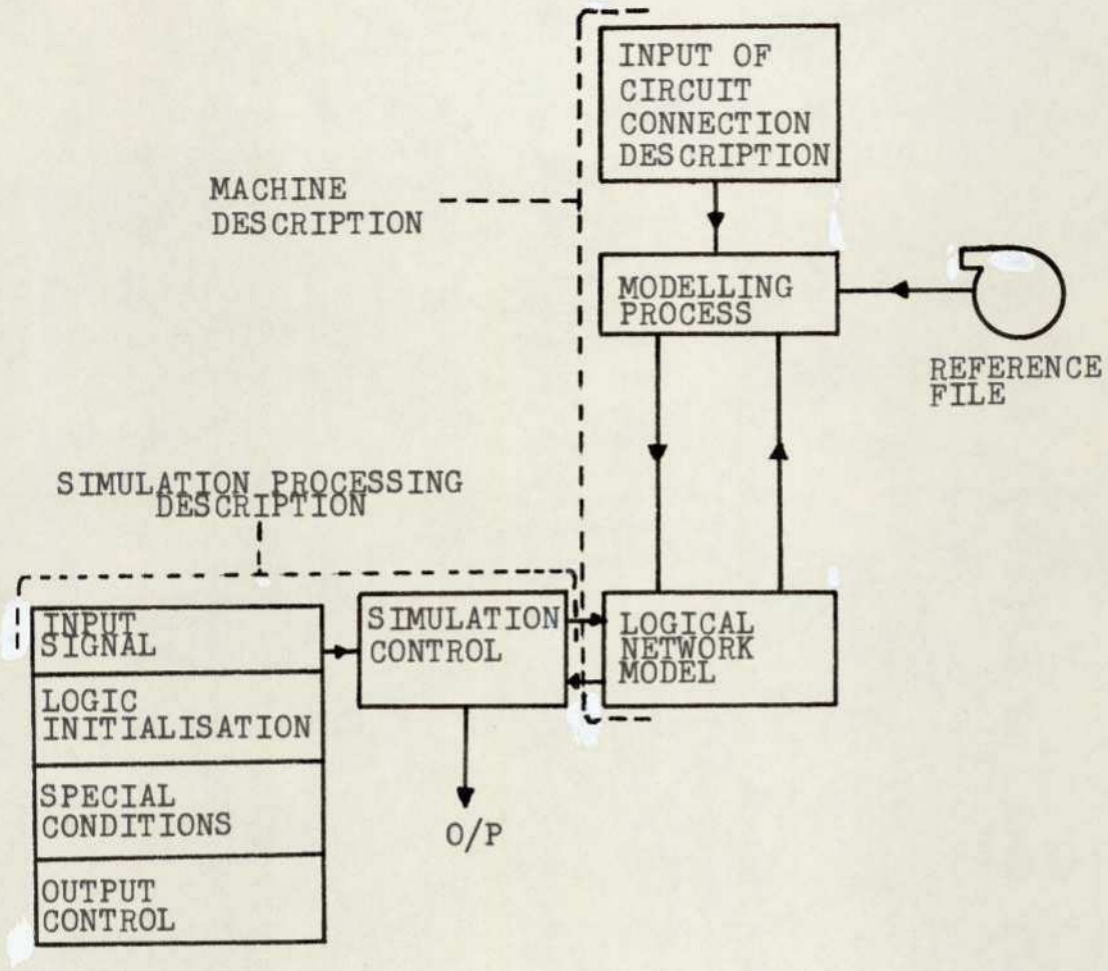
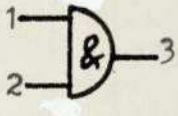


FIG. 2.1. THE SIMULATION PROCESS



	I/P		O/P
	1	2	3
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	1

(a)

	I/P	O/P
0	0	0
1	1	0
2	2	0
3	3	1

(b)

FIG. 2.2. AND GATE TRUTH TABLES

Input of circuit
connection
description

2.2.2 Table Look-up Simulation

This simulation process is by far the easiest and quickest, but suffers from several major drawbacks that limit its use.

Consider the truth table of a two input AND gate, Fig.2.2(a). Each row, 0 to 3, corresponds to one of the four possible input tests to the circuit. The three columns contain the inputs to connections 1 and 2 and the output obtained from 3, respectively. The circuit response to input test number i is obtained by accessing the i^{th} row element of the output column. The circuit response to a series of input tests is obtained in a similar manner by a series of table look-up operations.

More complicated circuits with n inputs and m outputs can be represented in a similar manner, with one row for each possible input test and n+m columns corresponding to each input and output. This produces a rather large matrix, even for a fairly small circuit, of 2^n x (n+m) elements. If the binary values of the inputs and outputs are coded in decimal form, Fig.2.2(b), only two columns are needed.

Referring to the general simulation process described in Fig.2.1, this method represents an extensive simplication resulting from removal of any circuit structural considerations.

Table look-up has several major drawbacks:

(a) It requires large amounts of storage since the number of tests is proportional to 2^n.

(b) The process only resolves the circuit input and output responses.

(c) From (b) it follows that there is a lack of timing information on the delays between application of particular inputs and the appearance of the resulting output.

(d) It also follows from (b) that it is not possible to inject faults into the simulated circuit.

(e) It is limited, essentially, to dealing with combinational circuits where consideration of circuit states and state transition is not required.

(f) The table has to be generated by hand or by using one of the other processes described later.

The problem of fault injection can be eliminated if p extra output columns are generated, each one containing the output response in the presence of p circuit fault conditions. The storage requirements increase from 2×2^n to $(2 + p) \times 2^n$.

2.2.3 Boolean Equation Simulation

In general Boolean equations contain little information of the particular technique of implementation or the delay properties of the actual circuit elements. The process of simulation involves evaluation of the Boolean equation of a circuit, in the presence of defined input variables, using normal Boolean logical calculation methods. The equations can be re-arranged and simplified to increase the process speed. This can be done by, for example, representing them in a simplest-sum-of-products form (62).

The simple circuit in Fig.2.3 could be simulated as follows:

(a) Set up input variables A,B,C,D and E to required logic levels.

(b) Evaluate F, $F = \overline{A + B + C}$.

(c) Evaluate G, $G = D + E + F$.

(d) Circuit output result is defined by the values of F and G.

Increased efficiency of computation can be obtained in the above example if, when A = 1, advantage is taken of the fact that F = 0 independent of B and C. Similarly, if any of the variables A,B or C = 1, F = 0 and the others need not be examined.

The main disadvantages of this particular simulation technique are:

(a) The logic equation rather than the actual circuit is being simulated.

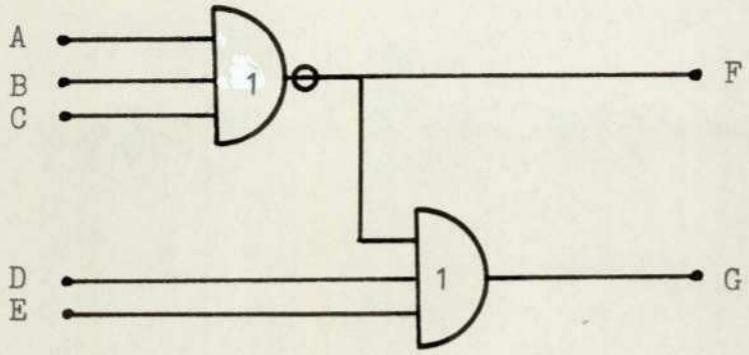
(b) Timing information is lacking.

(c) Complicated circuits can have excessively long Boolean equations resulting in very long computation times.

Techniques do exist to cut down excessive computation by ordering the simulation procedure in tree structures so that expressions containing several functions of the same input variable are arranged in a way that each is evaluated once only (61).

2.2.4 Compiled-Code Simulation

A logic circuit can be represented within a computer program as a series of interconnected subroutines. Each



$$F = \overline{A + B + C}$$

$$G = D + E + F$$

FIG. 2.3. BOOLEAN EQUATION SIMULATION EXAMPLE

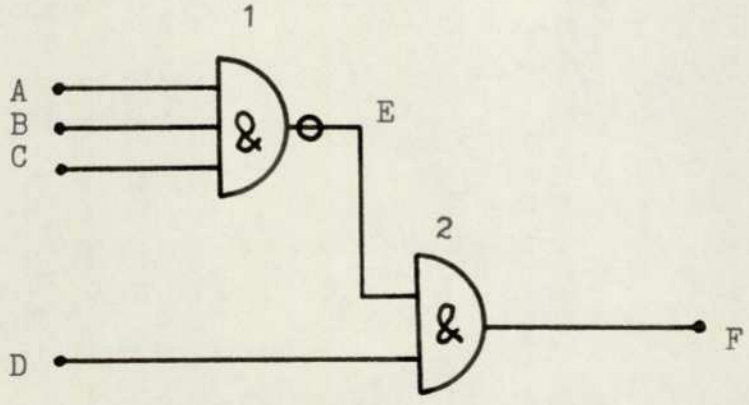


FIG. 2.4. COMPILED-CODE SIMULATION EXAMPLE

subroutine represents one of the circuit elements and consists, usually, of a series of executable machine language instructions coded to perform the element function evaluation. Evaluation of each element proceeds by determining the appropriate driving signals and evaluating the element response. For example, consider the circuit (61) in Fig.2.4 Element #1 can be represented by a series of (illustrative) computer machine-code instructions:

(LOAD ACCUMULATOR)	LDA	A	A
(AND WITH ACCUMULATOR)	AND	B	A.B
(" " ")	AND	C	A.B.C
(COMPLEMENT ACCUMULATOR)	COMP		$\overline{A.B.C}$
(STORE ACCUMULATOR)	STA	E	$E = \overline{A.B.C}$

Element #2 can be similarly represented:

	LDA	E	E $(= \overline{A.B.C})$
	AND	D	E.D
	STA	F	F = E.D

Providing that the routine for #1 is executed prior to that of #2, the output F as a function of A,B,C and D will always be correctly computed. The ordering of the compiled-code routines, representing the interconnected logical functions must correspond with the order in which signals would naturally propagate in a real circuit. The ordering can be performed at the time when the circuit description is compiled into the series of function routines. Each logic element is assigned to a level corresponding to the propagation of signals from input to output. An element must be in a higher level than any

driving element and in a lower level than an element to which it drives. It follows that, in the example of Fig.2.4, #1 is in the first level and #2 in the second level.

After assigning to levels, the compiler orders the compiled-code routines of the elements from each level in sequence corresponding to ascending level number. It is also possible to assign levels to circuits having feedback connections (63).

Combinational circuits are simulated by one pass through the sequence of compiled code instructions. Sequential circuits are simulated by repeated passes through the program until the circuit reaches steady state conditions.

There are a few limitations to this technique:

(a) Changing of the logic circuitry usually requires updating of the assigned levels and re-compilation of the program coding.

(b) In general all elements are evaluated as the program is processed, even if a particular signal path is not activated by the circuit input stimulation. Time is therefore wasted. (Hardie and Suhocki (64) have developed a simulator which evaluates only those gates most likely to be changing state).

Examples of compiled-code simulators can be found in the published literature, (64), (65) and (66).

2.2.5 Table-driven Simulation

A circuit interconnection network can be represented by a data structure with each circuit element as a series

of table entries corresponding to such characteristics as element function (i.e. AND, OR, etc.), propagation delay, source of inputs, destination of outputs, etc. The data structure can be operated on by a control program to interpret the propagation of logic signals between the circuit inputs and the outputs.

The circuit of Fig.2.4 could be represented in the tabular form of Fig.2.5. Signal propagation initiated by changes in the input variables A,B,C or D are traced through the table linkage address pointers corresponding to the circuit interconnections. When the output state of a particular element is to be evaluated the function-type is obtained from the table (i.e. AND, OR, etc.) and an appropriate subroutine is called to perform the evaluation. Each function is represented by a single subroutine.

As an example consider the input variables A,B,C and D being set to new states. A change of state in A,B, or C means that element #1 must be evaluated, and a change in D means #2 must be evaluated. The input states A,B and C form subroutine call variables when the NAND routine is accessed to evaluate E. Similarly, E and D are call variables for the AND routine when F is calculated. The element output states are stored in the output table at the locations indicated by the appropriate address pointers.

If only input D changed state the associated signal flow is traced with only element #2 being simulated. Time is not wasted in evaluating #1 because it is already in a logically correct state. This illustrates that only active elements are evaluated, representing a considerable saving

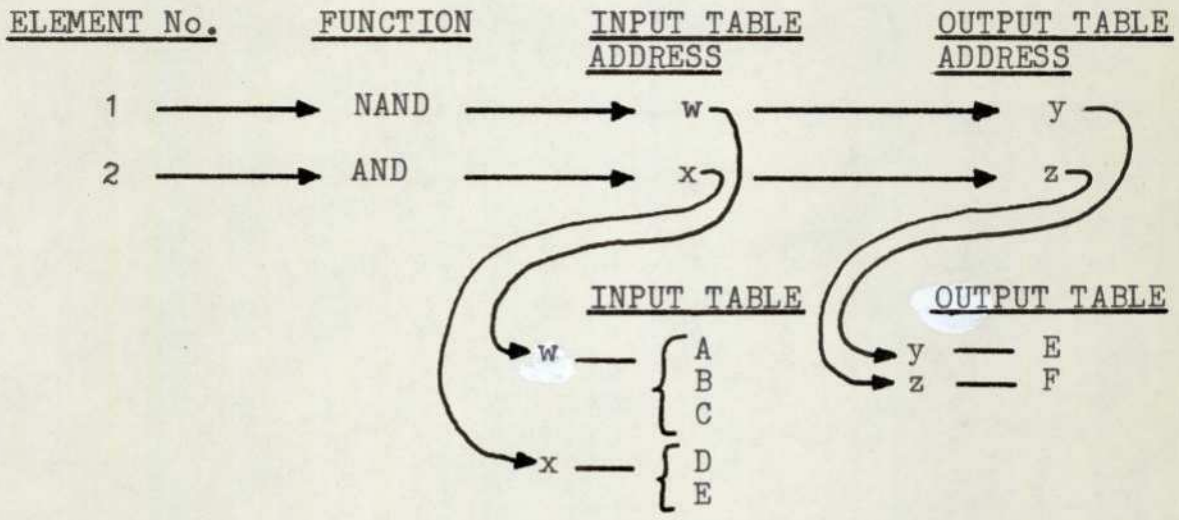


FIG. 2.5. TABLE-DRIVEN TABLE STRUCTURE (EXAMPLE)

in computation time since only a small percentage of circuit elements in most logic systems are active at any one time (61), (67).

Modifications to circuits can be easily arranged by adding table entries and changing address pointers. The use of single subroutines for logic functions reduces the storage requirements for a given circuit in comparison with the compiled-code technique. Also, greater amounts of valid information on circuit activity, particularly from signal flow aspects, is available from the table-driven technique because of the way in which it mimics the actual flow in real systems.

Examples of table-driven simulators can be found in the published literature, (67),(68),(69),(70) and (71).

2.2.6 A Comparison Of Simulation Processes

Any comparison of simulation processes must be placed in the context of the needs of the user. It is obvious that the requirement of a simulator for small sized combinational circuits, without the complexity of delays and timing information, is satisfied by the table look-up type. This offers the user a very quick and efficient solution providing the needs are no more extensive. However, the original table must be constructed in some way, either by manual interpretation or from another simulation process. Once the table is formed, it can be used repeatedly without reference to the actual circuit or the Boolean equations.

In many design situations evaluation of Boolean equations prior to conversion into equivalent circuit

diagrams is required. This is the occasion where only the Boolean equation process is viable. If fault simulation or timing data is required the equations must be synthesised into the circuit representation and one of the other processes used.

Choosing between the compiled-code and table-driven simulation techniques is really a matter of evaluating the user needs, the method of program implementation and the type of host computer, then deciding on the most cost-effective process (61). For example, McKay's table-driven simulator (68) utilises dedicated hardware in a special purpose computer and so obtains fast simulation processing. The compiled-code simulator of Hardie and Suhocki (64) is also fast, obtaining speed by the simultaneous simulation of thirty-three circuit conditions.

In this project it was decided to use the table look-up and table-driven types of simulator. The reasons are as follows:

(a) Combinational circuits to be considered during this project are to be small in size and the table look-up process provides a simple solution, even for fault injection.

(b) As far as sequential circuits are concerned, although simulation speed is not of primary interest in this project, the table-driven process does offer greater speed by not simulating inactive elements.

(c) The ability of the table-driven process to realistically model circuit signal flows makes it more personally acceptable.

The table on look-up technique is so simple to implement that it will not be discussed further. The following sections concentrate on the more difficult problem of the table-driven simulation process.

2.3 TECHNIQUES OF SIMULATOR IMPLEMENTATION

2.3.1 Timing

Logic level transitions in digital circuits occur at discrete instants in time. Therefore, any simulation of such circuits need only evaluate element functions at the points in time at which the transitions are due to take place. Time is thus quantized, the steps being determined by the delay characteristics of the elements and the input clock timing.

In a simple timing model each element would have a fixed standard delay of one unit (of time). The timing of the complete circuit would then be directly related to multiples of the unit of delay time. For example, the delay of the circuit output response to an input stimulation would be the sum of the number of elemental delay units encountered between input and output.

Where more realistic timing information is required, such as in analysing circuit hazard and race conditions, a more sophisticated model is used. Such a model could utilise statistically distributed delay times based on a Gaussian or some similar distribution (72), as is found in real logic elements.

A generally sufficient but rather less sophisticated model than the previous one has been presented by

Ulrich (67). This consists of allocating an expected value of delay to each of the n elements of a circuit, t_{d1}, t_{d2}, ..., t_{dn}. The smallest step of time, Δt, between the occurence of any two events in the circuit corresponds to the largest common divisor of these delays. All other events occur at multiples of Δt.

A variation on the above model, and the form used in the simulator constructed in this project, is to assume the value of Δt, calling it one unit of delay (time), and allocating delays to elements as multiples of this. For example, an AND gate could be defined as having 3 units of delay and an OR gate 4 units.

Two different techniques for determining and processing the passing of time in the simulation of logic circuits can be identified (61), one synchronous the other asynchronous. These terms refer to the process, not to the type of circuit being simulated. In the synchronous process the time stepping is performed in fixed intervals controlled by an imaginary clock signal. At the occurence of each clock those elements that are due to be evaluated are simulated. The speed with which a given element reacts, by itself generating a stimulation to a connected gate, is not relevant since it would not be evaluated until the next clock occurs.

In an asynchronous technique any stimulation is processed at the instant in time (simulation time) at which it occurs. Time thus passes in a series of steps determined directly by the propagation speeds of the circuit elements.

The choice of technique is related to the needs of the user. If clocked digital circuits using elements of standard delay are to be simulated, the synchronous method is the most simple to implement. The asynchronous method is a more realistic model of the actual process of signal propagation in digital circuits. For this reason the asynchronous technique was used in this project. Examples of both types of simulation timing can be found in the published literature; synchronous (65),(66) and (69); and asynchronous (64),(67),(68),(70) and (71).

2.3.2 Circuit Structure Coding

For a circuit to be simulated by a computer it must be coded in a format suitable for acceptance in the simulation program. The form of coding is to some extent dependent on the simulation process, the programming language and the type of computer. The coding must contain sufficient information to fully describe the structure and function of the circuit. In some cases this may be dependant on the coding methods used in an overall D-A system of which (perhaps) the simulator is to be a part.

The requirements for coding in this project are as follows:

(a) The coding should be suitable for interpretation and storage within the structure of a FORTRAN IV program.

(b) The format should be suitable for a table-driven asynchronous simulation process.

(c) The element description data should contain information on function and propagation delays of

simple multiple-input/single-output logic gates (AND, OR, etc.).

A description of how these requirements were implemented follows. The simple circuit of Fig.2.6(a) is used as an example.

To aid the utilisation of the circuit model in the simulation process, inputs and outputs are treated as circuit elements. Each type of element has a function code-number associated with it:

FUNCTION		CODE
INPUT	-	1
OUTPUT	-	2
AND	-	3
NAND	-	4
OR	-	5
NOR	-	6

INVERSION is performed by a single input NAND or NOR.

As an aid to consistency circuits are drawn as in Fig.2.6(a) with the input-to-output signal flow being from left-to-right. Each circuit element, including INPUT and OUTPUT, are given a number. Numbering of an n-input, m-output and p-gate logic circuit is always performed in order as follows:

(a) Starting at the top, assign numbers 1 to n to the n inputs (in order).

(b) Starting at the top, assign numbers (n+1) to (n+m) to the m outputs (in order).

simple multiple-input/single-output logic gates (AND, OR, etc.).

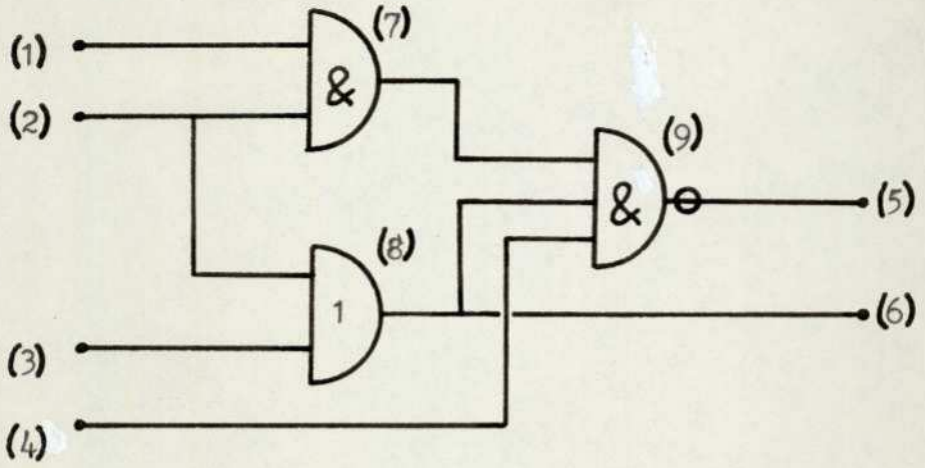


FIG. 2.6.(a). A CIRCUIT EXAMPLE

Card Numbers ↓ | Column Numbers →

Card Numbers	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1		1		1		1				7									
2		2		1		1				7		8							
3		3		1		1				8									
4		4		1		1				9									
5		5		2		1		9											
6		6		2		1		8											
7		7		3		4		1		2				9					
8		8		5		3		2		3				9		6			
9		9		4		3		4		7		8				5			...

Number — Function — Delay — Connection Data

FIG. 2.6.(b). CODED CIRCUIT EXAMPLE

(c) In any order, assign numbers (n+m+1) to (n+m+p) to the p circuit gate elements.

The results of this procedure is illustrated by the bracketed numbers in Fig.2.6(a). The consistency of such a description permits the use of a very simple input/output coding for simulation data. For example, the simulation of the binary-code 0 1 1 0 on inputs 1,2,3 and 4 respectively, can be described simply by Input = 6 (assuming the least-significant-bit is associated with the lowest input number). The resulting output 0 1 on outputs 5 and 6 respectively, is described by Output = 2. This will be discussed in more detail later.

The circuit structure and functions can be manually coded by stating, for each circuit element:

(a) the element number.
(b) the element function.
(c) the element propagation delay.
(d) the element numbers from which the input connections are dervied.
(e) the element numbers to which the output is connected.

The coding format developed for this project is suitable for placing the circuit information on normal 80 column computer cards. Each circuit element is given a separate card. For simplicity, element functions are directly described by their code-numbers rather than using descriptive code and having to pass through an intermediate interpretation in the simulation program.

Each piece of coding data is given two spaces on the

computer card (i.e. one I2 format in FORTRAN). This (artificially) limits the number of elements and the maximum element delay to values of 99. The nine coding cards required to fully describe the nine element circuit of Fig.2.6(a) is shown in Fig.2.6(b). The appearance of a double space (which is equivalent to zero in I2 format) indicates the termination of a list of input or output connections.

The computer cards containing the functional and structural circuit description are read in by the simulation program and the data is stored in several one-dimensional arrays. Three of these are named NELEM, NFUNC and NDEL. The i^{th} array element of each contains information associated with the i^{th} numbered circuit element, where i = 1,2,.....,n,(n+1),.....,(n+m),(n+m+1), ...,(n+m+p) for a circuit of (n+m+p) elements.

The i^{th} array elements of NFUNC and NDEL store the i^{th} circuit element function code and propagation delay respectively. The i^{th} array element of NELEM contains a pointer to array position q in another one-dimensional array NCONC where input and output connection data of circuit element i is stored. If the i^{th} circuit element has connections from r circuit elements and to s other, the data is stored in the qth to (q+r+s)th array elements of NCONC. This is in the form of the connected element numbers. For ease of identification inputs to the i^{th} circuit elements are given negative values and outputs positive values. The circuit of Fig.2.6(a), when

presented to the computer program in the format of Fig.2.6(b), is stored in the four arrays as shown in Fig.2.7(a). The arrows indicate the storage of information common to particular elements.

The four arrays described above contain the functional and structural characteristics of the circuit to be simulated. Further space is required to store the operational aspects of the simulation process, that is the circuit element logic states. A one-dimensional array NSTAT is used to store this operational data. The i^{th} array element stores the present logic level (0 or 1) of the output of the i^{th} circuit element. This is shown in Fig.2.7(b) and is illustrated with the example of Input = 3 (i.e. 1 1 0 0 at 1,2,3 and 4) and Output = 2 (0 1 at 5 and 6). The overall process of evaluating such a result (the simulation process) will be discussed later. The array NEXTS, associated with NSTAT in Fig.2.7(b), is utilised in this process for temporary storage of logic levels. The exact function of NEXTS will be described later.

2.3.3 Circuit Element Function Evaluation

The overall characteristics of a circuit is determined by the combined operational characteristics of the individual circuit elements. In a table-driven simulation process each type of element function is evaluated by a common routine.

To illustrate how these functional routines are constructed and utilised assume that at the present

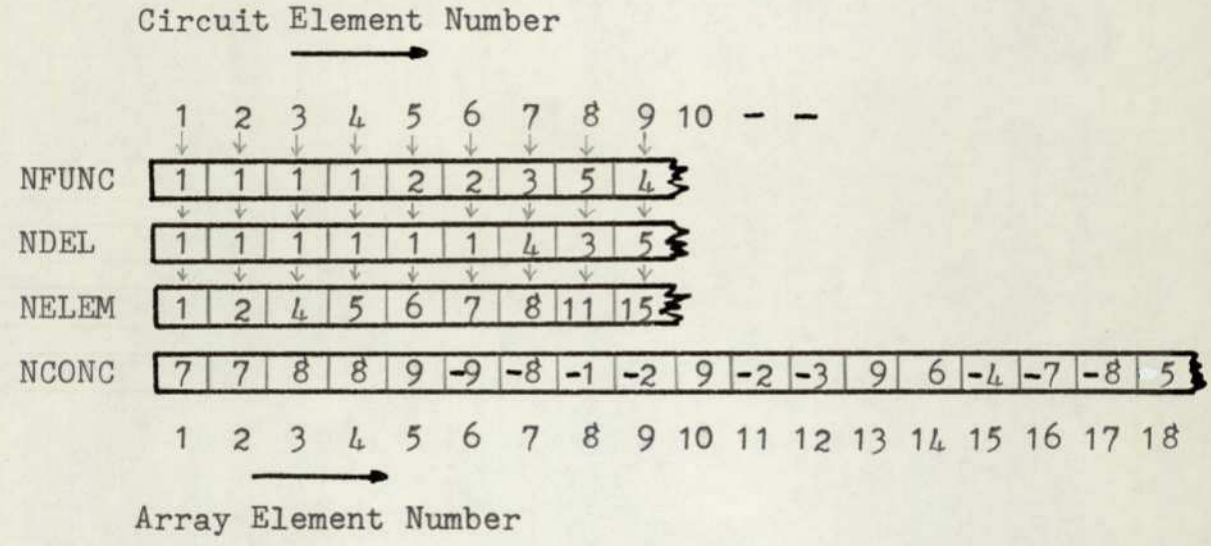


FIG. 2.7.(a). CIRCUIT FORMAT IN PROGRAM ARRAYS

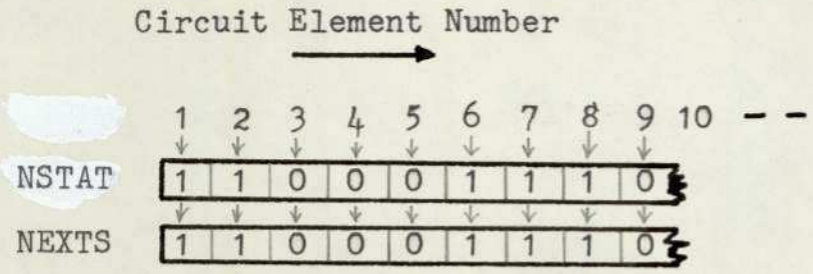


FIG. 2.7.(b). CIRCUIT OPERATIONAL CONDITION ARRAYS

instant in time, t_q, one of the inputs to circuit element i has changed state. Element i must be evaluated to determine if it is affected by the input stimulation. The function of element i is found simply by accessing the i^{th} array element of NFUNC. Depending on the function code, the appropriate function routine is called. These routines are:

(a) AND - Using the i^{th} array element of NELEM calculate the address q in NCONC at which connection data for circuit element i is stored. The connection data is used to determine the logic levels of all the circuit elements connected to the inputs of circuit element i. The function is evaluated by putting NEXTS(i) = 1 if all inputs are 1, or NEXTS(i) = 0 if any input is 0. The flow diagram is shown in Fig.2.8(a). The only data supplied external to the routine is the circuit element number i.

(b) NAND - This function is evaluated in the same way as AND, with the exception NEXTS(i) = 0 if all inputs are 1, or NEXTS(i) = 1 if any input is 0.

(c) OR - The logic levels at the inputs are evaluated in a similar way to the AND gate. NEXTS(i) = 1 if any input is 1, or NEXTS(i) = 0 if all inputs are 0. The flow diagram of this routine is shown in Fig.2.8(b).

(d) NOR - This routine is evaluated in the same way as OR, with the exception NEXTS(i) = 0 if any input is 1 or NEXTS(i) = 1 if all inputs are 0.

(e) INPUT - This routine has no input connections to be accessed and is only evaluated when called by an

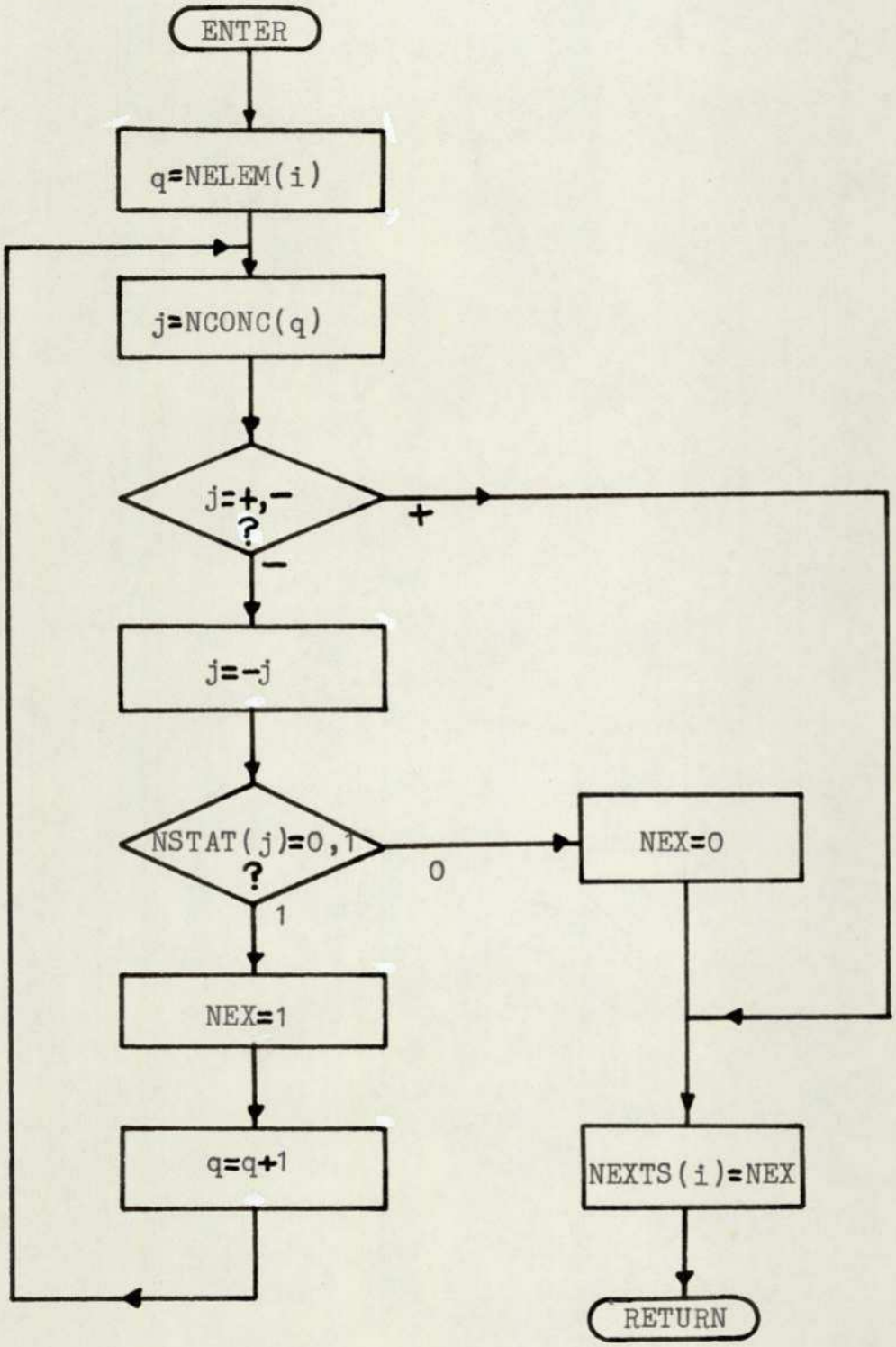


FIG. 2.8.(a). 'AND' GATE FUNCTION FLOW DIAGRAM

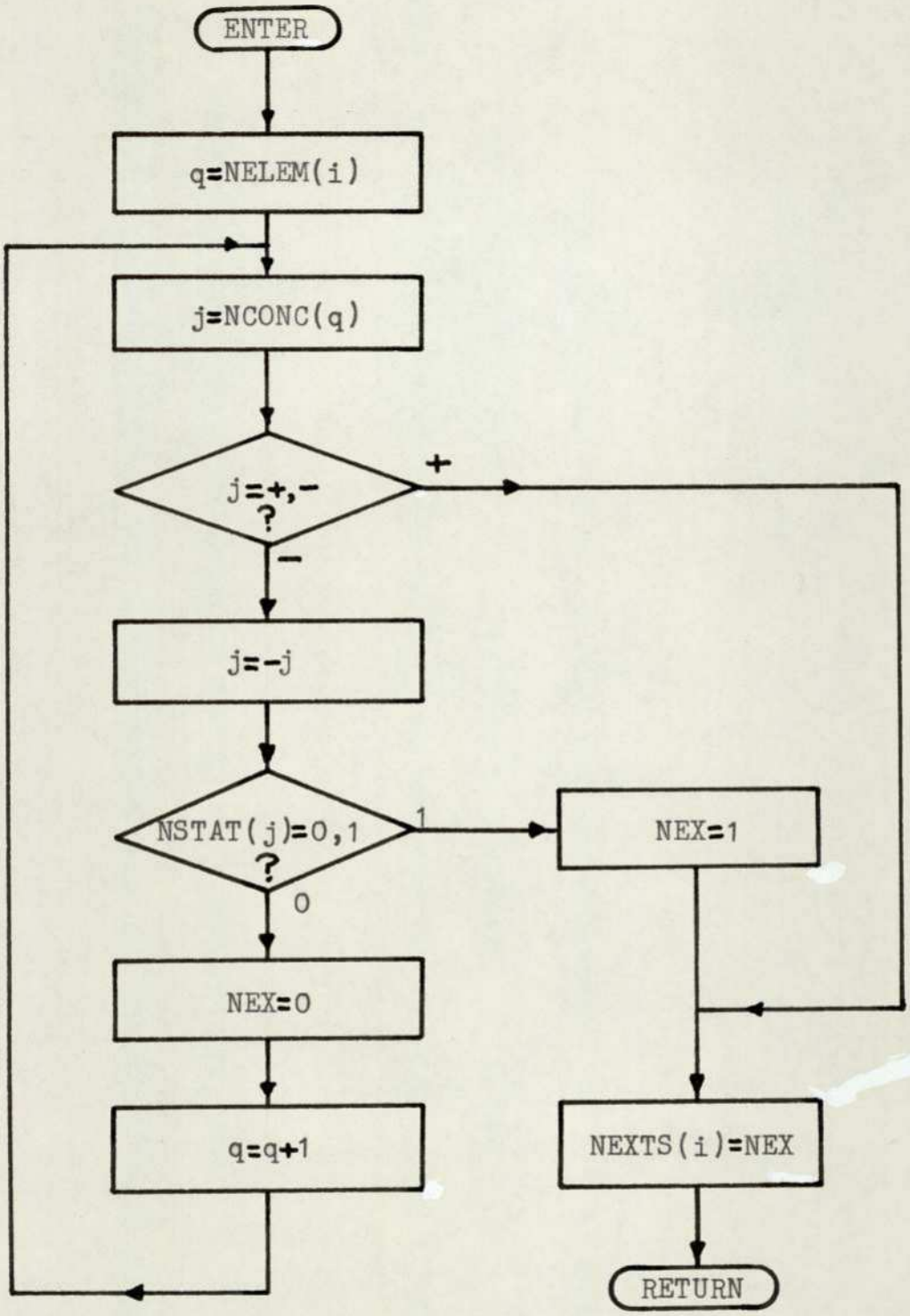


FIG. 2.8.(b). 'OR' GATE FUNCTION FLOW DIAGRAM

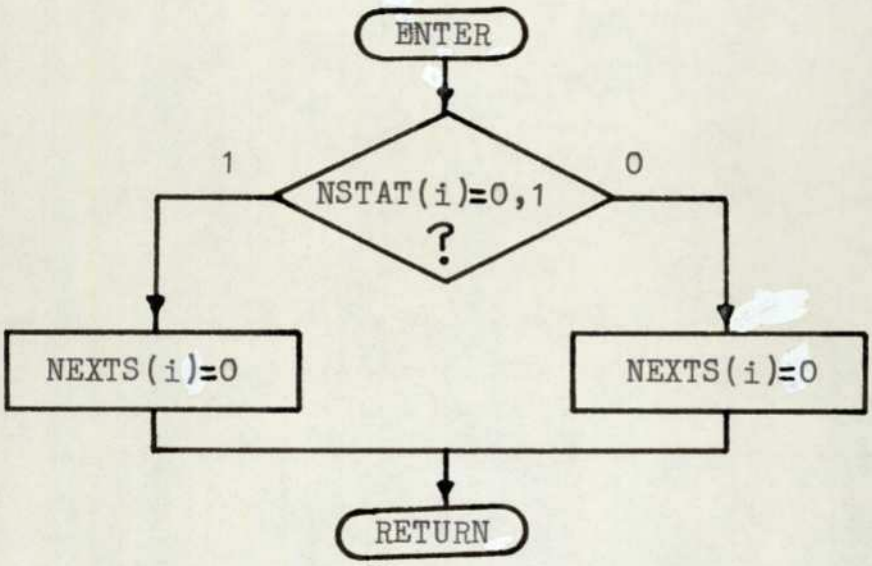


FIG. 2.8.(c). 'INPUT' FUNCTION FLOW DIAGRAM

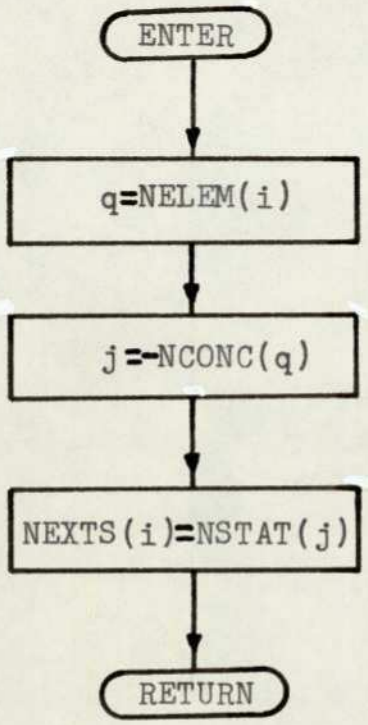


FIG. 2.8.(d). 'OUTPUT' FUNCTION FLOW DIAGRAM

external input stimulation routine (i.e. when a new input test is to be evaluated). The routine is only called when a change of state is defined. So, NEXT(i) = 0(1) if NSTAT(i) = 1(0) as shown in the flow diagram of Fig.2.8(c).

(f) OUTPUT - This is a very simple routine. The element state takes on the value of the output of the element from which it is connected. So, NEXTS(i) = NSTAT(j) where j = -NCONC(q) and q = NELEM(i) as shown in the flow diagram of Fig.2.8(d).

The reader will note that in each case the newly evaluated state is not set directly in NSTAT(i), rather it is stored in NEXTS(i). This permits (psuedo-) simultaneous evaluation of several elements that are stimulated at time t_q. Only when all have been evaluated are the new states set in the appropriate NSTAT(i), with NEXTS(i) being used only as a temporary store.

2.3.4 Logic Circuit Initialization

When the power supply to a logic circuit is switched on the elements making up the circuit react dynamically to reach steady state conditions satisfying the various logical restrictions of the elements. A computer program simulation using the table-driven process, only following active paths of a circuit, is pseudo-dynamic and might not always generate logically consistent states when the program is switched on. Unless the values of NSTAT are pre-defined, for some initial circuit input, inconsistencies such as the output of a NAND gate being 0 when the inputs

are 0, could occur. The circuit model does not react dynamically to correct such situations, it must be programmed to do so. This is a process called Initialization.

A very simple form of initialization would be to manually define and supply logically correct circuit states by setting the array elements of NSTAT appropriately. This could be a complicated process for large circuits.

The method used in this project is to initially define the n circuit input states in NSTAT(j), j = 1,....,n, and then evaluate each of the other (m+p) circuit elements by calling the appropriate function evaluation routines. These evaluations are repeated until no inconsistencies are found. When this situation is reached the circuit is correctly initialized.

A flow diagram of the initialization procedure is shown in Fig.2.9. If the number of initialization cycles, LOOP, exceeds some defined limit, LIMIT, it is assumed that a logical inconsistency that cannot be reconciled exists (i.e. oscillation).

If the circuit being simulated is sequential it is possible that initialization to pre-determined stored states is required. This is done by setting the logic levels of the elements from which the circuit feedback lines derive, to the required values. A one-dimensional array NFB is used in this process. All of the (n+m+p) array elements of NFB are set to zero except those corresponding to the feedback elements, these are set to 1. The normal initialization proceeds. However, when an element is

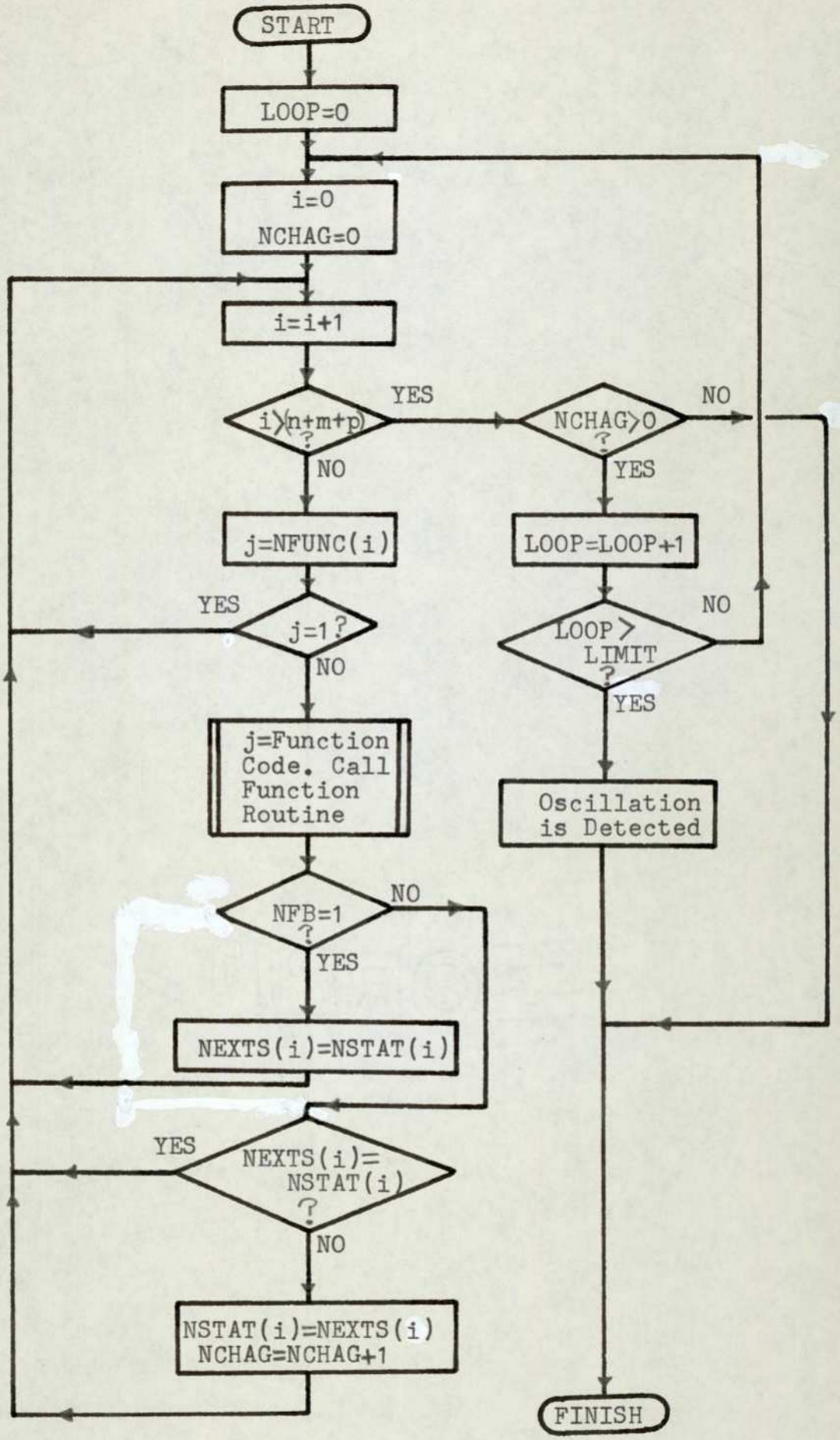


FIG. 2.9. FLOW DIAGRAM OF INITIALISATION PROCESS

evaluated having NFB(i) = 1, the output state, NSTAT(i), is not allowed to change. The state is thus fixed to the required value. When the initialization has been successfully completed all array element of NFB are returned to 0.

It is possible that the state fixing procedure can induce undetected inconsistencies. An example is when the output of an AND gate is fixed at logic level 0 with all inputs at logic level 1. When the fixing process is terminated the circuit can be simulated to determine the effect of removal of forced states.

2.3.5 Logic Circuit Simulation Routine

The process of initialization sets the circuit states into logically consistent and quiescent conditions. The next task is to simulate the operation of the circuit under the influence of changing input logic levels.

Ensuring that events occur at the correct points in time is an important consideration in the design of a simulation process. When a given circuit element i is stimulated by a changing input at time t_q, the effect will not appear at the element output until time $t_q + t_{d_i}$ (where t_{d_i} is the i^{th} element propagation delay). The simulation of such an event is correct only if the output change is made to occur at $t_q + t_{d_i}$.

One technique for controlling simulation timing is due to Ulrich (67) (see Section 2.3.1). The individual element delays determine the smallest circuit time step Δt. Future activity is stored in a circular timing stack, each stack

position corresponding to a time increment of Δt. For example, present activity is located at the stack position corresponding to t_q. An output event at element i due t_{d_i} later in time is scheduled to occur at a stack entry $t_q + t_{d_i}$, where t_{d_i} represents a defined number of Δt steps. All other events due at the same point in time are stored in the same future activity stack position. When present activity has finished the time is incremented by Δt and the events scheduled for the next stack position are allowed to occur. Empty stack positions are are not considered. The circular nature of the stack reduces the problems of storage space.

Another technique used in the published literature (68),(70), and the one used in this project, is to store future activity information in a list that is ordered in a *time-to-be-executed* form. This is rather less efficient than the previous method since, each occasion a future event is designated, the list has to be re-arranged to fit it in at the appropriate point. This method has the advantage that time passes in steps defined only by the time at which future activity occurs, not in the basic Δt units.

In this project present circuit activity is stored in a one-dimensional array named NSIM. This consists of a list of circuit elements, designated by their associated numbers, that are to be evaluated at the present instant in time, t_q. Associated with NSIM is a counter NT containing a count of the number of elements to be simulated. Future activity is stored in two one-dimensional arrays NSORT and

NTIME. NSORT lists the element numbers at which future activity is due, in their order of occurence. NTIME stores the actual time at which each future event is due to occur. NTIME information is arranged in sequence of element number, NTIME(i), i = 1,.....,(n+m+p), corresponding to element i. If the i^{th} element is not designated for future activity then NTIME(i) = 0. NZ is a counter for the number of events stored in NSORT.

Not all present activity results in a change of state either immediately or in the future. For example, a logic level change 0 to 1 at one input of an OR gate whose other input is already at 1 does not produce a change in the gate output. Those activities resulting from present activity, scheduled for the future, are stored in NSORT and NTIME. Those activities due immediately at the next time step are stored (as circuit element numbers) in a one-dimensional array NSTCH. NS is a counter for the number of events. Also, any future event in the NSORT stack due at the next time step is un-stacked and stored in NSTCH. When present activity ceases, time is incremented to the next time step (if any events are due) and the new NSIM stack is calculated by determining to which elements those stored in NSTCH are connected. This forms the new current activity stack.

The full simulation process is shown in the form of a flow diagram in Fig.2.10. To use such a routine some external data must be supplied. The time count, NCOUN, is defined at the time the routine is entered and is incremented as the simulation proceeds. The circuit input

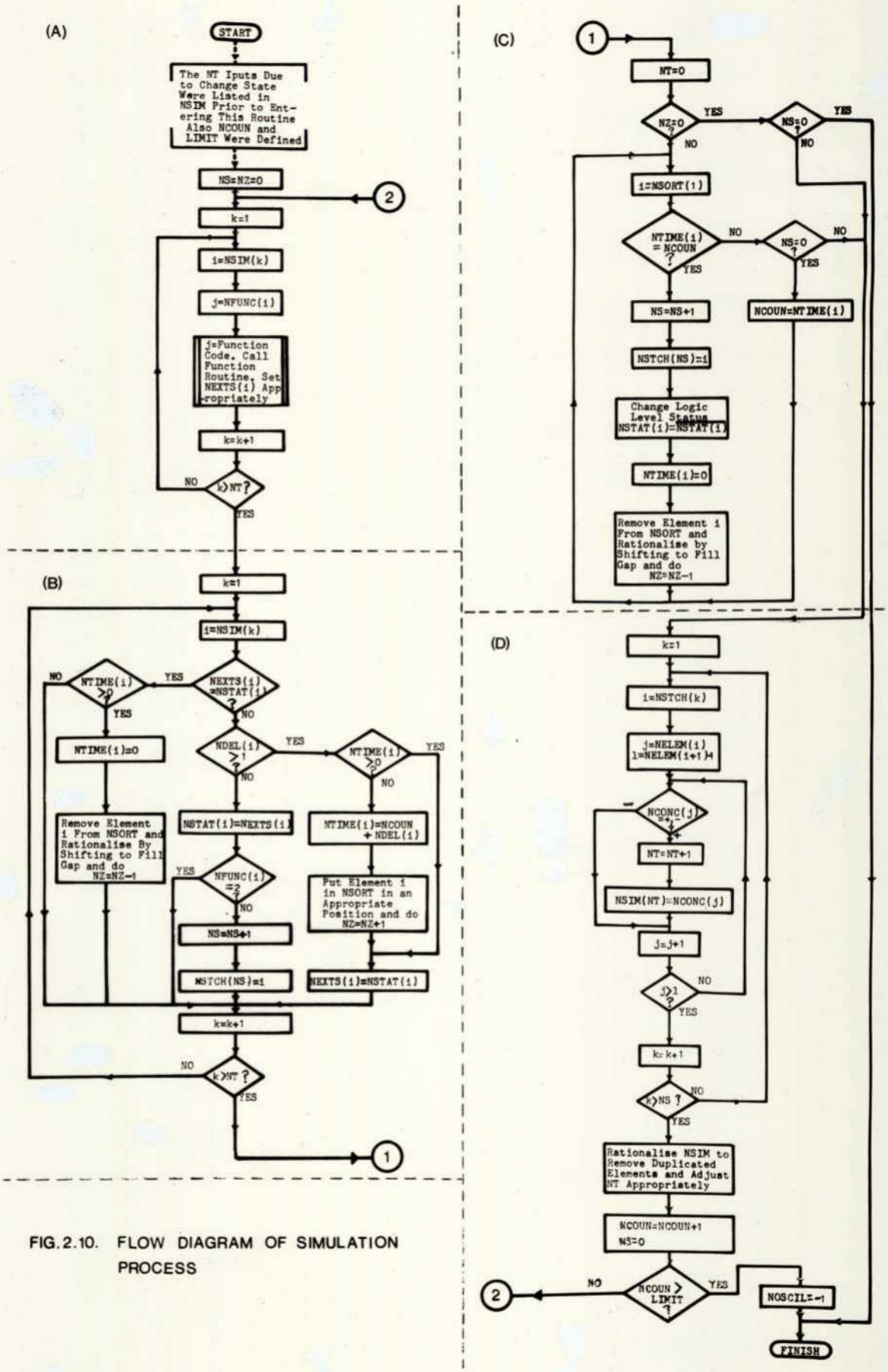


FIG. 2.10. FLOW DIAGRAM OF SIMULATION PROCESS

test being simulated is defined by setting up in NSIM the element numbers of the inputs that are to change state and by setting the counter NT appropriately. The other counters, NS and NZ, are set to zero prior to entering the routine.

A given element number may be listed more than once in the current activity stack (NSIM) due to it being stimulated by more than one input. Multiple entries in NSIM can be removed since the circuit element need only be evaluated once.

After having processed the current activity stack, if NS = 0 there is no immediate activity due. The NSORT stack is then examined to find the next future activity. The counter NCOUN is set to the value of time at which this event is due, thus making it the current event (NCOUN = NTIME(i)). If both NS and NZ are zero, indicating no future activity is due, the simulation is complete and the circuit has reached logical equilibrium. The circuit outputs can be found by examining the appropriate NSTAT(i) array elements, i = (n+1),.....,(n+m).

If NCOUN, the time counter, exceeds some defined limit, LIMIT, the routine is terminated because, in all probability, oscillation has occurred. This could be checked by observing if a sequence of simulation events is repeating itself.

In the flow diagram of Fig.2.10 section A shows the way in which the current activity stack is processed. Section B illustrates the stacking of future events in NSORT and the determination of those events occurring at the next time step which are to be stored in NSTCH. Section C shows the un-stacking of future events from NSORT and their

listing in NSTCH at the appropriate times. Section D creates the new future activity stack that results from NSTCH at the next time step.

2.3.6 The Simulation Routine As A Fortran Subroutine

Several options are open for the type of computer language and method of implementation of the simulation routine described in the previous sub-sections. One possibility would be to write the program in the machine language of the host computer. This would permit the use of the complete computer word for compact storage and also allow utilisation of all the sophisticated bit manipulation and logical instructions to give fast and efficient routines.

Another method would be to use a high level language such as FORTRAN IV. Although not as efficient for this type of work, it does offer considerable programming flexibility and removes much of the hard work of writing special routines for input/output control and general format problems. Since there was no unreasonable restrictions on the amount of computer storage or program run-times it was decided to use FORTRAN IV in this project. This does have the added advantage of making the programs machine-independant.

It was decided to write the simulation routine as a general subroutine since it is required for use in several different applications. The subroutine is divided into three independant sections each being accessed by setting up a flag, NSTEP, prior to the subroutine call. NSTEP is

set to -1, 0 or +1 to call the first, second or third sections, respectively. The first section reads in the circuit functional and structural data and sets up the various arrays described in Section 2.3.2. The second section is the initialization routine. Prior to calling this, initial inputs must be set in NSTAT array. The third section is the full simulation routine. The input test that is to be simulated is set up by listing those inputs that change state in the NSIM stack and by setting NT appropriately. When control is returned to the master routine the simulation has been performed and the output result can be obtained from the appropriate NSTAT array elements. Premature exit from either the initialization or simulation routines occurs when oscillation is detected. This is indicated to the master routine by setting NOSCIL to a value of -1 (otherwise it is zero).

The subroutine is used in the following manner:

(a) Read in circuit data:

  (i) NSTEP = -1.

  (ii) CALL SIMULATION.

(b) Initialize circuit with Input = X:

  (i) Set up NSTAT array for Input = X.

  (ii) NSTEP = 0.

  (iii) CALL SIMULATION.

(c) Simulate circuit with Input = Y:

  (i) Set up in NSIM those input elements which are due to change when Input = Y.

  (ii) NSTEP = 1.

  (iii) CALL SIMULATION.

(iv) Interpret output result by examining appropriate NSTAT array positions.

(d) Repeat from (c) for each other Input = Y required to be simulated.

External routines for setting up initial inputs ((b)(i) above), listing those inputs changing state ((c)(i) above) and interpreting the output results ((c)(iv) above) are shown in the form of flow diagrams in Fig.2.11(a),(b) and (c). The circuit inputs (X and Y) are supplied to the routines in integer decimal form through the integer variable INPT. The output is converted from the binary form in NSTAT to a decimal integer and stored in NOUT. The output results can then be used in any way desired by writing the appropriate handling, interpretation and printing routines in the master routine.

2.4 LOGIC FAULT SIMULATION

2.4.1 Circuit Testing

One use of logic circuit simulation is the production of circuit testing data. Such data is usually in the form of testing schedules. These sequences of input tests can be applied to the circuit and, by examining the resulting output reactions, faults can be detected or located. Such schedules are generated by simulating the logic circuit in the presence of the faulty conditions.

The usual restrictions on the type of fault are as follows:

(a) The fault is logical and of the s-a-0 or s-a-1 variety.

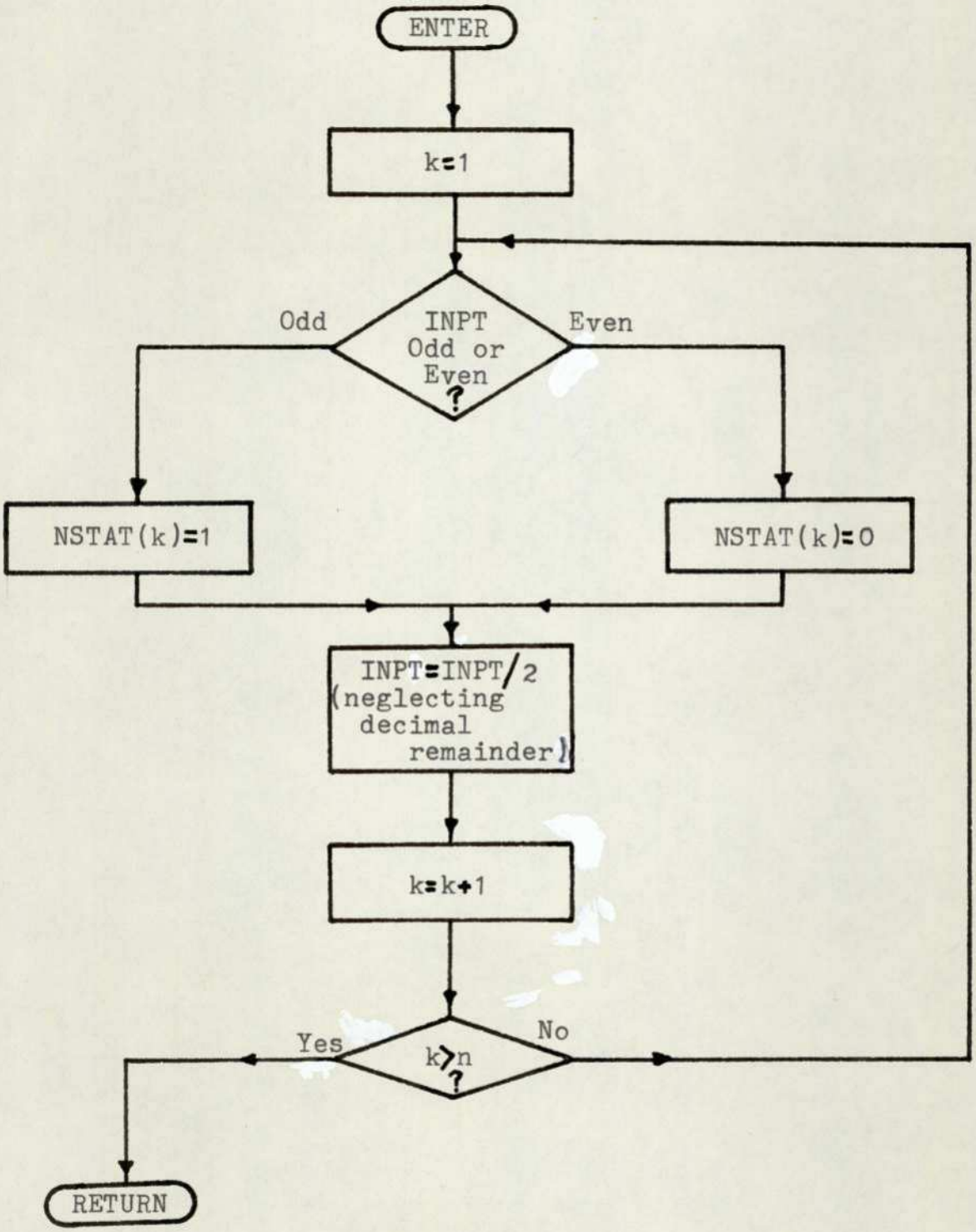


FIG. 2.11.(a). FLOW DIAGRAM FOR SETTING INITIAL INPUT STATES

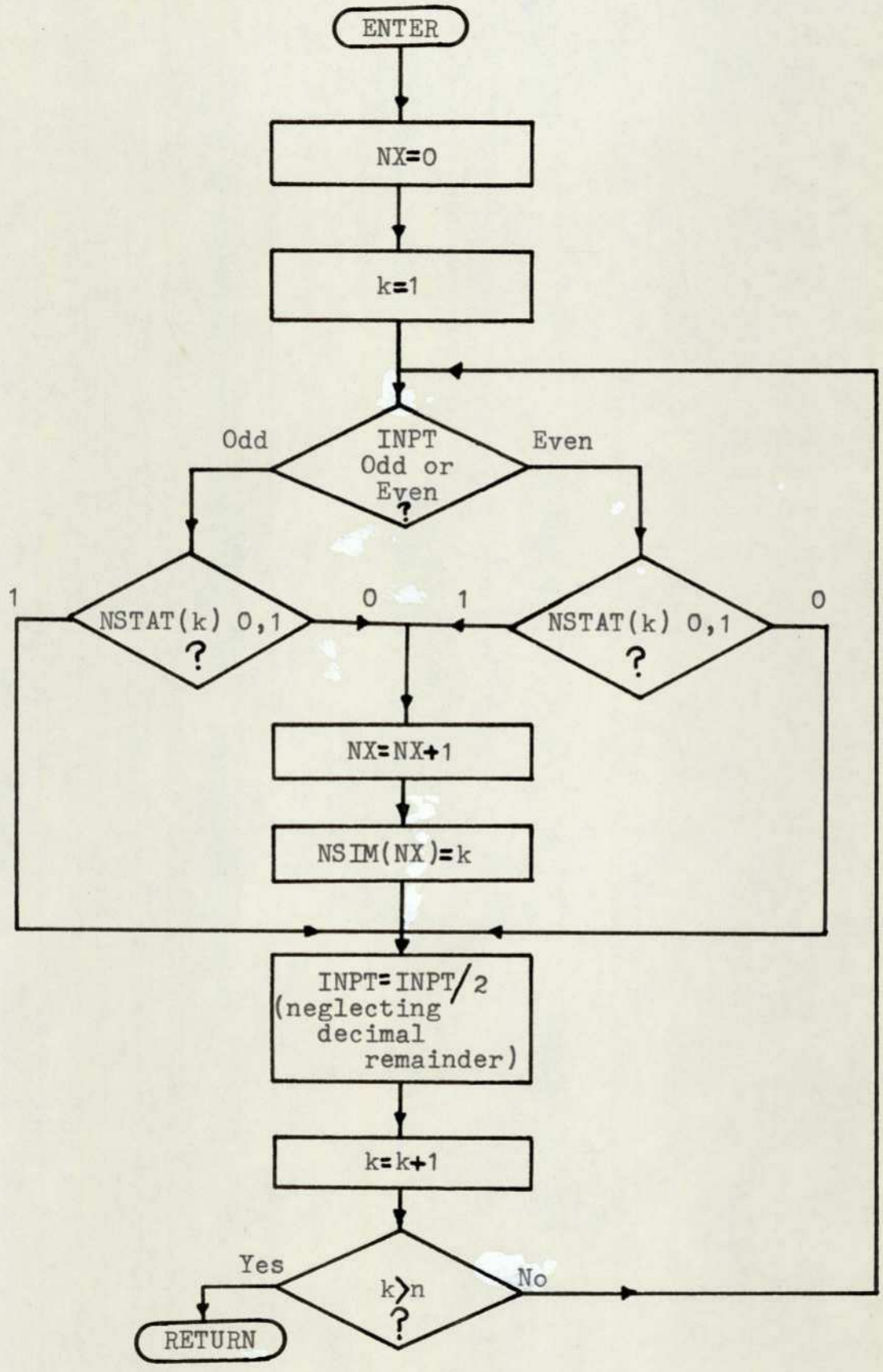


FIG. 2.11.(b). FLOW DIAGRAM FOR LISTING CHANGE OF STATE INPUTS

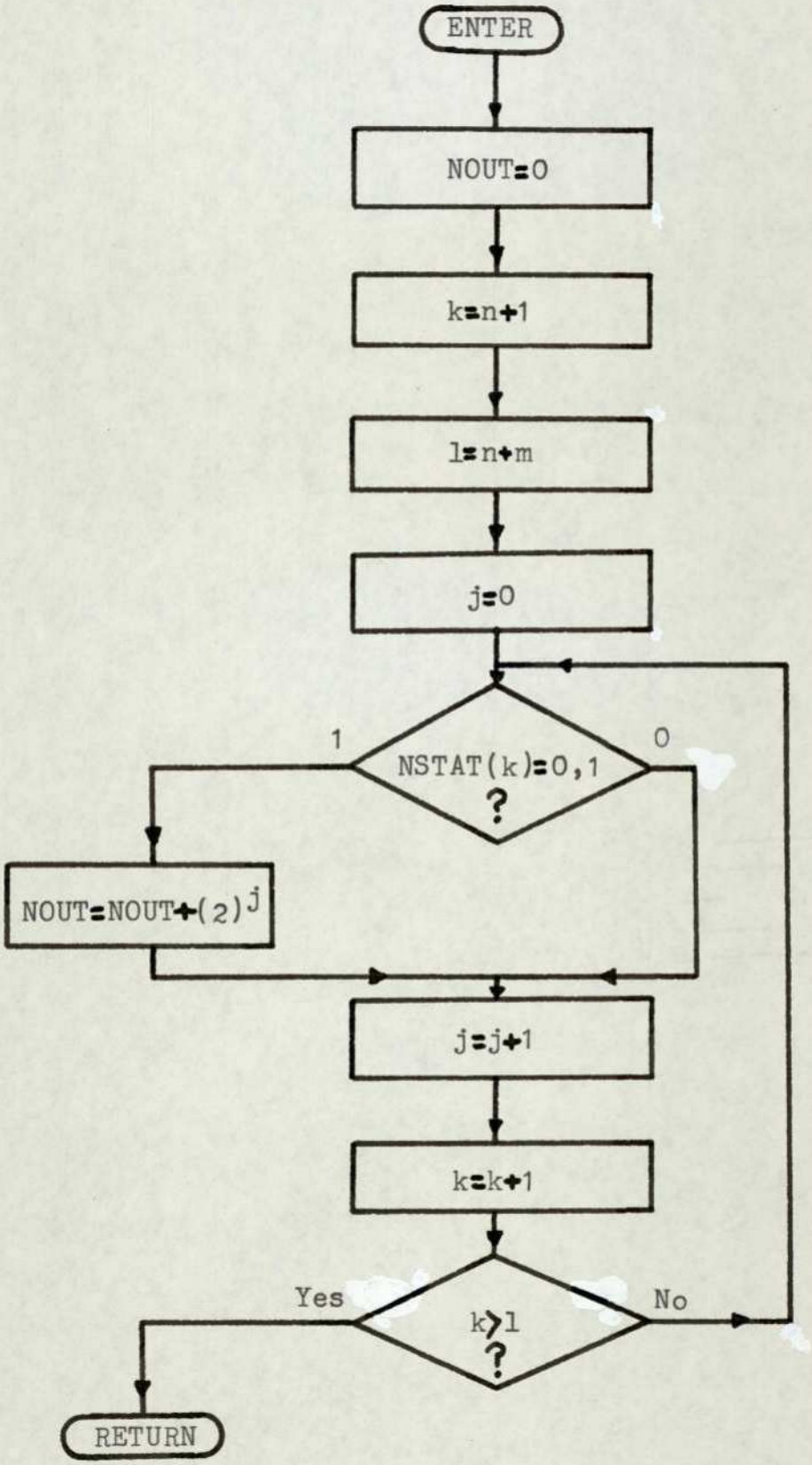


FIG. 2.11.(c). FLOW DIAGRAM FOR INTERPRETING THE OUTPUT RESULTS

(b) The fault affects the circuit behaviour, since there is no redundancy.

(c) Only one fault occurs at any given time.

Three methods of fault simulation have been identified (28), manual, physical and digital. The first consists of the generation of testing sequences by hand. This is done by the design engineer tracing the effects of each fault in the presence of each input test to the circuit outputs. This can be very efficient for small circuits with limited size fault sets, but is liable to inaccuracies due to human error.

The second method, and one used extensively by Bell Telephone Laboratories for their electronic telephone switching systems (73),(74), requires manual selection of possibly relevant tests or testing sequences, and application of these to the real circuit in the presence of manually inserted failure conditions. In many cases the construction of prototype circuits and systems for performing such studies can be excessively costly.

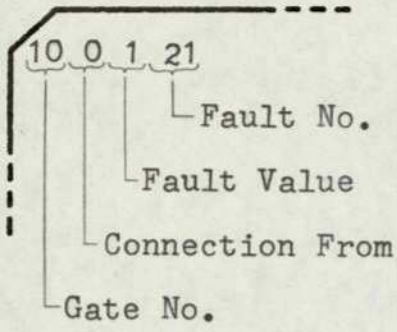
The third and most commonly used method is that of software simulation using the type of simulator described previously in this section of the thesis. The faulty conditions are injected into the circuit model and the reactions of the circuit are determined in the presence of various input tests. Such an approach removes the human error problems and the need for a commitment to a hardware design at an early stage in the design process. Because of the flexibility offered by this approach it was chosen as the method for use in this project.

2.4.2 Fault Injection

The presence of a fault in a circuit alters the overall logic function of the circuit. This can be simulated by reconstructing the circuit model to reflect the new logical function. However, the task of reconstruction by replacing and altering part of the program code for each of the possible faults can be lengthy. This problem is overcome if fault parameters, which can be set or reset depending on the presence or absence of the fault, are included as part of each logic element function evaluation routine. A simple control operation external to the simulation subroutine can be used to set or reset any of the fault conditions at will. This technique has been described in published literature (38),(64), and is used in this project.

For simplicity it is assumed that faults occur only at the inputs and outputs of the logic gates. This is a reasonable assumption since the effects of most faults internal to the gates do appear as input or output terminal malfunctions. Faults are thus injected at these terminals. The format for describing them is illustrated in Fig.2.12 (a) and (b). (a) indicates the output of gate number 10 being s-a-1. The fault is numbered #21. Fault #20,(b), is a s-a-0 fault at that input of gate number 10 which derives from the output of element number 7. For element output faults the CONNECTION FROM data is set to zero (as in (a)). This fault data is supplied to the master routine on computer cards, one card for each fault, in the format shown in Fig.2.12.

(a) Fault No. 21, Output of Gate 10 Stuck-at-1



(b) Fault No. 20, Input to Gate 10 from Gate 7 Stuck-at-0

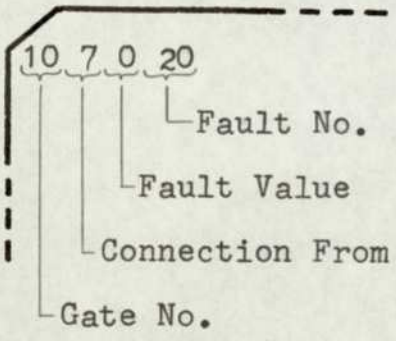


FIG. 2.12.(a)and(b). FAULT DESCRIPTION FORMAT

The data is stored in four one-dimensional arrays, JFALT, IFALT, N1STR and N2STR. Consider the data of q faults (not necessarily numbered 1,2,.....,q) input into the master routine. The k^{th} element of N1STR contains the element number (i) associated with the k^{th} fault that has been input, where k = 1,2,.....,q. For fault number z, the zth element of N2STR contains the address k at which fault data is stored in N1STR and in the other arrays. IFALT(k) contains the logic level of the fault (s-a-1 or s-a-0). JFALT is used to store data on input connection faults. JFALT(k) contains the addresses of the NCONC structural data array (see Section 2.3.2) to which fault number z applies. These four arrays perform no active part in setting up or removing the fault, they only contain the information which allows this to be quickly and simply done. Fig.2.13 illustrates how this data is compiled into the form described above.

The active role in setting up and removing faults is performed by fault parameter arrays NFOUT and NFALT. In the absence of any fault the array elements of both are all zero. If the output of element i is to be fixed at the (faulty) logic level 0 or 1, NFOUT(i) is set to 1 or 2 (where 1 corresponds to logic 0, 2 to logic 1 and zero to no fault). If that input to element i which derives from the output of element x is to be fixed at (faulty) logic level 0 or 1, NFALT(s) is set to 1 or 2. Here s corresponds to the position in NCONC where the relevant connection data between x and i is stored. The data in the four arrays N1STR, N2STR, JFALT and IFALT are used in the setting up of

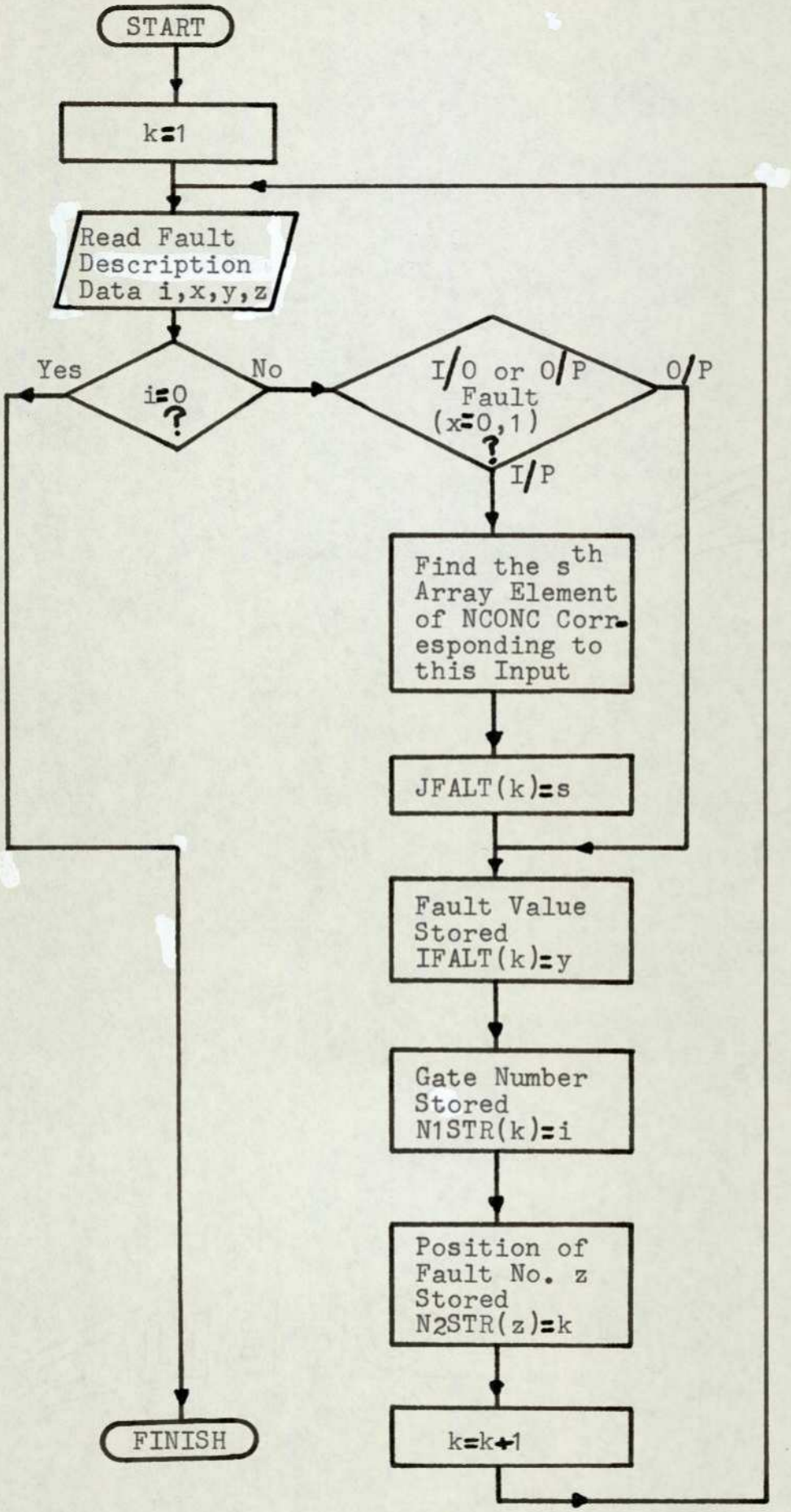


FIG. 2.13. COMPILATION OF FAULT DESCRIPTION DATA

the faults and in their removal. Flow diagrams for these operations are shown in Fig.2.14(a) and (b). Before setting up a new fault and calling the simulation subroutine the present one should be removed, otherwise multiple fault conditions will exist.

The contents of NFOUT and NFALT interact with the simulation process to model the injection of faults within the function evaluation routines of the simulation subroutine. These arrays are COMMON to the master routine and the subroutine. The function evaluation routines described previously in Section 2.3.3 are modified in such a way that the fault parameter arrays are interrogated each time a particular function is evaluated. The circuit element output logic level is set appropriately. This interrogation process and the modifications involved is illustrated with the flow diagrams for the AND, INPUT and OUTPUT evaluation routines in Fig.2.15(a), (b) and (c) respectively.

Intermittent faults can be modelled by setting and removing the fault state at appropriate times as the simulation of a sequence of input tests proceeds.

2.5 SIMULATION SUBROUTINE - RESULTS, LIMITATIONS AND DEVELOPMENTS

2.5.1 Results

The simulation subroutine and the peripheral control routines described in the preceding sub-sections were implemented in FORTRAN IV programming language. This task was simplified by the high level nature of the language. The more simple table look-up type of simulator, using a two dimensional truth-table for storing data on the

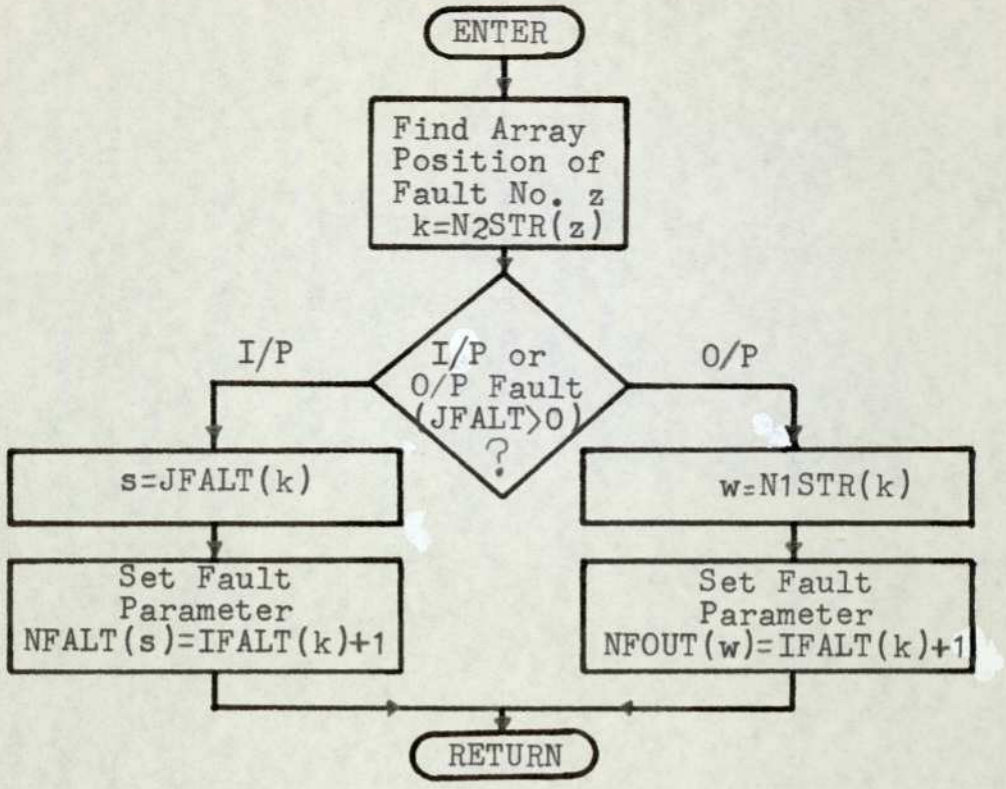


FIG. 2.14.(a). FAULT SET-UP PROCESS

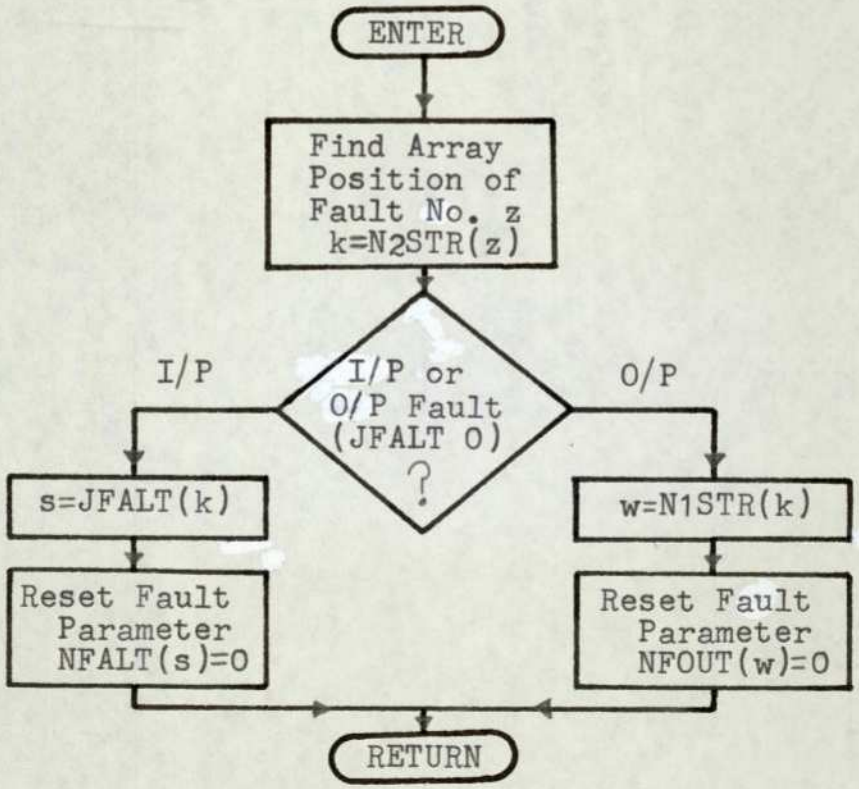


FIG. 2.14.(b). FAULT REMOVAL PROCESS

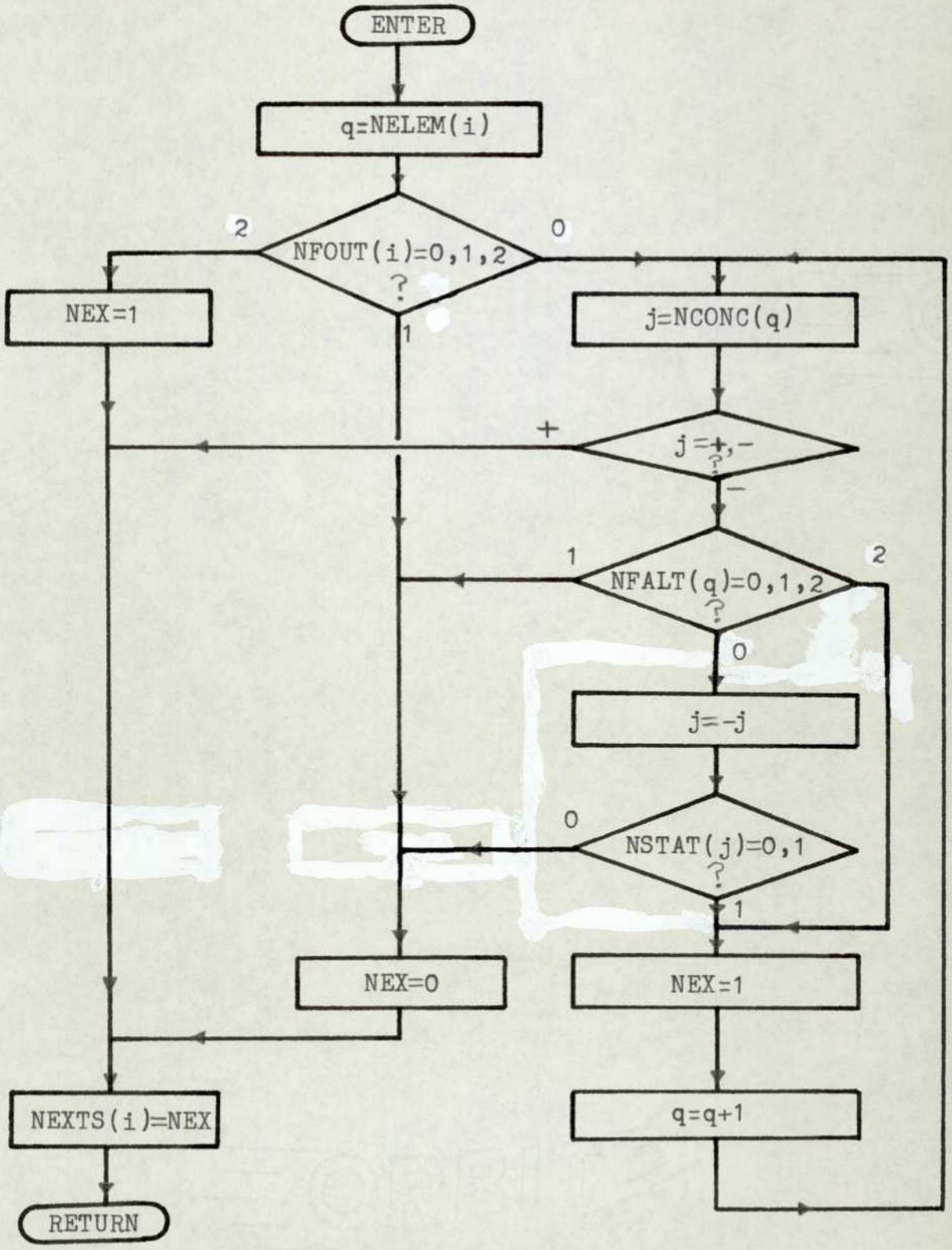


FIG. 2.15.(a). MODIFIED 'AND' GATE FUNCTION

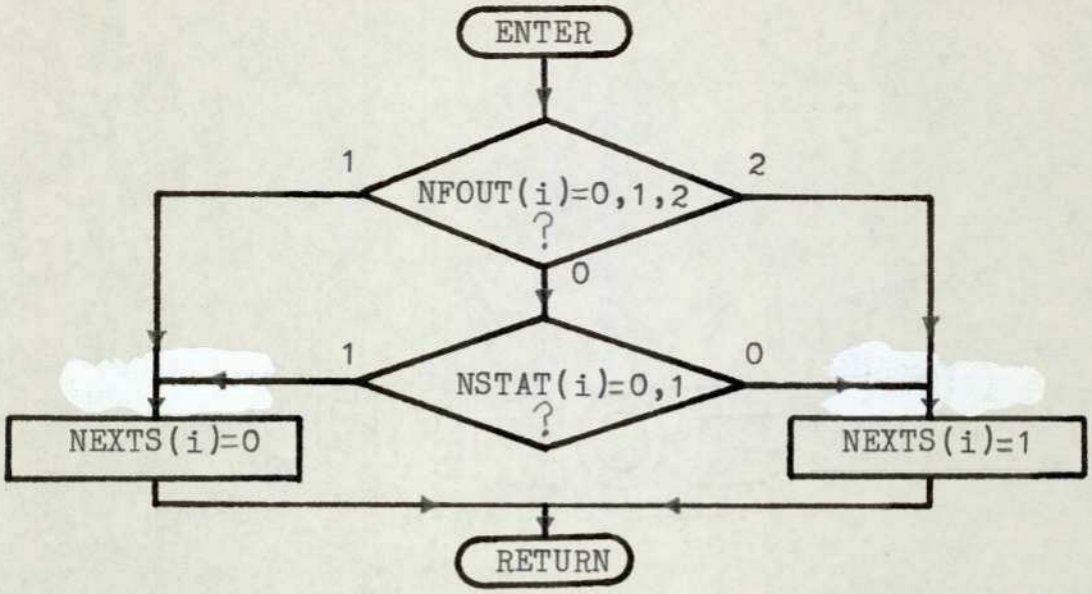


FIG. 2.15.(b). MODIFIED 'INPUT' FUNCTION

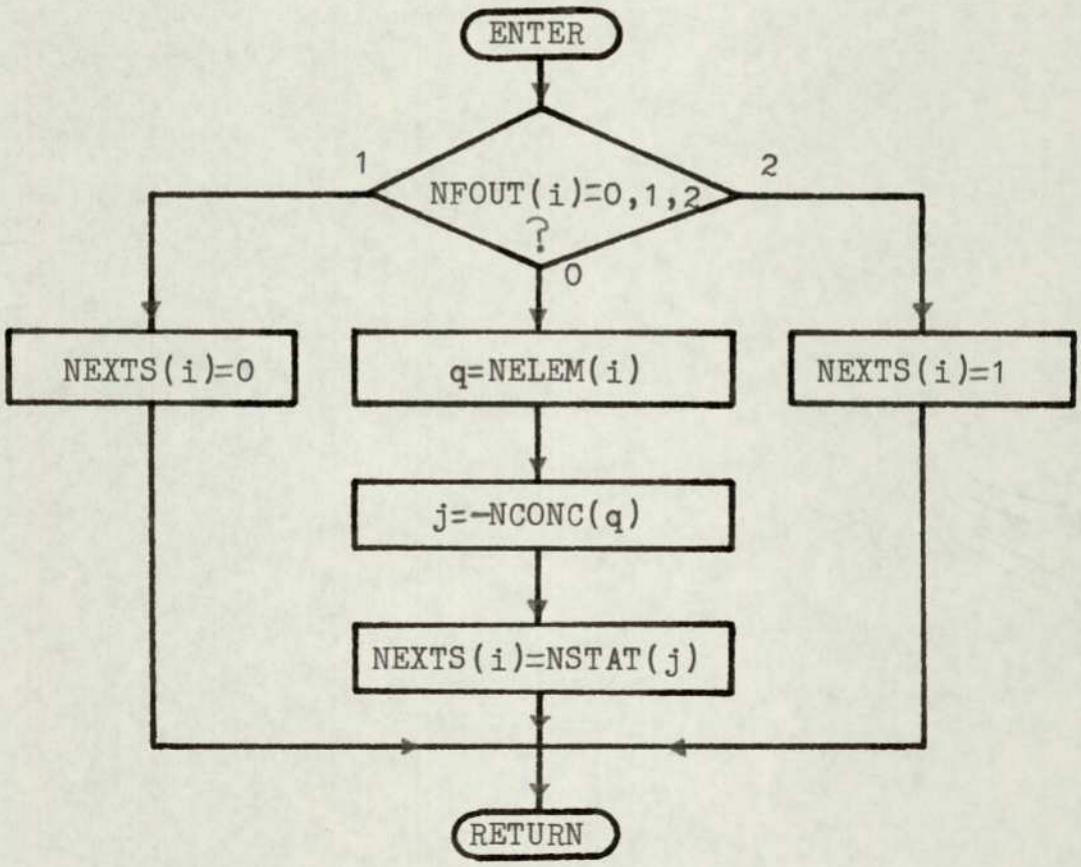


FIG. 2.15.(c). MODIFIED 'OUTPUT' FUNCTION

correctly working circuit and the faulty circuit, was also implemented in FORTRAN IV.

The first type of simulator required approximately 20 computer words of storage per circuit element for the structural and functional description of a circuit, including the fault parameter data. The second type, although in need of no structural data, required a variable number of computer words dependant on the number of circuit inputs (i.e. n inputs give 2^n tests) and the number of faults in the circuit. So, the same number of elements with a larger number of inputs requires greater storage space.

The speed of simulation is defined by the type of implementation, the language used, the use to which the simulator is being put and the speed of the host computer. It is therefore difficult to make a statement on the speed of simulation for comparative evaluation against other simulators. The simulation subroutine, when used in the construction of testing schedules (a process requiring much analysis external to the actual simulation), performed approximately 100 simulations per second on average. The table look-up simulation, used for test schedule construction of combinational circuits, performed at an average rate of 1,150 simulations per second. In both cases the host computer was an ICL 1905 computer.

2.5.2 Limitations

Two main limitations of the simulator implementation described in this section can be identified:

(a) The high level nature of FORTRAN IV limits its

efficiency for this kind of use. FORTRAN is not designed specifically to deal with the requirements of bit and logical manipulations that are performed during simulation. This results in inefficient use of storage. For example, a logic level (0 or 1) uses a complete 32 bit computer word. Also the speed of processing is low.

(b) Only a limited number of types of logic element are used, and these are at the lowest possible level; AND, OR, etc. This does not reflect the use in modern systems of M.S.I. and L.S.I. components, where quite complex circuits are integrated into a single component assembly.

These limitations are minimised to some extent by the nature of the project in which the simulator is used. Speed and storage restrictions are not too important since the size of circuit under consideration is quite small. Also, for the problem to be investigated with limited available time and support, the restriction of considering simple logic elements only is justifiable.

2.5.3 Developments

For use in wider applications the simulator subroutine must be extended in speed and storage capabilities. This could be done by using the computer machine language to allow the packing of circuit functional, structural and operational data at the bit-level in the computer store words. Using the comprehensive bit manipulation and logical instructions available at this programming level, easy

coding and packing and unpacking of circuit data is permitted. Increase in the simulation speed results from the circuit structure and function being modelled with a language working at a comparable level to the logical operations and signal flows in real circuits.

The basic simulation process can be re-designed to permit a more realistic modelling of logic circuit operation. One possible development is the use of three-level simulation (75), (76), where the normal logic levels (0,1) have an added possibility (u) corresponding to the period of state uncertainty during transition from one level to another. The state u exists whenever a transition 0(1) to 1(0) occurs, for a defined period of uncertainty. Additional logic rules are required:

(a) $\bar{u} = u$

(b) $0 + u = u$

(c) $1.u = u$

(d) $0.u = 0$

(e) $1 + u = 1$

(f) $u . u = u$

(g) $u + u = u$

Such a simulation process can be used to detect hazards in combinational and sequential circuits (75). If, when simulating a sequential circuit, equilibrium is reached with one or more stored states in the u condition, a critical race is detected.

Other functional elements commonly found in logic circuits could be added to the function list of the simulator. For example, EXCLUSIVE-OR, MONOSTABLE, FLIP-FLOP

and FREE RUNNING MULTIVIBRATOR. When the subroutine was in development several versions were written. One included such functions, and preliminary investigations indicated that it worked successfully. However, time for further development was not available.

3. DIAGNOSIS OF INTERMITTENT FAULTS IN COMBINATIONAL LOGIC CIRCUITS

Several different techniques have been developed in the past for determining input tests which are sensitive to solid faults in combinational logic circuits. Various criteria have been used in evaluating these tests and incorporating them in efficient fault detection and location testing schedules.

Two aspects of test selection and schedule construction for the diagnosis of intermittent faults in combinational circuits are investigated here:

(i) Interpretation of the solid fault test schedule data produced in the presence of intermittent faults as an aid in locating such faults.

(ii) Formation of test schedules designed specifically for intermittent fault diagnosis.

This investigation involved the development of new schedule formation and diagnostic data interpretation techniques and the writing of several computer programs to implement them.

3.1 INTROUDCTION

One aspect of the maintenance of circuit and system operating integrity is to use of testing schedules to allow detection and location of malfunctions. The classical form of testing schedules commonly used can be essentially divided into two types, combinational and sequential. These terms refer not to the type of circuit but to the form of the testing process. Both processes utilise series of tests. A test, in the context of detection and location, refers to a circuit input stimulation (a set of logic levels at the circuit inputs) which induces an output reaction, the form of which gives an insight to the nature of a circuit malfunction. It is the order in which tests are applied and the process of interpreting the output results that defines the testing procedure as being combinational or sequential.

In a combinational process all the defined tests are applied in a fixed sequence. The resulting output responses are recorded and analysed when the testing sequence is finished. Fig.3.1(a) illustrates how the three input test T_1,T_2 and T_3 can be used to locate any of the circuit faults F_1,F_2,.....,F_6. If the series of recorded output responses is Z_3,Z_3,Z_2, the malfunction is F_3. The set of faults (F_2,F_5) cannot be resolved by this diagnostic process and is termed an indistinguishable fault set. It is common practice to include the correctly working circuit in a testing schedule. This might be F_1 in the example. Since T_1,T_2 and T_3 are always used, the average test length is 3.0

In the sequential testing procedure the sequence of test is not fixed. A subsequent test is selected on the basis of the output response of the circuit to the current input test. This

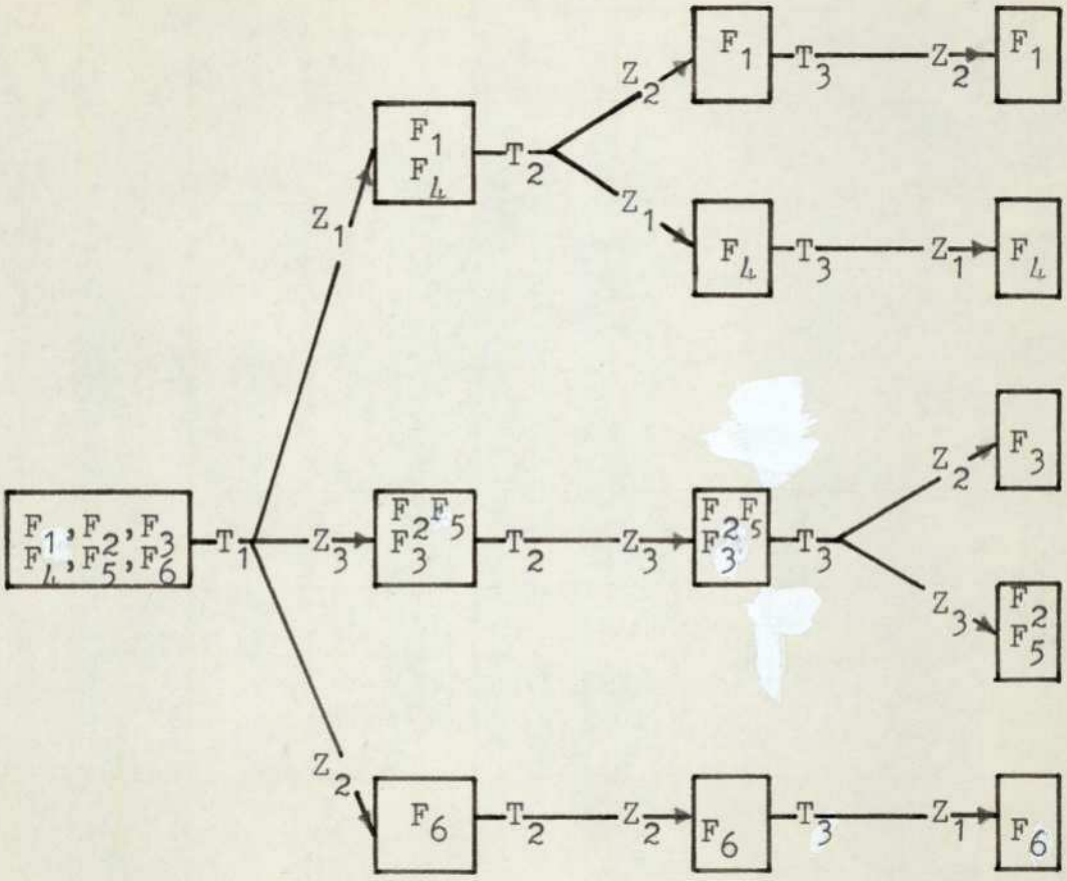


FIG. 3.1.(a). COMBINATIONAL TEST SCHEDULE EXAMPLE

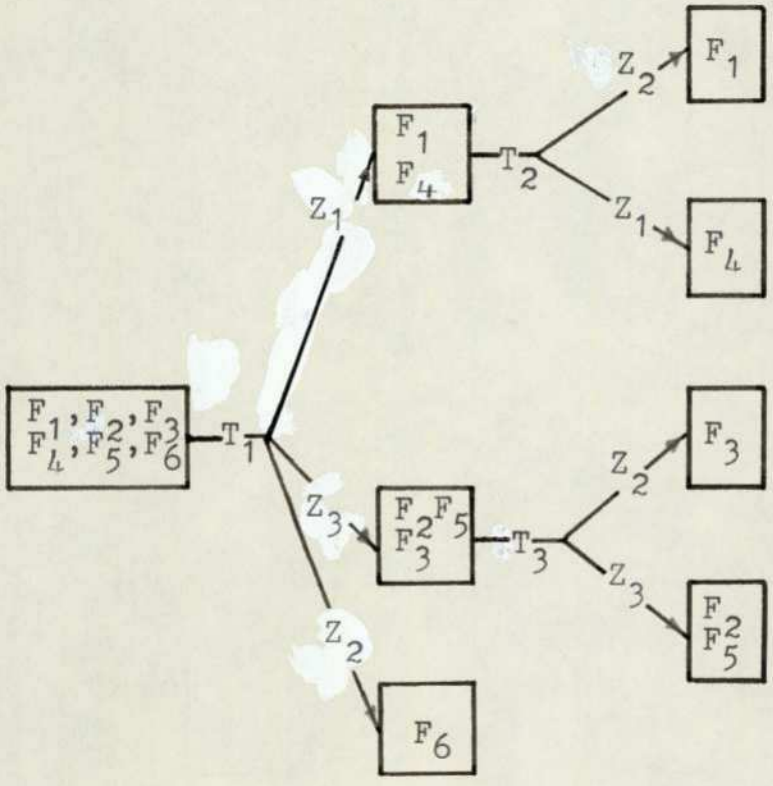


FIG. 3.1.(b). SEQUENTIAL TEST SCHEDULE EXAMPLE

permits the removal of tests which are redundant for certain faults, thus shortening the average test length. Fig.3.1(b) shows how the same example may be treated in this process. F_3 can be diagnosed in a two test sequence by applying T_3 after an output response Z_2 has been obtained from T_1. Assuming an equal probability of fault occurrence the average test length becomes 1.83, a considerable reduction on the combinational process.

The above example has illustrated the process of fault location. In some cases, such as with MSI or LSI components, it is sufficient to just detect that a fault is present and simply replace the component. Since the required diagnostic resolution is low the testing schedule can be much shorter.

The sequential process is adopted for most diagnostic uses (38) because of the reduction in average testing time it offers. However, the amount of information to be stored by an automatic testing machine is rather greater than for the combinational process. Also the decision making abilities of the tester have to be rather more sophisticated. The sequential process was chosen for use throughout this project.

This section of the thesis will concentrate on combinational circuits. A combinational circuit is one where the present output state is a direct function of the present input stimulation only. This can be represented by:

$$Z = A(X)$$

where Z is the circuit output, $(z_1, z_2, \ldots\ldots, z_m)$, X is the input stimulation, $(x_1, x_2, \ldots\ldots, x_n)$, and A is the logical function performed by the n-input and m-output circuit. Sequential

circuits, with their associated stored states, represent a rather more difficult problem and will be considered separately in a later section.

The main objective of this section of the thesis is to describe the formulation of sequential testing schedules to be used for diagnosing intermittent faults in combinational circuits. The problem is examined in two ways:

(i) Consideration is given to the utilisation of ordinary solid fault testing schedules in diagnosing intermittent faults. This requires the formulation of a technique for interpreting the diagnostic information produced by such schedules when used in the presence of this type of fault. The justification for such an investigation is that the investment of D-A time to produce solid fault schedules is not wasted and that it can only enhance and extend the diagnostic capabilities of such schedules.

(ii) Techniques are developed for constructing test schedules for specifically diagnosing intermittent faults.

Sub-section 3.2 introduces the basic principles involved in test schedule construction related to solid faults. It concentrates particularly on the various processes of test generation and the way these tests are selected and formed into efficient schedules using certain selection criteria. The use of the solid fault schedule for diagnosing intermittent faults is discussed in Sub-section 3.3. The construction of schedules specifically for diagnosing intermittent faults is described in Sub-section 3.4. This is followed by a general discussion in

Sub-section 3.5.

3.2 TEST GENERATION AND SCHEDULE CONSTRUCTION FOR THE DIAGNOSIS OF SOLID FAULTS IN COMBINATIONAL CIRCUITS

3.2.1 Test Generation

The problem of diagnosing faults in logic circuits can be divided into the two areas of:

(i) Generating a test(s) which will detect the presence of a given fault in a circuit and determination of other faults detected by the same test(s).

(ii) Applying some form of selection criterion for choosing, from a set of possible tests, a minimal (or near minimal) sequence which will give maximum (or near maximum) diagnostic resolution.

The first area will be considered in this sub-section.

Test generation is a topic that has been comprehensively reviewed by several authors in recent years, (54),(28), (29) and (55), and will only be briefly described here.

(a) FAULT MATRIX

The most simple technique is to initially define the test set as the complete set of all possible input tests. For an n input combinational circuit these would number 2^n. The effect at the circuit output of the presence of each of the p circuit faults in the presence of each input test can be represented in a Fault-Matrix (37),(41). This consists of a (2^n) x (p+1) matrix, the (i,j)th element of which contains the circuit output response in the presence

of input test number i and fault j. j=1 corresponds to the correctly working circuit.

Redundant data can be removed from the fault-matrix to reduce the required storage size as follows:

(i) remove rows corresponding to those tests which produce no diagnostic data. That is, where row elements (i,j), j = 2,3,.....,p, are all equal to elements (i,1).

(ii) remove rows corresponding to those tests which produce identical diagnostic data. That is, remove row i whose elements (i,j), j = 1,2,.....,p, equal the elements (k,j) of another row k.

(iii) remove columns corresponding to those faults which are indistinguishable from other circuit faults. That is, remove (fault) column j whose elements (i,j), i=1, 2,.....,2^n, equal the elements (i,ℓ) of some other (fault) column ℓ.

The data for such a fault matrix can be generated by simulating the circuit in the presence of each fault with each input test applied. This data is recorded and simplified as indicated above. The tests remaining after the simplification process represent an exhaustive (not minimal) test set. The selection of a subset of these test to form efficient testing sequences is discussed in the following sub-section 3.2.2.

The storage requirements of the fault-matrix limit its use to small size circuits with small fault sets and a small number of inputs. A circuit with 10 inputs and 100 faults would initially require a 103,424 element storage matrix.

(b) ELDRED'S RESULTS

Eldred (77) published the first accounts of test generation for combinational circuits in 1959. Although the work is not appropriate to modern requirements, being applicable to one or two level logic circuits, it represents the first steps in hardware testing techniques.

Eldred used the circuit diagram as the starting point. He noted that a two input OR gate could be tested for inputs s-a-0 faults by the tests (1,0) and (0,1), and that these also correspond to a check for output s-a-0. A diagnostic result is obtained when an inappropriate output is generated for a given input test(s). Also, the test (0,0) could be used to check for the faults of inputs s-a-1 and output s-a-1, but is not be able to distinguish between them. With this knowledge the approach can be generalised to deal with any type or number of gates in a one or two level configuration. By choosing sets of tests which made sure every circuit elements was exercised in the appropriate way, Eldred was able to claim complete diagnostic coverage for the test set.

(c) ONE-DIMENSIONAL PATH SENSITISING

This is an extension of Eldreds work to deal with more than one or two levels of logic. The technique was described and named by Armstrong (31). Essentially the process works by finding a path, between the location of a fault and an output, which can be sensitised by setting various circuit input conditions. The state of the output

then reflects the presence or absence of the fault.

Determination of sensitive paths is as follows:

(i) Assign a logic level at the fault location opposite in value to that expected in presence of the fault.

(ii) Choose a path from the fault location to a circuit output.

(iii) The inputs to gates along this path are assigned logic levels so as to propagate any change due to the occurrence of the fault along the path to the output. The path is then sensitised to the fault.

(iv) The input test required to detect the fault is generated by a backward trace from the assigned gate logic levels to the inputs, setting logic levels appropriate to the required gate conditions. If a logical contradiction is encountered another set of levels is attempted. If this does not produce a solution a different sensitised path is selected and the backward trace is repeated.

Through an example Schneider (78) has shown that in some cases one-dimensional path sensitising fails to produce a test for a given fault when one does exist. This is a result of reconvergence of fan-out paths (31), and has limited the usefulness of the one-dimensional technique.

(d) THE d-ALGORITHM

The problems of reconvergent fan-out can be removed if all possible paths from the site of a fault can be

sensitised simultaneously. This two-dimensional path sensitising was first formulated by Roth (32) in 1966 and is termed the d-Algorithm.

Essentially the process of test generation is as follows:

(i) Generate all possible paths from the fault site to all circuit outputs simultaneously. Cancel all paths that form reconvergences of fan-out. This operation is termed the d-drive.

(ii) Using a backward trace process construct consistent input tests which realise all the appropriate gate logic level conditions required by the d-drive. This is called the consistency operation.

The enumeration of all paths ensures that if a test for a particular fault exists it will be found. The calculus by which the process is implemented uses a circuit element model similar to a truth table. One extra symbol is used, the d-notation. d represents a signal that is a logic level 1 in the good circuit and 0 in the faulty circuit. $\bar{d}$ represents the opposite case of a 0 level which becomes 1 in the faulty circuit. These definitions are interchangable, but once defined apply throughout the circuit.

A primitive d-cube of a failure is defined as the expression of the tests for the logic element input failures in terms of the inputs and outputs of that element. The propagation d-cubes for an element indicate the ability of elemental input lines to control elemental

outputs. Using the primative d-cube of failure and the propagation d-cubes an attempt is made to d-drive from the failure site to the outputs. The d-drive consists of a series of d-intersections where propagation d-cubes are intersected with the primative d-cube of failure to calculate the propagation of the effects of the fault to the outputs. The intersection calculation follows well defined rules (32). The sensitivity of the circuit to the fault is represented by the propagation of d or $\bar{d}$ to the sensitive outputs.

Roth, et al, have shows how, having determined a test for a given fault, examination of the associated sensitised paths can allow the other faults detected by the same test to be easily evaluated (79). The d-calculus need not be performed on these fault conditions since a detection test has now been found. However, such an approach may not give a sufficiently comprehensive test set for high diagnostic resolution if fault location is required, even though, as pointed out by Chang, et al, (28), efficient fault detection tests often turn out to be efficient for diagnosis as well.

(e) EQUIVALENT NORMAL FORM

Armstrong (31) considered the problem of constructing efficient (near minimal) test sets and developed a method known as the equivalent-normal-form (e.n.f.). The e.n.f. for a simple single-output circuit is constructed as follows:

(i) Describe the circuit by a Boolean expression. Corresponding to each gate G_j in the circuit there is a pair of parentheses in the expression, these being labelled with the symbol j. The sub-expression corresponding to G_j is enclosed in the parentheses. All literals in the Boolean expression denote input variables.

(ii) Expand to sum-of-products form. When a pair of parentheses is removed the associated label is attached to each literal inside the parentheses as a subscript. No redundant terms are removed. The e.n.f. is the sum-of-products resulting from this operation. Each literal appearance corresponds to a path between an input and output specified by the string of subscripts.

The e.n.f. is used in the following manner, based on a theorem due to Armstrong (31): "A test for a literal appearance in the e.n.f. sensitises the corresponding path in the circuit. So, if a set of literal appearances can be selected whose corresponding paths together contain each element of the circuit, and if a set of tests can be found which checks at least one appearance of each literal for the s-a-1 and s-a-0 faults, then the set of tests detects every fault of the circuit".

This method can be computer programmed, but the problems of storage and computation times become unmanagable for large and complex circuits. The most important feature of this technique is the automatic recognition of all faults detected by the same test.

It is worth mentioning here the work of Poage (35) in producing minimal sets of tests to detect all circuit faults. The key feature of this work was a special calculus developed to describe the functions realised by the circuit under the failure conditions. This required the definition of Boolean fault parameters to be associated with each fault connection in the circuit. The reader is referred to references (28) and (29) for illustrative descriptions of this technique. Generally, this method is impractical for large circuits because of excessive calculation times and storage problems.

(f) BOOLEAN DIFFERENCES

This is a technique by which circuit tests can be derived directly from the Boolean equations representing the circuit functions. Given a circuit of function $F(x_1,x_2,.....,x_i,.....,x_n)$, where each x_i represents a Boolean input variable, in order to observe a failure in that function, a valid term with one of its variables (x_i) inverted must cause the complement of $F(x_1,x_2,.....,x_i,.....,x_n)$ to be produced. That is, the fault must be detectable at a circuit output. The existence of a test for the fault affecting variable x_i is determined by the simple rule:

$$F(x_1,x_2,.....,x_i,.....,x_n) \oplus F(x_1,x_2,.....,\overline{x}_i,.....,x_n) = 1$$

where $\oplus$ defines the EXCLUSIVE-OR operation. This is called the Boolean Difference (33) and is commonly denoted as $\frac{d}{dx_i}$. So, a test for a fault affecting variable

x_i exists if $\frac{dF(x)}{dx_i} = 1$ and does not exist if $\frac{dF(x)}{dx_i} = 0$. The set of tests for a fault on x_i can be found by calculating:

$$x_i . \frac{dF(x)}{dx_i} \quad \text{for } x_i \text{ s-a-0}$$

and

$$\bar{x}_i . \frac{dF(x)}{dx_i} \quad \text{for } x_i \text{ s-a-1}$$

for $i = 1,2,.....,n.$

Various simplifying operations exist for allowing Boolean differences of complex circuits to be represented in terms of the Boolean differences of simpler circuits.

It is debatable whether this method of test generation offers any advantages over the d-Algorithm because of the large amounts of computer time required for repeated evaluations of the Boolean equations and the storage problems (1). Bennetts (80) has suggested that a compromise between Boolean and path sensitising techniques can enhance the Boolean Difference method.

Because of the length of time and the programming effort required to construct computer programmed implementations of most of these test generation techniques it was decided to utilise the most simple method for combinational circuits, the Fault Matrix. This choice is justifiable since only small circuits are to be considered, and also there is a logic simulator available for constructing the fault matrix.

3.2.2 Formation Of Efficient Diagnostic Test Schedules

Having generated a set of fault detection tests by one of the techniques described in the previous sub-section (3.2.1), the next problem is to choose from the this set the smallest possible subset that will permit any of the faults to be detected or diagnosed.

Firstly, brief consideration is given to the construction of combinational types of schedule. Assume that the initial test data is formed as a Fault-Matrix. Various refinements are made to this data, such as removing redundant tests and recognising indistinguishable fault sets. Fault detection is solved by finding the minimal number of rows (tests) such that each column (fault) is detected as being different from column #1 (the correctly working circuit) at least once. The diagnosis problem of distinguishing all faults from each other, and from the correctly working circuit, is solved by determining the minimal number of rows which satisfy the detection requirement above and which has all columns distinguishable from all others. Exhaustive enumeration would produce solutions for both these problems but is costly in terms of time for anything other than small circuits. Using the similarity of finding minimal test sets to problem of finding minimal covering in switching theory, a technique such as the Quine-McClusky (81) algorithm can be used. Kautz (82) and Poage (35) have developed such techniques and obtain minimal test sets. Another method is to pre-process the fault-matrix to a reduced form using the method of 'test set intersection' (36). This does not guarantee

minimal test sets. A third solution, due to Chang (37), uses a distinguishability criterion which assigns weights to the various tests relative to their ability to distinguish between faults. This frequently produces near minimal test sets. The distinguishability criterion will be described later in more detail during the sequential schedule formation discussion. The tests produced by these methods are applied to the circuit under test in any order. An inconsistent output or a particular pattern of inconsistency detects or diagnoses a fault.

In sequential schedules the selected tests are placed in a particular order so as to aid in the reduction of testing time. The selection of tests and the ordering process make exhaustive enumeration to find a minimal implementation even more impracticable than with combinational schedules. This is because all combinations of ordering for each permutation of available test set would have to be evaluated. For this reason local rather than global optimisation test selection procedures are used.

The object of a fault detection schedule is to utilise the smallest number of tests which identify the correctly working circuit. Any deviation from the schedule indicates the presence of a fault and testing can cease. A criterion for choosing tests has been suggested by Seshu and Freeman (38). This is to select at each point in the procedure that test which detects the greatest number of hitherto undetected faults. This is repeated until all faults are detected. The figure of merit

defined for this selection criterion is used in the following manner:

(i) α_j, the figure of merit, is calculated for each of the remaining tests. $\alpha_j = (N_b - N_a)/N_b$ for the jth test, where N_b is the number of undetected faults before applying test T_j, and N_a is the number remaining undetected after applying T_j.

(ii) Select the T_j with the largest α_j.

(iii) Repeat from (i) until all faults are detected.

Brule, et al, (39), have used the concept of information gain as a figure of merit for choosing tests suitable for diagnosing faults. Essentially, the process works by choosing at each point that test which gives maximum gain for the remaining undetected faults. If p_q is the probability that fault f_q will occur, the total level of uncertainty as to which of the faults will be present is given by (83):

$$A = -\sum_q p_q \log_2 p_q . \quad \text{——} \quad \text{——} \quad \text{——} (i)$$

The application of a test removes some of this uncertainty, which is equivalent to information being gained. The next best test is therefore the one which gives the highest average information gain. Mandelbaum (84) and Seshu and Freeman (38) have implemented figure of merit calculations based of this concept.

A further technique known as the distinguishability criterion was introduced by Chang (40). This is best described by a simple example. Fig.3.2 illustrates the branch point at the (i,x)th level of the test diagram,

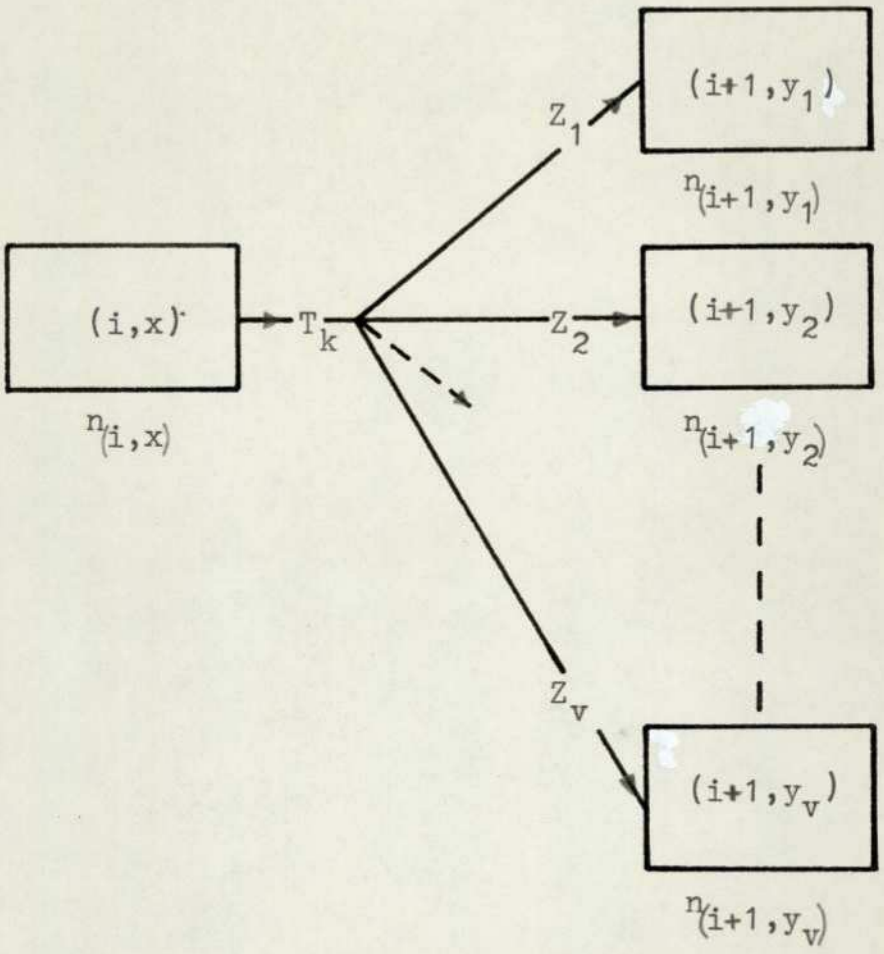


FIG. 3.2. TEST DIAGRAM BRANCH POINT

consisting of $n_{(i,x)}$ faults, at which test T_k produces an increase in diagnostic resolution. This results in the division of the $n_{(i,x)}$ faults into v new partitions $(i+1,y_1)$, $(i+1,y_2)$,.....,$(i+1,y_v)$, containing $n_{(i+1,y_1)}$, n_{i+1,y_2},.....,$n_{(i+1,y_v)}$ faults, and defined by the output responses $z_1,z_2,.....,z_v$, respectively. To maximise diagnostic resolution T_k is chosen as that test which distinguishes the greatest number of pairs of faults from each other. In partitioning (i,x) into v new partitions T_k distinguishes every fault in $(i+1,y_s)$ from every fault in $(i+1,y_t)$, $s \neq t$. The total number of pairs distinguished among the v new partitions by T_k is therefore:

$$\alpha_k = \sum_{s=1}^{(v-1)} \{n_{(i+1,y_s)} \cdot \sum_{t=(s+1)}^{v} n_{(i+1,y_t)}\} \quad --- (ii)$$

α_k is a maximum when T_k identifies all faults uniquely. It is a minimum (=0) when no faults are distinguished (v=1). Chang (40) has extended this to deal with diagnosis at the replaceable package component level. This is useful for M.S.I. and L.S.I. implementations.

The distinguishability criterion was chosen as the technique to be used for test selection in this project because of the ease in which α_k could be calculated and by which the process programmed. It should be noted that the local optimisation nature of this technique does not guarantee minimal length diagnostic sequences, but near minimal results are normally obtained in practice (28).

3.2.3 Implementation Of Schedule Formation Process

The distinguishability criterion procedure is used in this project as the method of test selection for diagnostic schedule construction. Circuit test data is supplied in the form of a Fault-Matrix. This matrix could be formed by simulating each fault condition in the presence of each possible input test. The Fault-Matrix can be pre-processed to removed redundant tests and indistinguishable fault sets. Column #1 of the matrix always contains the correctly working circuit output responses.

The schedule construction process is as follows:

(i) Form the Fault-Matrix, removing redundant tests and indistinguishable faults.

(ii) Define the current partition of faults as the complete set of circuit faults (or indistinguishable fault sets).

(iii) Apply the distinguishability criterion to select which of the 2^n possible input tests produces the maximum diagnostic resolution (i.e. greatest α). This is done by calculating α_j, $j=1,2,.....,2^n$, according to the formula defined in 3.2.2, by examining the Fault-Matrix for each of the current partition fault conditions, for each test T_j.

(iv) Examine α_j, $j=1,2,.....,2^n$ and select the largest value (α_ℓ). T_ℓ is then the best possible next test.

(v) Use the ℓth row of the Fault-Matrix to create the new partitions that result from applying T_ℓ to the

circuit in the presence of those faults defined in the current partition. The value and test number of T_ℓ, the new partition numbers, and the circuit output responses associated with them, are recorded. If no $\alpha_j > 0$, then the current partition of faults are indistinguishable. This set is given a new fault number and recorded as having been diagnosed. If any newly created partition contains only one fault, this fault is uniquely identified and is recorded as having being diagnosed.

(vi) Select one of the newly created partitions and call it the current partition. If all new partitions have been processed then schedule construction is finished. Otherwise, repeat from (iii).

The test schedule data can be recorded in a simple tabular form. This consists of (2+z) columns and y rows, where z is the number of different possible circuit output responses and y is the number of tests used in the schedule. The first element of each row contains the test (or partition) number (1,2,.....,y). The second contains the value (in decimal) of the associated input test. The following z elements $(3,4,.....,(z+2))$ corresponding to the possible circuit output responses $(\text{which in decimal are } 0,1,2,.....,(z-1))$:

(a) pointers to the next test number in sequence to be accessed.

or (b) the fault number of any fault diagnosed by the present circuit test output response. (This is given a negative number to allow identification from the ordinary pointers in (a) above).

or (c) a zero entry. This corresponds to an output which under normal fault conditions could not be obtained with the given input test.

Such a schedule would be used by starting at the first test, applying the corresponding test value to the circuit and examining the output response. The column of the schedule appropriate to this output is accessed to determine the next operation, either pointing to a subsequent test number or termination of testing due to a diagnostic result being obtained.

The schedule formation procedure was implemented as a FORTRAN IV program named SCHED. A flow diagram illustrating the program is shown in Fig.3.3. The arrays and variables named in this diagram have the following meanings and uses:

NFALT(n,m) - a two-dimensional array containing the Fault-Matrix data on the n possible tests and the m circuit faults. (Note: test values range from 0 to (n-1)).

LISTT(x) - a one-dimensional array used for listing the members of the various fault partitions.

NPART(y) - a one-dimensional array for storing the address (x) in the array LISTT where the yth partition is stored.

NCURR - an integer counter used to store the number (y) of the current partition being processed.

NEXTS - an integer counter used to store the number (y) to be assigned to the next newly created partition.

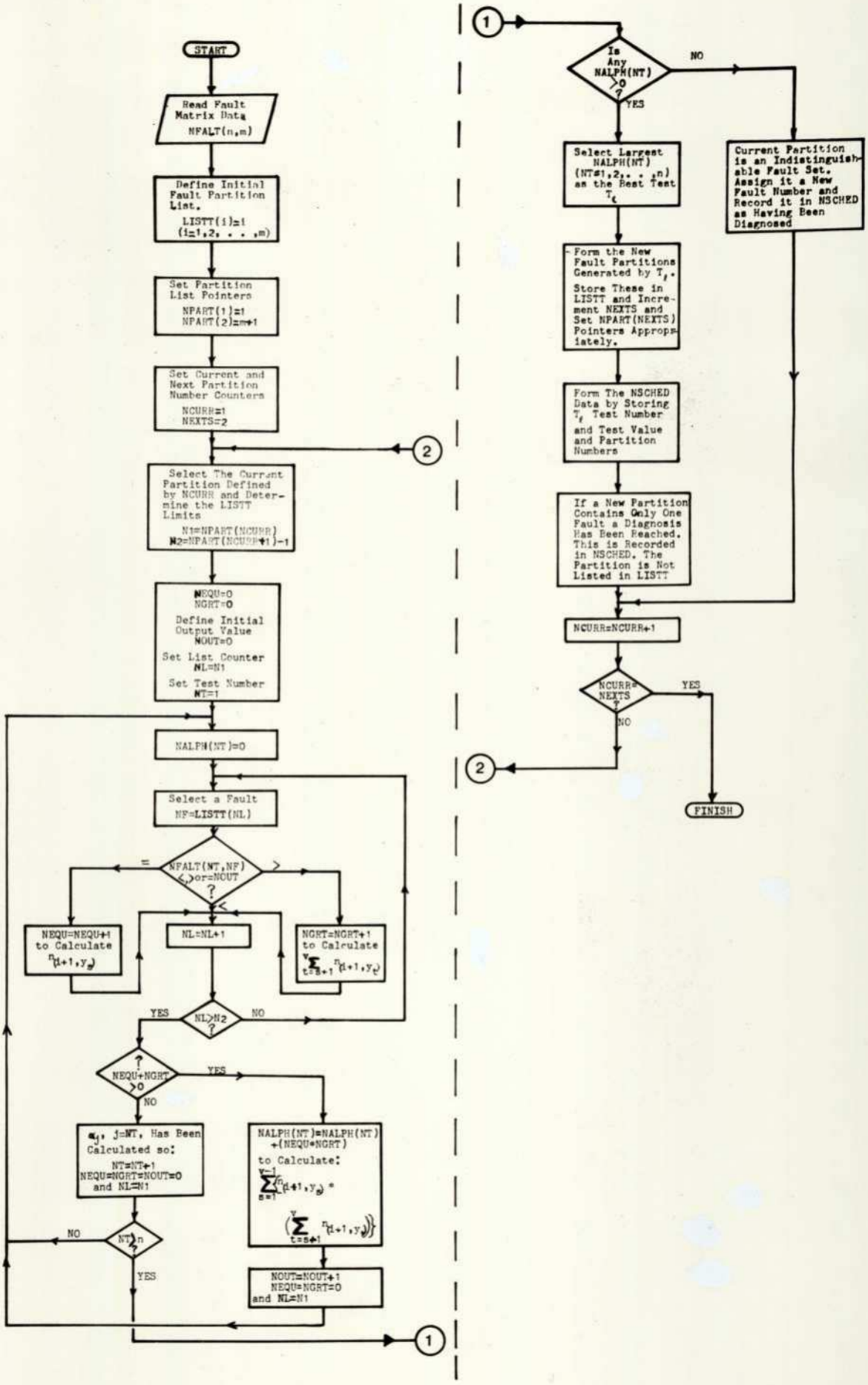


FIG. 3.3. FLOW DIAGRAM OF PROGRAM 'SCHED'

NALPH(z) - a one-dimensional array for storing the figures of merit α_z, z=1,2,.....,n, generated by the n input tests.

N1, N2 - are used to store the addresses in LISTT between which the current partition data is stored.

NL - is used as an address counter for pointing to the location of the fault in LISTT under present consideration.

NF - the number of the fault under present consideration (NF = LISTT(NL)).

NT - an integer variable which takes on a value between 1 and n corresponding to one of the possible input tests. (Note: (NT-1) corresponds to the actual test value.)

NOUT - defines the circuit output value under present consideration.

NEQU,NGRT - are used to count up the number of faults which produce equal and greater (respectively) output responses for a particular input test.

NSCHED(i,j) - a two-dimensional array for storing the schedule data, the ith row holding the current (NCURR) test number, the test value (NT-1), and the pointers to the next (NEXT) tests or diagnostic results (-NF).

3.2.4 A Circuit Example

Consider the full-adder circuit shown in Fig.3.4. This was simulated using the SIMULATION subroutine (see

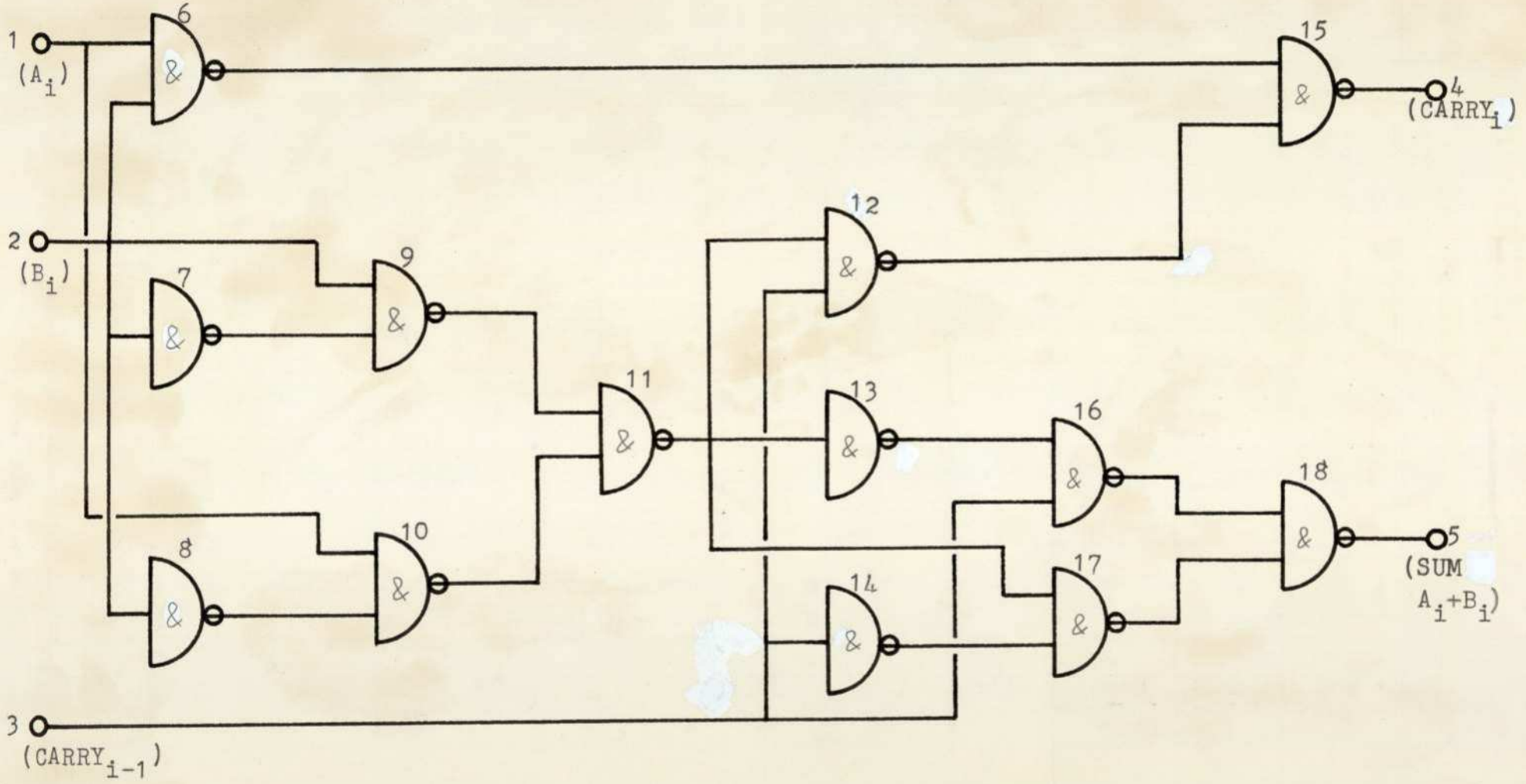


FIG. 3.4. A CIRCUIT EXAMPLE (FULL-ADDER)

Section 2) in the presence of each of the 81 possible circuit faults and 8 input tests. After identifying the indistinguishable faults sets, only 27 faults (or fault sets) remained. This is shown as the Fault-Matrix of Fig.3.5(a). The faults associated with each of the fault numbers are shown in Fig.3.5(b), using the previously defined description format (2.4.1).

The Fault-Matrix was supplied as data to the program SCHED. The resulting testing schedule is shown in Fig.3.5 (c). This took two seconds of computer time, including program compilation, on an ICL 1905 computer. More insight into the testing procedure may be gained by examining the testing diagram of Fig.3.6, which shows the actual partitioning at each test. The diagnosis of the correctly working circuit (shown as (-1)) is obtained by the sequence of schedule test numbers 1,4,9,15 and 18, which correspond to test values 1,6,4,3 and 2 and outputs 2,1, 2,1 and 2, respectively.

The average test length for the schedule is calculated by summing the number of tests in the diagnostic sequences of each fault and dividing by the total number of faults. This comes to a value of 3.54 tests. The calculation assumes each fault has equal probability of occurrence.

3.3 INTERPRETATION OF SOLID FAULT DIAGNOSTIC TEST SCHEDULE DATA FOR DIAGNOSING INTERMITTENT FAULTS

3.3.1 The Effects Of Intermittent Faults On Schedule Data

Having constructed a diagnostic test schedule for

Test No.	Test Value	Output 0	1	2	3
1	1	2	3	4	5
2	2	6	0	7	0
3	2	0	-27	-25	0
4	6	8	9	10	11
5	0	12	-14	0	0
6	4	-21	0	13	0
7	3	0	-7	-22	0
8	3	-18	-11	0	0
9	4	-16	14	15	-12
10	2	16	0	-26	0
11	0	-15	0	17	0
12	2	0	0	-4	-13
13	5	0	-17	-10	0
14	3	0	-8	0	-9
15	3	-2	18	0	19
16	3	0	-6	-24	0
17	2	0	-23	-20	0
18	2	0	0	-1	-3
19	0	-5	0	-19	0

FIG. 3.5.(c). CIRCUIT TEST SCHEDULE

TEST No.	Val	FAULT No. 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	0	0	0	0	0	0	0	2	2	0	0	0	0	1	0	0	0	0	2	2	0	0	2	0	2	0	2
2	1	2	2	2	3	2	2	0	2	2	0	2	2	3	3	2	2	0	2	2	2	0	0	2	2	1	2	1
3	2	2	2	3	2	2	0	2	2	2	0	2	2	3	3	2	2	0	2	2	2	0	2	1	0	2	2	1
4	3	1	0	1	1	3	1	1	1	3	1	1	1	1	1	1	1	1	0	3	3	1	2	1	2	1	1	3
5	4	2	2	2	2	2	2	2	1	1	2	2	3	2	3	2	0	2	2	2	2	0	2	1	2	1	0	2
6	5	1	1	1	1	1	1	2	1	1	2	0	1	1	1	3	1	1	0	1	3	1	2	1	1	3	2	1
7	6	1	1	1	1	1	2	1	1	1	2	0	1	1	1	3	1	1	0	1	3	1	1	3	2	1	2	1
8	7	3	2	3	3	1	3	3	3	1	3	3	3	3	3	3	1	3	2	3	3	1	1	3	1	3	1	3

CIRCUIT OUTPUT RESPONSES TO INPUT STIMULATIONS

FIG. 3.5.(a). CIRCUIT FAULT-MATRIX

Fault No.	Fault Description
1	0 0 0
2	15 6 1
	6 0 1
	6 2 1
	6 1 0
3	6 1 1
4	6 2 1
5	9 7 1
	7 0 1
	7 1 0
	10 8 1
	8 0 1
	8 2 0
6	11 9 1
	9 0 1
	9 7 0
	9 2 0
	7 0 0
	7 1 1
7	11 10 1
	10 0 1
	10 8 0
	10 1 0
	8 0 0
	8 2 1
8	9 2 1
	10 1 1
9	11 0 1
	11 10 0
	11 9 0
	9 0 0
	10 0 0

(cont.	
10	11 0 0
11	15 12 1
	12 0 1
	12 3 0
	12 11 0
12	12 11 1
13	12 3 1
14	4 0 1
	15 0 1
	15 12 0
	15 6 0
	6 0 0
	12 0 0
15	16 13 1
	13 0 1
	13 11 0
	17 14 1
	14 0 1
	14 3 0
16	18 16 1
	16 0 1
	16 3 0
	16 13 0
	13 0 0
	13 11 1
17	18 17 1
	17 0 1
	17 14 0
	17 11 0
	14 0 0
	14 3 1

(cont.	
18	4 0 0
	15 0 0
19	16 3 1
	17 11 1
20	5 0 1
	18 0 1
	18 17 0
	18 16 0
	16 0 0
	17 0 0
21	5 0 0
	18 0 0
22	1 0 0
23	1 0 1
24	2 0 0
25	2 0 1
26	3 0 0
27	3 0 1

=Indistinguishable Fault Set

FIG. 3.5.(b). CIRCUIT FAULT DESCRIPTIONS

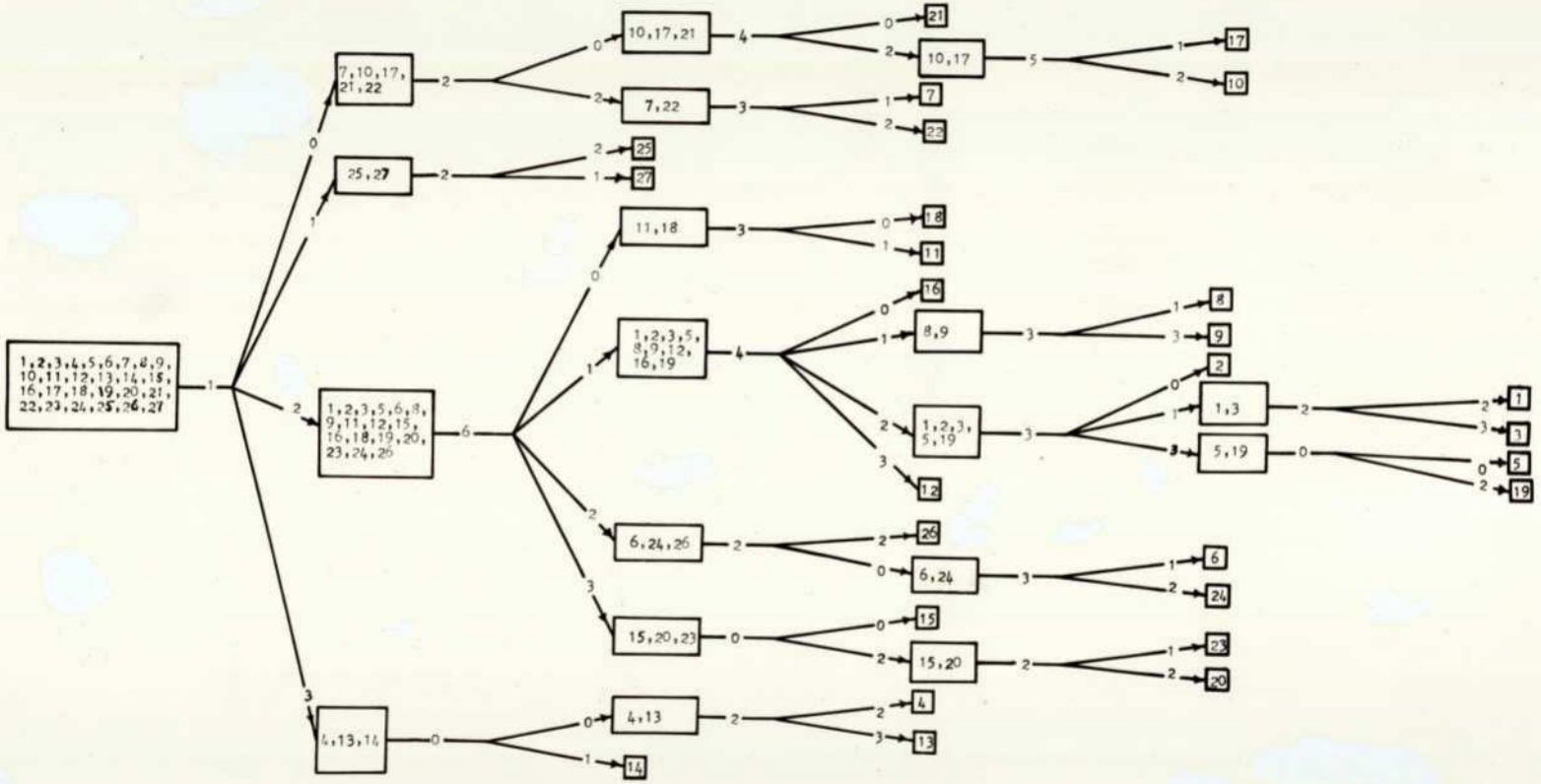


FIG. 3.6. TESTING DIAGRAM FOR FULL-ADDER CIRCUIT

locating solid faults the next step is to determine how this schedule reacts in the presence of intermittent faults. In this context an intermittent fault is defined as a logical fault of the s-a-0 or s-a-1 variety that appears and disappears in an intermittent pattern. It is assumed that the change of state from one condition to another is instantaneous with no intermediate condition.

Breuer (58) uses a restricted definition of fault intermittency, which is also used in this project, to simplify analysis problems. The intermittent fault is defined to be *well-behaved*. For combinational circuits a well-behaved fault is one where a state transition does not occur at the instant the circuit output is being sampled. That is, the fault is either present or absent at that instant and the outputs reflect logically appropriate results. This removes the possibility of one or more of the output connections being in state-transition at the time of sampling. Multiple fault state transitions are permitted at all other times. The fault restrictions are extended somewhat for sequential circuits and will be discussed in the following Section.

The effects of fault intermittency during application of a testing schedule can best be described through a simple example of the faults in the full-adder circuit of Fig.3.4. Consider fault number #14. If, at the start of the schedule, #14 is present, test #1 (of value 1) produces an output of value 3 (see Fig.3.6) which indicates test #5 as the next test to be applied. If #14 continued to

locating solid faults the next step is to determine

be present the application of test #5 would result in an output of value 1, indicating a (correct) diagnosis of #-14 (fault #14). However, if, just before test #5 was applied, #14 went to the absent condition (the circuit working correctly) the circuit reacts to produce an output value 0. Such a result under normal circumstances would only be obtained by faults #4 or #13. The subsequent application of test #12 gives output results of value 2 or 3 for the absence and presence of #14, respectively. These results indicate (incorrectly) a diagnosis of #-4 or #-13. This example shows the possibility of an intermittent fault being incorrectly diagnosed. In fact this example can be extended to show that #14 can be diagnosed as fault #-1,#-3,#-4,#-12 and #-13 as well as #-14. Zero entries in the schedule table are not accessible to solid faults. However, under intermittent conditions such entries might be reached. Consider fault #22. Under intermittent conditions this fault could cause the schedule to reach test #15. If #22 is present when test #15 is applied, an output result of value 2 obtained which refers to a zero entry in the table. Similar diagnostic inconsistencies happen to the other circuit faults under intermittent conditions.

Diagnostic results of the type described above are meaningless unless a scheme for intepreting them is developed. The first step is to find the sets of faults which can, because of intermittency, reach each diagnostic or zero exit of the schedule. These sets of faults are named here as diagnostic-fault-profiles (d.f.p.). The

d.f.p.'s could be determined by an exhaustive process of simulating each fault for all possible intermittent occurrence patterns and recording the diagnostic results obtained. However, this is an inefficient and costly method because of the 2^n different, and possibly relevant, occurence patterns of each fault in a schedule of average test length n. A different approach has to be developed.

3.3.2 Efficient Calculation Of Diagnostic-Fault-Profiles

Under solid fault conditions efficient testing schedules are constructed by the repeated evaluation and selection of those circuit tests which partition sets of faults with maximum diagnostic resolution. The process is terminated when each fault or indistinguishable fault set is uniquely partitioned.

If the same test schedule is now considered in the intermittent fault situation, the partitions of such faults resulting from the application of the selected tests can be calculated. When all diagnostic exits have been reached the appropriate partitions will contain the d.f.p. fault sets, rather than the single faults obtained previously. This process can be performed by evaluating the partitioning with both the fault-present and fault-absent conditions. The d.f.p. calculation is then very efficient because it requires only a single pass through the schedule.

A suitable evaluation process is as follows:

(i) Define the circuit Fault-Matrix, the solid-fault testing schedule and the set of faults to be considered.

(ii) The current partition of faults is initially defined as being the complete set of circuit faults. The initial test number (TN) is defined as #1 (i.e. start at the beginning of the schedule).

(iii) Find the value of the input test of test number TN. Use the Fault-Matrix to find the circuit output response to input test TN for each of the faults defined in the current partition.

(iv) Create the new partitions of faults defined by the various output responses.

(v) The absent conditions of all the faults in the current partition produce the same output responses as the correctly working circuit. If an appropriate partition was created in (iv), add all these faults to that partition. If no appropriate partition was created in (iv), a new one is defined which contains all these faults.

(vi) If the output value corresponding to a new partition defines a diagnostic result, the list of faults in the partition is the d.f.p. of this diagnosis. If the output value corresponds to a zero entry, this is a new diagnostic exit with a d.f.p. defined by the faults in the new partition. All other new partitions are stored for future processing together with their associated next test number (which are determined from the schedule).

(vii) Select a new partition from store and call it the current partition. Set TN to the corresponding test number. Repeat from (iii) until all new partitions

have been processed and all d.f.p.'s calculated, then terminate.

A computer programmed version of the above process was written in FORTRAN IV and named FAULTS. The flow diagram is shown in Fig.3.7. The named arrays and variables have the same meanings and functions as those described for the SCHED program of section 3.2.3. FAULTS automatically generates new diagnostic exit numbers for the zero entries accessed by the intermittent faults, and the schedule is modified appropriately.

3.3.3 A Circuit Example

As an example of the d.f.p. calculation process consider the full-adder circuit of Fig.3.4. The Fault-Matrix, Fig.3.5(a), and the test schedule, Fig.3.5(b), are supplied as data for program FAULTS. The d.f.p.'s of the 27 diagnostic exits and 3 newly created exits (numbered #-28 and #-29 and #-30) are shown in Fig.3.8(a). The slightly modified schedule is shown in Fig.3.8(b). The program run-time on an ICL 1905 computer required to generate these results was 0.75 seconds. This is rather shorter than the program NSCHED because FAULTS uses the ready made schedule rather than having to generate it from first principles.

The d.f.p.'s of Fig.3.8(a) show that diagnostic results # -9,-10,-14,-18,-21,-22,-23 or -27, are only reached by faults of the same numbers. Such results obtained from applying the schedule to the circuit, would uniquely identify these faults. Closer examination of

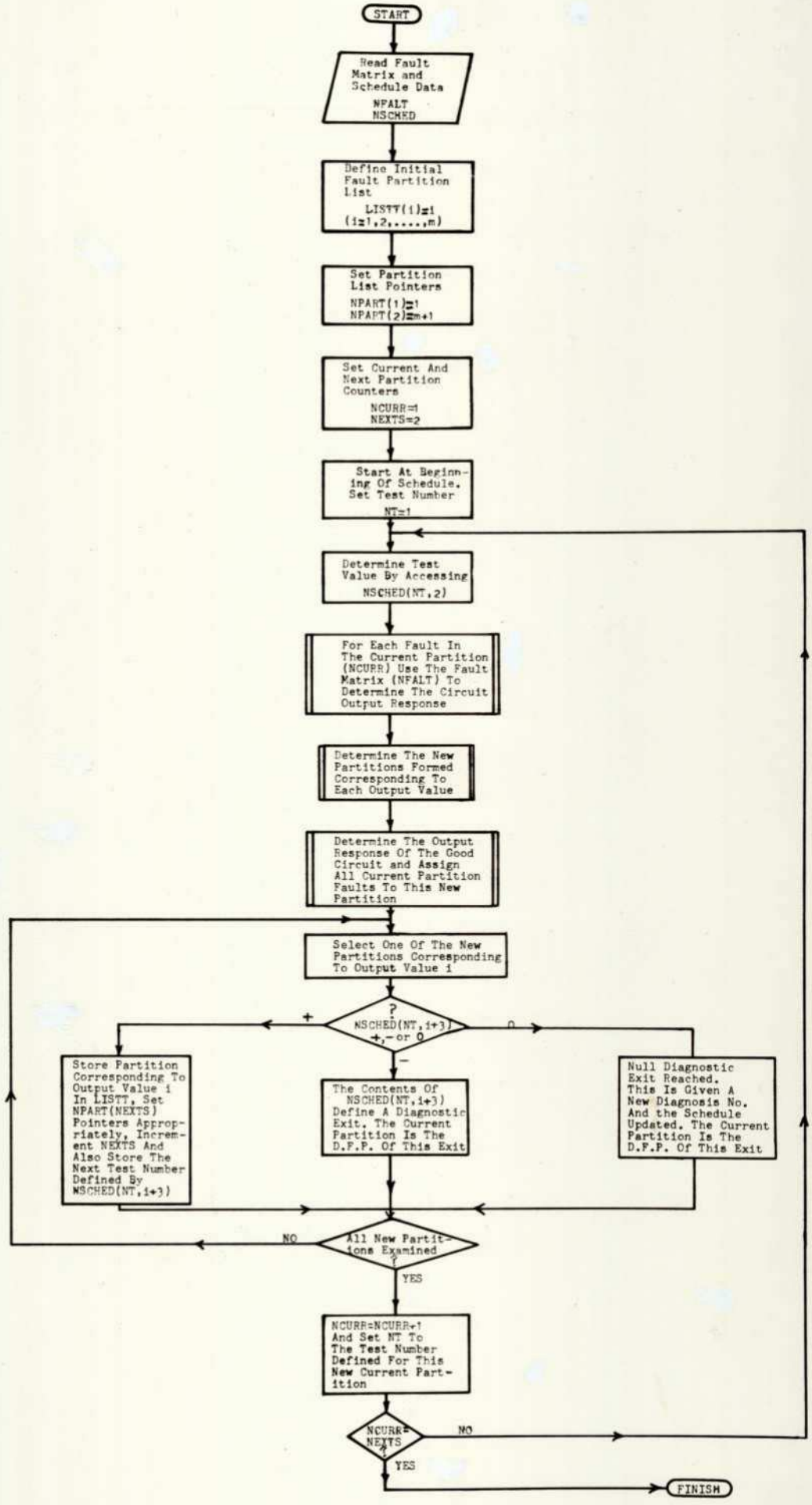


FIG. 3.7. FLOW DIAGRAM OF PROGRAM 'FAULTS'

Diagnostic Exit		Diagnostic Fault Profile
1	—	1,2,3, ... ,26,27
2	—	2,18
3	—	3,13,14
4	—	4,13,14
5	—	5,9,19,20,27
6	—	6,10,24
7	—	7,10,17,21,22
8	—	8,9,23,25
9	—	9
10	—	10
11	—	11,18
12	—	12,14
13	—	13,14
14	—	14
15	—	15,20,23
16	—	16,21,26
17	—	17,10,21
18	—	18
19	—	19,9,20,27
20	—	20,23
21	—	21
22	—	22
23	—	23
24	—	24
25	—	25,27
26	—	26,6,10,24
27	—	27
28	—	6,10,17,21,24
29	—	23,27
30	—	22,24

FIG. 3.8.(a). CALCULATED DIAGNOSTIC-FAULT-PROFILES

Test No.	Test Value	Output 0	1	2	3
1	1	2	3	4	5
2	2	6	0	7	0
3	2	0	-27	-25	0
4	6	8	9	10	11
5	0	12	-14	0	0
6	4	-21	0	13	0
7	3	0	-7	-22	0
8	3	-18	-11	0	0
9	4	-16	14	15	-12
10	2	16	0	-26	0
11	0	-15	0	17	0
12	2	0	0	-4	-13
13	5	0	-17	-10	0
14	3	0	-8	0	-9
15	3	-2	18	-30	19
16	3	0	-6	-24	0
17	2	0	-23	-20	0
18	2	-28	-29	-1	-3
19	0	-5	0	-19	0

FIG. 3.8.(b). MODIFIED CIRCUIT TEST SCHEDULE

the d.f.p. lists show that faults # 2,3,4,5,7,8,11,12,15 and 16 each appear in two d.f.p.'s only, that of the correctly working circuit (#-1) and in the lists associated with their own diagnostic results. Repeated applications of the testing schedule to the circuit in the (intermittent) presence of any one of these faults produces only the correct diagnosis or result #-1. These faults cannot be uniquely diagnosed because other faults can also produce the same results. For example, #18 can also be diagnosed as #-2 and #-1. However, repeated observation of the result #-2 and #-1 (and no others) would tend to indicate #2 as being the fault, since #18 would be expected to produce the result #-18 occasionally.

Fault #18 is diagnosed uniquely and therefore distinguishable from #2. However, #2 is not uniquely diagnosable and is indistinguishable from #18. This is an example of partial indistinguishability that is associated with the intermittency. This is caused by the circuit responses of those *relevant* tests chosen to diagnose fault #2 being the same as some of the *relevant* tests chosen for #18. Here, a *relevant* test is defined as one which, in the presence of the fault, stimulates a different output response to that of the fault-absent condition. Partial indistinguishability causes a loss in diagnostic resolution since these faults could have been uniquely resolved as solid faults.

The next stage in the process of diagnosing intermittent faults is to find a way of utilising the d.f.p.'s.

Consider fault #27. This is uniquely diagnosed by a result #-27. However, intermittent conditions may cause results #-25 and #-29 to be obtained when the schedule is applied to the circuit several times. The diagnostic process can be terminated since only fault #27 can reach both #-25 and #-29. Time is not wasted in waiting for result #-27 to appear. This example indicates a simple logical procedure of intepretation that allows a diagnostic result to be obtained, increases the speed of diagnosis, and which can be incorporated in automatic testing systems:

(i) Let REFINE contain the set of all possible fault numbers. Set counter N=0 and define a LIMIT value.

(ii) Apply test schedule to circuit and record a diagnostic result, #-x. Let N=N+1.

(iii) Intersect the set of faults in REFINE with the set of faults in d.f.p. -x. Store result in REFINE, i.e. (REFINE) = (REFINE)$\cap$(d.f.p.-x).

(iv) If REFINE now contains only one fault number it is the unique diagnosis. Otherwise, continue.

(v) If N > LIMIT the faults in REFINE are declared to be indistinguishable from each other. Otherwise, repeat from (ii).

It would also be possible to incorporate in the above process a check to determine if diagnosis #-x was consistently repeated. Repetition of #-x above a certain limit would indicate to a reasonable level of certainty that the correct diagnosis was actually #-x.

The above process of interpretation is used in

Section 5, where comparisons of test schedule efficiency are made under different intermittent fault conditions. It is not possible to quote 'average test lengths' as was done for the diagnosis of solid faults because the number of tests used is a function of the pattern of intermittent occurrence.

3.4 TEST SCHEDULE CONSTRUCTION FOR INTERMITTENT FAULT DIAGNOSIS

3.4.1 A Philosophy For Efficient Testing

The objectives of constructing a schedule for diagnosing intermittent faults are that:

(i) the position of a fault should be located with maximum diagnostic resolution.

(ii) the time required to perform the diagnosis is a minimum (i.e. a minimum number of tests).

The sequential type of testing schedule will be considered here. This is where the subsequent circuit input test depends on the output response to the present input stimulation. Generating sets of possible input tests has been considered previously in Section 3.2.1.

A simple and rather obvious philosophy for selecting tests to form efficient diagnostic testing schedules for intermittent faults is:

(i) Take maximum advantage of the time for which a fault is present.

(ii) At all stages of testing make maximum use of diagnostic information gained from previous stages.

These can be considered in more detail. Since an intermittent fault is present only for limited periods of time, any diagnostic process must be capable of utilising this time to best advantage. For this to be possible the schedule should minimise the delay between the fault appearance and the time when 'effective diagnosis' can proceed. The term 'effective diagnosis' refers to the creation of greater diagnostic resolution by using 'relevant' tests, rather than wasting valuable fault-present time by repeating irrelevant or redundant tests. Once effective diagnosis has started, to make the most of the fault-present time, maximum effort has to be made to ensure that the testing is efficient. When the fault goes absent the level of diagnostic resolution reached so far should not be wasted. When the fault re-appears the diagnostic process should be able to continue where it left off without any wastage of information.

A process for implementing the above philosophy can best be described using a simple example. Fig.3.9(a) is the initial partition of faults in a simple combinational circuit numbered #1,2,.....,9. #1 refers to the correctly working circuit. A test T_1 is selected on the basis that, of all possible tests, it maximises diagnostic resolution (based on the distinguishability criterion). T_1 produces the partitions (1,5,8,9), (2,3) and (4,6,7) corresponding to output results z_3,z_2 and z_1, respectively. This is shown in Fig.3.9(b). Under intermittent conditions the same test produces the partitioning (1,2,3,4,5,6,7,8,9), (2,3) and (4,6,7), Fig.3.9(c). Referring back to

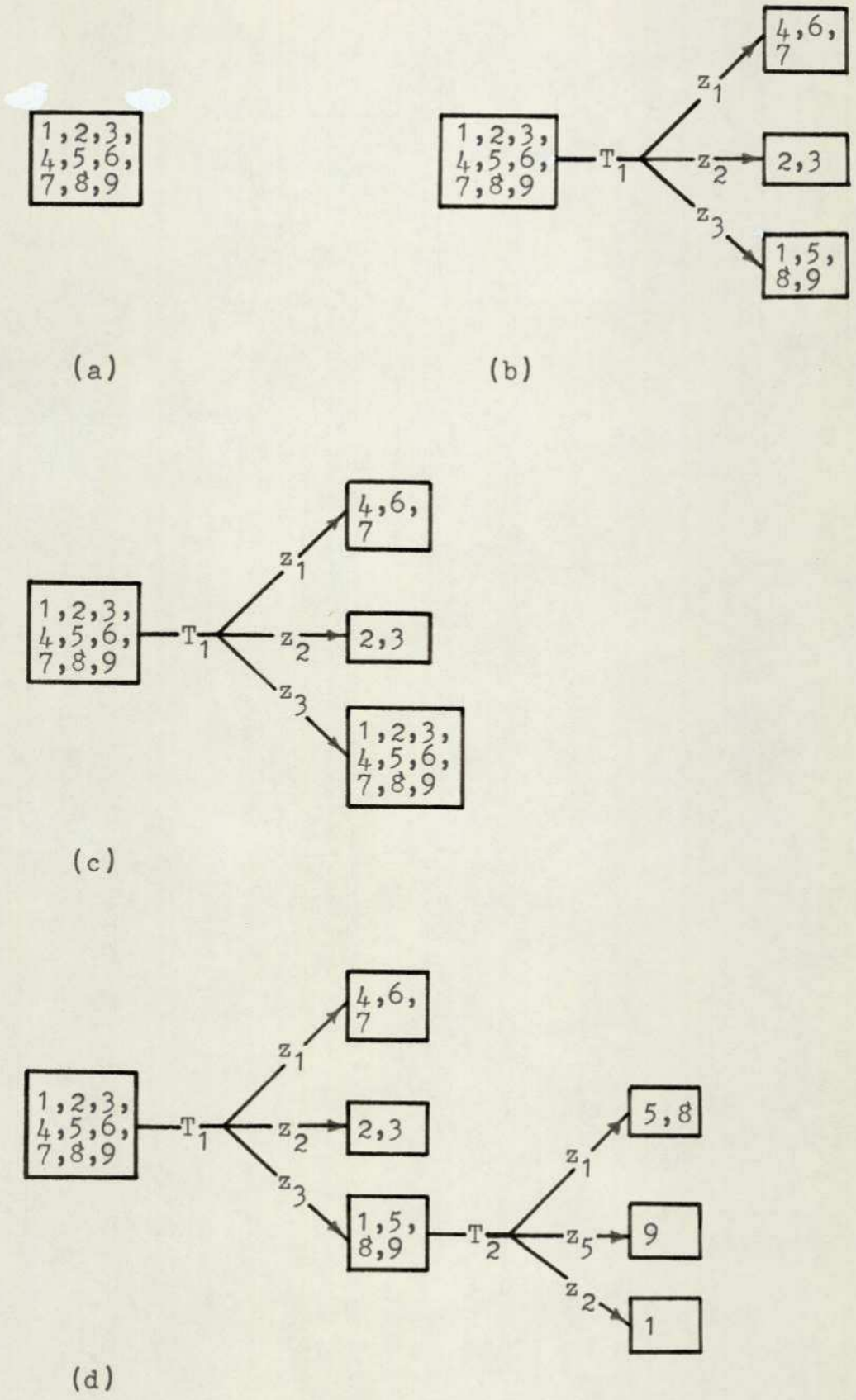
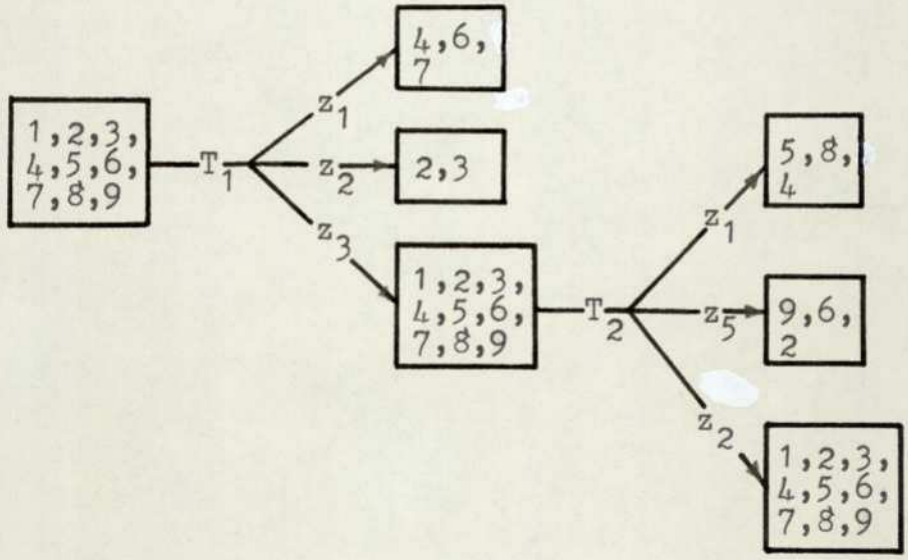
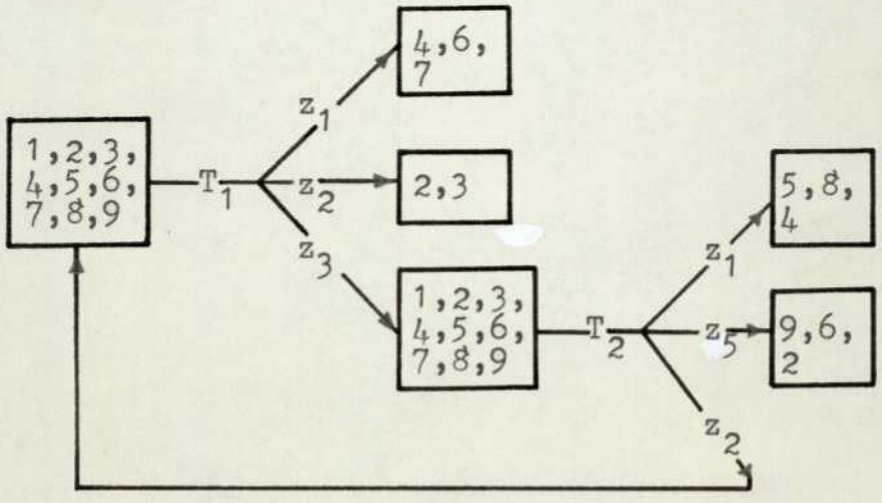


FIG. 3.9. EXAMPLES OF TEST PARTITIONING



(e)



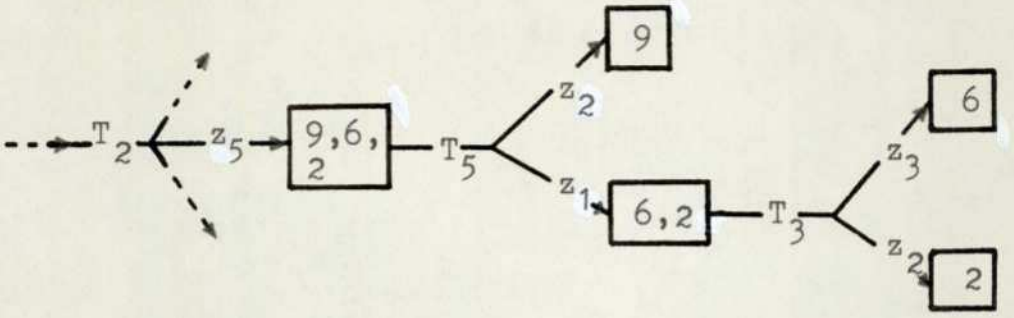
(f)

FIG. 3.9. (cont.

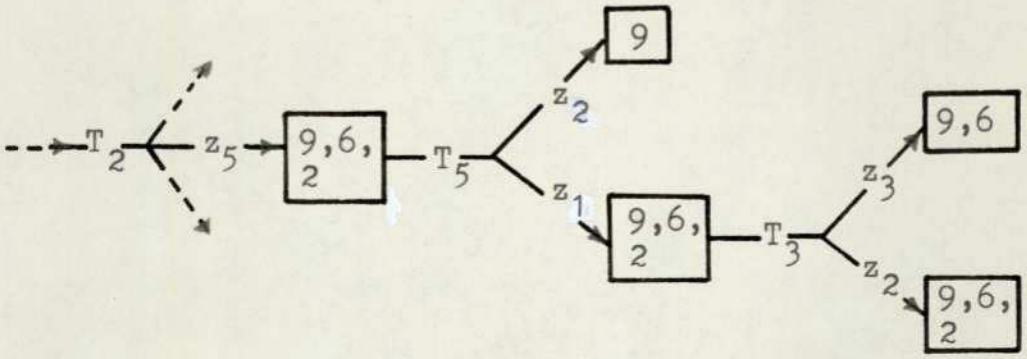
Fig.3.9(b), a new test T_2 is selected to maximise the partition containing the correctly working circuit. This is shown in Fig.3.9(d). The effect of intermittency is shown in Fig.3.9(e). T_2 partitions (1,2,3,4,5,6,7,8,9) into (5,8,4), (9,6,2) and (1,2,3,4,5,6,7,8,9). All faults have thus been distinguished at least once from the correctly working circuit by tests T_1 and T_2. If the distinguishability criterion is applied to find further tests for (1,2,3,4,5,6,7,8,9), the same tests, T_1 and T_2, will be selected. Since this partition is identical to the initial partition the schedule can effectively extended by looping from the z_2 output response of T_2, back to the initial partition, Fig.3.9(f).

The intermittent nature of the faults causes #2,4 and 6 to be detected by both T_1 and T_2 (Fig.3.9(e) and (f)). In the fault-absent conditions the schedule will loop around the tests $T_1, T_2, T_1, T_2, T_1, \ldots$ etc. During this process, if any of the faults appear, it is only a very short time before a relevant test is applied and effective diagnosis continued.

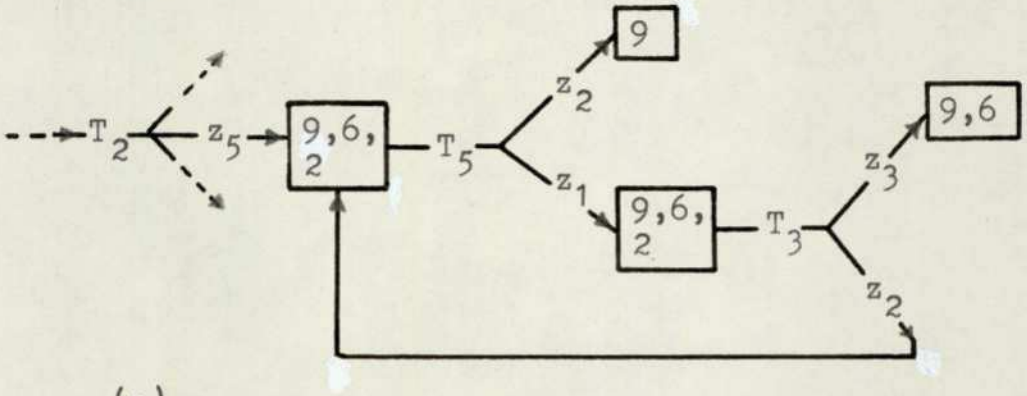
The other partitions produced by tests T_1 and T_2, (4,6,7), (2,3), (5,8,4) and (9,6,2), can be dealt with in a similar manner. Having determined that the fault is either #9,6 or 2 (for example) by obtaining the result z_5 from test T_2, other tests can be selected to increase this resolution further. Test T_5 would partition (9,6,2) into (9), (6,2) and T_3 would partition (6,2) into (6) and (2), Fig.3.10(a). Under intermittent conditions the partitioning



(a)



(b)



(c)

FIG. 3.10 FURTHER TEST PARTITIONING

is rather different, as shown in Fig.3.10(b). Since all faults have been distinguished at least once from the fault absent conditions, a loop can be formed. This is shown in Fig.3.10(c). If the fault goes absent the schedule loops around the tests as shown. Re-appearance of the fault allows effective diagnosis to start quickly and the previous diagnostic information, the knowledge that the fault is either #9,6 or 2, is not wasted. The criterion of test selection guarantees that maximum diagnostic use is made of the time the fault is present.

When all the remaining fault partitions are treated in a similar manner the schedule becomes a series of intersecting loops which terminate when a fault is uniquely identified. Transition from loop to loop can only occur when the fault is present.

The treatment of the simple example has illustrated how the stated testing philosophy could be implemented and the form a testing schedule could take.

3.4.2 Implementation of Schedule Formation Process

A formal process of test schedule construction, conforming with the stated philosophy of the previous sub-section (3.4.1), can be divided into several parts. The first is the selection of efficient tests from a set of possible tests. This uses the same selection criterion as the solid fault schedule construction process. The second part is the calculation of partitions created by the selected tests for the fault-absent and fault-present

conditions. The final part is the formation of the schedule loops which the testing sequence follows during the fault-absent conditions.

The overall process of schedule formation is somewhat similar to the solid fault schedule procedure discussed in Section 3.2.3, and is as follows:

(i) Form the Fault-Matrix, removing redundant tests and indistinguishable faults.

(ii) Define the current partition of faults as the complete set of circuit faults (or indistinguishable fault sets).

(iii) Make a copy of the current partition in COPY.

(iv) Use the Fault-Matrix data and the distinguishability criterion to select which of the tests (T_ℓ) produces the largest diagnostic resolution for the faults in COPY.

(v) Eliminate from COPY all faults which, in the presence of test T_ℓ, produce different output results to that generated by the fault-absent condition.

(vi) Use the ℓth row of the Fault-Matrix (corresponding to T_ℓ) to create the new partitions generated by applying the test T_ℓ to the circuit in the presence and in the absence of the faults of the current partition. These new partitions are treated as follows:

(a) If a partition contains only one member the fault is uniquely diagnosed. This diagnostic result is recorded.

(b) If a partition contains the same faults as the current partition, but is not the new partition associated with the non-faulty condition, then partial indistinguishability exists. The current partition is declared indistinguishable and the new current partition number is set equal to that from which this one was derived. Go to (viii).

(c) Other new partitions are stored for future processing and the schedule data recorded appropriately.

(vii) The new partition associated with the fault-absent conditions now becomes the current partition. Repeat from (iv) until COPY is empty (indicating that all the faults in the current partition have been distinguished at least once from the fault-absent condition).

(viii) The sequence of selected tests form a complete section of the testing schedule. This is turned into a closed loop by branching from the output value associated with the fault-absent condition generated by the last test, back to the first test in the sequence. The schedule data is recorded appropriately.

(ix) Select one of the stored new partitions and call it the current partition. Repeat from (iii) until all new partitions have been processed, then schedule construction is finished.

A computer programmed version of the above process was written in FORTRAN IV and named INTERMIT. The flow diagram is shown in Fig.3.11. The named arrays and

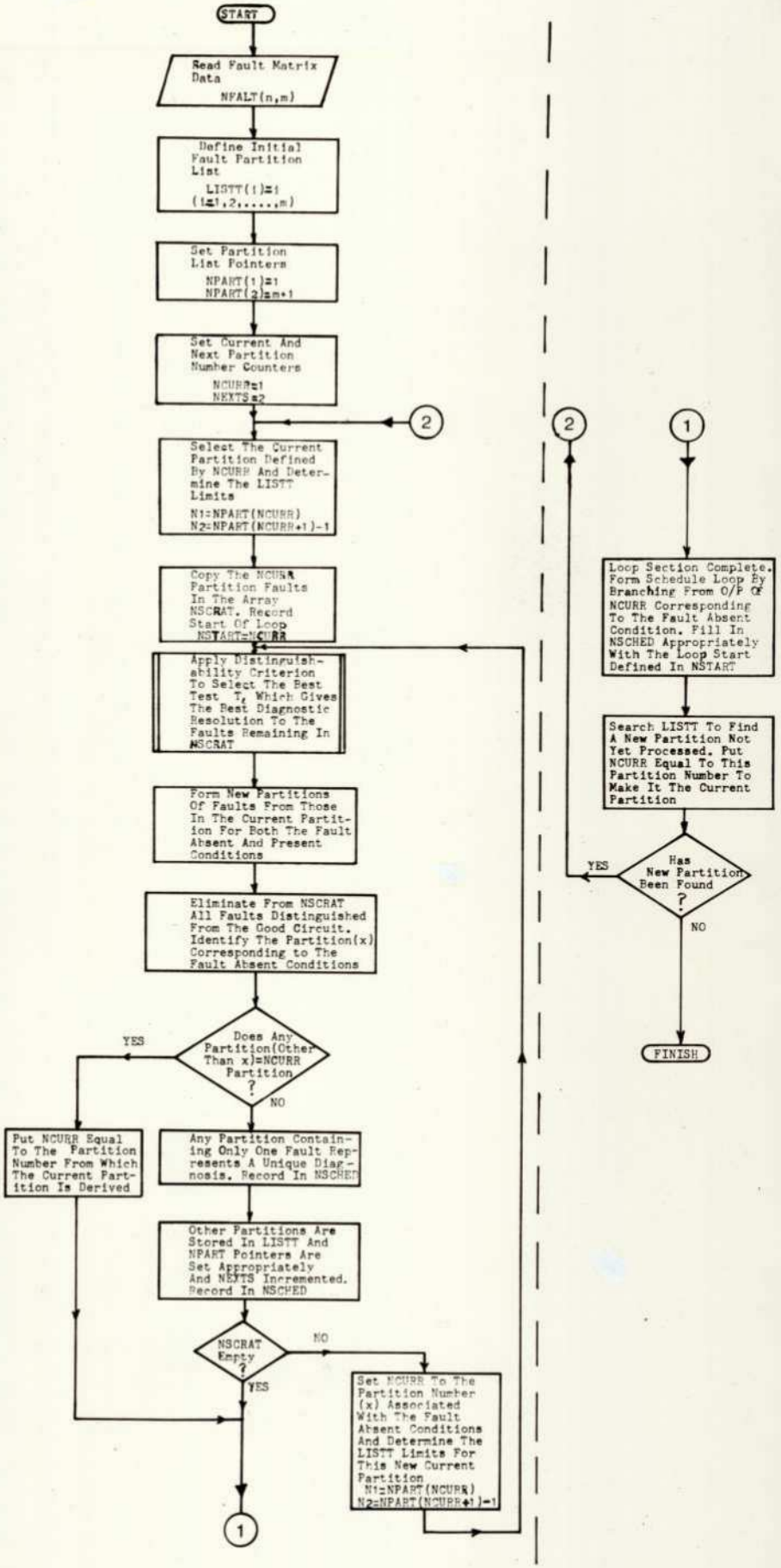


FIG. 3.11. FLOW DIAGRAM OF PROGRAM 'INTERMIT'

variables have the same meanings and functions as those described for the SCHED program of Section 3.2.3. The only additional names are:

NSCRAT - a one dimensional array for storing the current partition to permit deletions to be made when a current partition fault is distinguished from the fault-absent condition.

NSTART - is an integer variable for storing the number of the partition forming the start of a schedule loop.

The schedule produced by this program is in the same form as that generated by NSCHED. The schedule entries correspond to test numbers, test values, pointers to next test numbers (+ integers), and diagnostic results (- integers representing the number of the fault diagnosed). Zero entries represent invalid output responses not obtainable, even under intermittent fault conditions.

Partial indistinguishability, mentioned previously in 3.3.3, results in the inability of certain faults to be uniquely diagnosed. In this situation the testing schedule will loop continuously once it has reached the maximum level of resolution. Continuous looping in a particular section of the schedule is an indication of the presence of such a fault, not an absolute diagnosis. This indicator could be incorporated in the decision making process of an automatic tester to predict this is a diagnosis.

3.4.3 A Circuit Example

As an illustration of the intermittent fault diagnostic schedule construction process consider the full-adder circuit of Fig.3.4. The Fault-Matrix, Fig.3.5(a), is supplied as data to the program INTERMIT. The test schedule generated is shown in Fig.3.12. Eight seconds of program run-time were required using an ICL 1905 computer. The test (or partition) numbers at which loops feed back are shown ringed. The indistinguishable faults associated with some of these loop sections are shown in Fig.3.13. The testing diagram associated with this schedule is rather complicated and is not shown here.

In designing the schedule to deal with intermittent faults, the size is rather larger than the solid-fault equivalent (59 tests compared to 19). This schedule would have to be used in conjunction with an automatic testing system of suitable storage and decision making capabilities to be of any real use.

A solid fault can be considered a special case of an intermittent, where the fault-present condition lasts for a period greater than the time required to complete the diagnostic process. Therefore, this schedule can also be used to diagnose solid faults although there is a reduction in diagnostic resolution compared with an equivalent solid fault schedule.

Since the number of tests required to diagnose a particular intermittent fault is a function of the pattern

Test No.	Test Val	Output 0	1	2	3
1	1	2	3	5	5
2	2	26	0	27	0
3	2	0	-27	31	0
4	0	32	-14	0	0
5	6	6	9	7	8
(6)	3	-18	6	0	0
7	5	0	35	34	0
8	0	40	0	39	0
9	4	10	11	13	12
10	1	-21	0	42	0
11	1	0	-25	43	0
(12)	0	12	-14	0	0
13	3	14	17	15	16
(14)	5	-18	14	0	0
15	1	-22	0	45	0
16	7	0	46	0	47
(17)	2	18	19	1	20
18	1	21	0	22	0
19	1	0	-27	53	0
20	0	54	-14	0	0
21	4	-21	0	56	0
22	3	0	23	-24	0
(23)	6	0	18	24	0
24	1	-10	0	25	0
(25)	3	0	24	-24	0
26	4	-21	0	57	0
27	3	0	28	-22	0
(28)	5	0	2	29	0
29	2	-10	0	30	0
(30)	3	0	29	-22	0
(31)	4	0	-25	3	0
(32)	2	0	0	4	33
(33)	0	33	-14	0	0
34	1	-10	0	58	0
35	3	0	36	-24	0

(cont.

No.	Val	0	1	2	3
(36)	2	37	0	7	0
37	1	-10	0	38	0
(38)	3	0	37	-24	0
39	2	0	-23	59	0
(40)	5	0	8	0	41
(41)	0	41	0	-20	0
(42)	5	0	10	-26	0
43	2	0	-23	44	0
(44)	3	0	11	0	-9
(45)	2	-24	0	15	0
(46)	0	46	0	-9	0
47	1	0	-27	48	0
48	5	0	49	0	-20
(49)	0	16	0	50	0
50	1	0	-27	51	0
51	4	0	-9	52	0
(52)	5	0	50	0	-20
(53)	4	0	-23	19	0
(54)	1	0	0	20	55
(55)	0	55	-14	0	0
(56)	5	0	21	-10	0
(57)	5	0	26	-10	0
(58)	4	-26	0	34	0
(59)	3	0	39	0	-20

FIG. 3.12. INTERMITTENT FAULT TEST SCHEDULE EXAMPLE

Fault No.	Loop No.	Associated Indistinguishable Faults
2	14	18
3	54	13,14
4	32	13,14
5	46	9
6	25	10,24
6	38	10,24
7	30	10,22
8	44	9,23,25
11	6	18
12	12	14
13	33	14
13	55	14
15	41	20
16	42	21,26
17	56	10,21
17	57	10,21
19	52	9,20,27

FIG. 3.13. INDISTINGUISHABLE FAULT SETS

of occurrence, it is not possible to quote a figure for 'average test length'. Comparisons of operating efficiency of this type of schedule and the d.f.p. technique are made in Section 5.

3.5 DISCUSSION

Two different techniques have been developed for diagnosing intermittent faults. One makes use of the diagnostic data generated by a ordinary solid fault testing schedule. The other constructs schedules specifically for diagnosing intermittent faults.

The main drawback to the use of either type of schedule is the reduction in diagnostic resolution caused by the partial indistinguishability between some faults. However, this is related to the circuit and the set of possible input tests, and is not a function of the construction processes. This does not detract in any way from the usefulness of such schedules because most test and maintenance engineers would welcome assistance of any kind in locating this most troublesome class of faults.

A suitable definition for diagnostic resolution permitting partial indistinguishability to be taken into account can be 'the sum of the totals of faults distinguishable from each circuit fault'. This can be thought of as the number of *fault-singles*. Using this definition the distinguishability of the d.f.p. schedule for the full-adder circuit (Section 3.3.4) is 1938 fault-singles. A greater resolution, 1956 fault-singles, is obtained for the schedule constructed specially for diagnosing intermittent faults (Section 3.4.3).

The investment of time and effort in implementing and using a solid fault schedule, and designing the schedule construction process, is not wasted if the d.f.p. method of interpreting the diagnostic data is used. However, it will be shown later, in Section 5, that the diagnosis of intermittent faults using the specially designed schedule is performed with greater efficiency in terms of time. Such a schedule does require greater storage space than the solid fault equivalent but has other advantages as well. If the schedule is left running over a period of time, without any need of manual control or observation, it will gradually work closer and closer to full resolution of the fault condition (if it is possible). For example, a system with a troublesome intermittent fault could be tested overnight when the system is not normally in operation. This is a period of time which is perhaps long enough for the fault to have occurred often enough and for a sufficient total time to permit a useful degree of diagnostic resolution to be obtained.

Either schedule could be useful for detecting those faults sensitive to mechanical and thermal stresses. Observation of the testing process could perhaps allow determination of particular levels of the different kinds of stress stimuli that induce the faults to occur. Information such as this would be particularly useful from the component realiability aspects of system design and construction.

In complex systems, having the capabilities for time-sharing between normal operational functions and self-checking and diagnosis, an intermittent fault test schedule could prove most useful. Detection of a malfunction during normal operation, by

parity or special hardware circuit checks, could be used to switch to the self-checking mode, calling on the testing schedules to perform diagnosis. If looping occurs in the schedule, indicating that the fault has disappeared, normal operations could be re-started. A subsequent occurrence of the fault would permit the schedule to continue where it left off, without any loss of previously gained diagnostic information. When the required level of diagnostic resolution is reached, corrective action can be taken (i.e. mannual repair or reconfiguration).

Limitations in both of the schedule construction processes and in the d.f.p. evaluation lie with the implementation techniques rather than the underlying philosophies. One major limitation was in the use of the complete set of input tests rather than a selected subset. This is acceptable if only small sized circuits are to be used. For more general use one of the formal test evaluation techniques should be used, such as the d-Algorithm or Boolean Difference. The use of FORTRAN IV imposed certain limitations to calculation speed and storage availability. Such a high-level language neglects the useful speed and storage advantages that may be obtained by using the comprehensive bit manipulation and logical instructions of the basic machine language.

4. DIAGNOSIS OF INTERMITTENT FAULTS IN SEQUENTIAL LOGIC CIRCUITS

The diagnosis of faults in sequential logic circuits is a rather more complex task than for combinational circuits. This is a result of feedback connections and the existence of associated stored states. Fault intermittency only increases this complexity. To simplify the problem the concept of the 'well behaved' fault can be redefined and the class of sequential circuit limited.

Two aspects of test selection and schedule construction are investigated:

(i) Interpretation of solid-fault testing schedule reactions in the presence of intermittent faults to allow construction of diagnostic fault profiles (d.f.p.) (see Section 3).

(ii) Formation of schedules designed specifically for dealing with the diagnosis of intermittent faults.

This investigation involved the development of new schedule formation and diagnostic data interpretation techniques and the writing of several computer programs to implement them.

4.1 INTRODUCTION

The present output states of a sequential circuit are a function of the present input test and also previous input tests. This is a result of circuit feedback connections allowing stored state conditions to exist. Such a circuit can be represented by:

$$Z = A(X,S)$$

where Z is the present circuit output $(z_1,z_2,.....,z_m)$, X is the present input stimulation $(x_1,x_2,.....,x_n)$, S is the present set of p stored states $(s_1,s_2,.....,s_p)$, and A is the logical function performed by the n-input, m-output and p-feedback connection circuit.

Unless all feedback connections are also circuit outputs the present set of states is not known and the circuit reaction to a given input stimulation is (generally) unpredictable. However, this uncertainty can be removed if the circuit states are reset to known and permissible values. Subsequent input stimulations then produce predictable outputs reactions and stored state transitions. To reach a known set of states either special reset circuitry must be used, or a sequence of input tests generated which take the circuit from any of the unknown states to a known state. In some cases such homing sequences are lengthy and difficult to calculate, or even non-existent. Hennie (85) has investigated this problem.

It follows from the above discussion that the presence of faults in a sequential circuit will cause even greater problems. The reset circuitry or homing sequence must be able to cope with

all fault conditions and still be able to reset the circuit to known or predictable states. A required reset state may not be allowed by certain fault conditions, so the actual state reached after applying the reset must be predictable. Most research in the field of circuit testing and fault diagnosis assumes that some reliable mechanism exists to reset the circuits to known or predictable states prior to calculation of fault detection test sequences (38). The assumption is also used in this project.

The reset mechanism can be easily modelled in a logic simulation system. The circuit is first initialised to logically consistent states with either the fault absent or present, as required. The circuit reset states are then specified and forced on to the circuit (see Section 2), which is then re-initialised. Particular faults may not permit the required state to exist and cause a logical inconsistency in the circuit model. For example, if an OR-gate output is also a stored state and is required to be at a 1, but a s-a-0 fault exists at the same location, an inconsistency occurs. The removal of the forced states can be simulated and any inconsistency is removed by the circuit changing to some new but predictable values. Chang, et al, (28), call this operation 'momentarily resetting' the feedback lines. With the circuit in known or predictable states the selection and evaluation of suitable circuit input tests for detecting and locating faults can proceed.

In general, the limitations and assumptions used to evaluate tests and construct testing schedules for solid and intermittent faults in combinational circuits also apply here. The one major difference is in the definition of the 'well-behaved' intermittent

fault. Previously the fault was permitted to change state at any time and at any frequency as long as a transition did not occur at the instant the circuit output was being sampled. For sequential circuits this definition is restricted further to permit fault state transitions to occur only once before a test is applied and once before the resulting output value is sampled. The reason for this restriction is that a fault transition may result in a change in circuit state each time a transition occurs. Multiple transitions between the times of applying an input stimulation and sampling the output result could cause multiple circuit state changes, the final circuit state being predictable only if the number of fault state transitions is known (which is unlikely).

Sequential circuits are classed as either synchronous, asynchronous or a mixture of both. For synchronous circuits any next input is a valid possibility as a test. This means that if the circuit has n inputs the choice of next test is one of 2^n possibilities. For asynchronous circuits, successive input tests must not differ in more than one bit (86). This means that an n-input circuit only has n possible next tests to choose from. Obviously, selecting and evaluating tests for an asynchronous circuits is a rather easier job because of this restriction in numbers. This was the major reason why work in this project was limited to consideration of the asynchronous class of sequential circuits. The limitation is all the more valid when the difficulties of fault test generation are considered (see subsection 4.2.1).

As in Section 3, two methods of schedule construction for the diagnosis of intermittent faults are developed. One is based on

the interpretation of the diagnostic data produced by solid-fault schedules under intermittent fault conditions. The other method constructs schedules from first principles specifically for diagnosing intermittent faults. These are discussed in 4.3 and 4.4 respectively. Sub-section 4.2 outlines the various test selection and schedule fromation techniques used in solid fault diagnosis. Sub-section 4.5 ends this section of the thesis with a general discussion.

4.2 TEST GENERATION AND SCHEDULE CONSTRUCTION FOR THE DIAGNOSIS OF SOLID FAULTS IN SEQUENTIAL CIRCUIT

4.2.1 Test Generation

Techniques for generating sets of tests for detecting and locating faults in sequential circuits have been reviewed by several authors (28),(29),(54) and (55). For this reason they will be discussed only briefly here.

(a) Successor Tree

The problem of deriving test sequences for detecting faults in sequential circuits using an algorithmic approach was considered by Poage and McClusky (87). Their approach is best described by a simple illustrative example. Consider the tree structure of Fig.4.1. $Y_1, Y_2, \ldots\ldots, Y_n$ correspond to the possible circuit states. $Y_1{}^{o}Y_1{}^{i}$ are the initial (known) states of the correctly working circuit and the faulty circuit (fault #i). x_1 and x_2 are the possible next input tests. Each tree node shows the states of the two circuit conditions, and each branch represents

the application of some input test. In this simple example it is assumed there are only two possible input tests x_1 and x_2.

When the output of the faulty circuit is different from that of the correctly working circuit the fault is detected, indicated by F in the diagram. The ringed states are ones which are the same values as previous circuit states and which form branches that can give no further useful testing information. Fig.4.1 shows two possible testing sequences which would resolve fault i from the good circuit, $x_2x_2x_1x_2$ and $x_2x_2x_2x_2x_1$. The first sequence is the shortest and is of minimal length since all possible paths and tests have been evaluated.

A successor tree of the form Fig.4.1 can be constructed for each circuit fault. However, storage problems and excessive computation time make this a practical process only for small circuits having few inputs and states. The problem of locating faults is even greater since not only must fault i be distinguished from the correctly working circuit but it must also be distinguished from each other fault condition.

The general process is valid for either synchronous or asynchronous circuits. It does assume that, in the presence of any of the fault conditions, the circuit can be reset to known state values.

This work has been extended by Hennie (85) to derive sequences of tests for determining initial circuit states. Such distinguishing sequences can be applied at any time

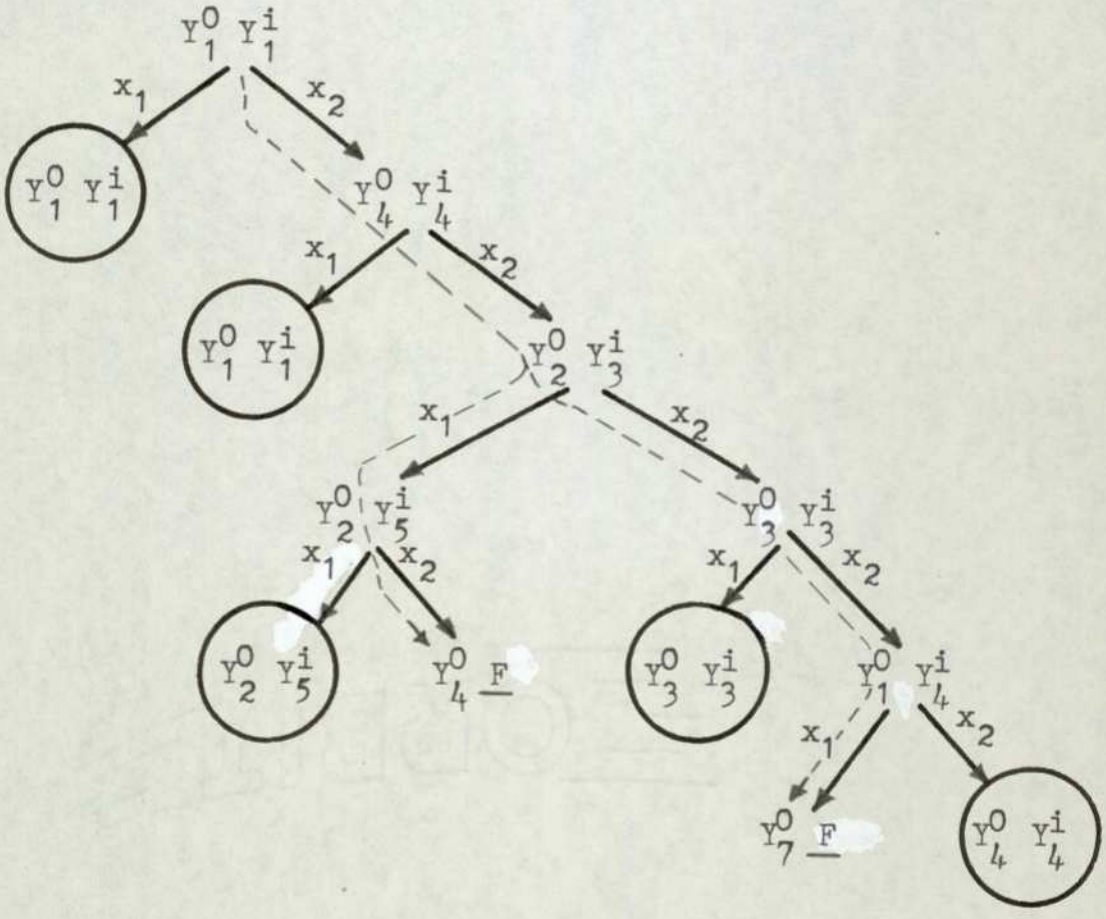


FIG. 4.1. SUCCESSOR TREE EXAMPLE

and, together with certain state transition sequences, permit fault conditions to be detected. Not all circuits have distinguishing sequences. Kohavi and Lavallee (88) have suggested a method for designing circuits such that distinguishing sequences are guaranteed.

(b) d-Algorithm

The techniques of the d-Algorithm, discussed previously in Section 3.2.1, have been extended for sequential circuits (89),(90),(91). The sequential circuit is thought of as a cascaded connection of identical combinational circuits each one representing the sequential circuit at a given instant in time. The problem then becomes one of multiple fault diagnosis since the fault in the sequential circuit is duplicated in each of the combinational circuits. Fig.4.2 illustrates the circuit cascade where x(i), y(i) and z(i) associated with the ith circuit represent inputs, states and outputs of the circuit at the ith instant in time.

The process of test generation starts by attempting to d-drive from the site of a fault in i = 1 to the primary outputs z(1). If this is successful then an input test for the fault can be determined by a backward-trace to the inputs x(1). If it was not successful a d-drive to the feedback connections, y(1), is made. Any d or $\bar{d}$ propagated to these connections form feedback inputs at the next time slot, i = 2. This is repeated for i = 2,3,.....,q until a d or $\bar{d}$ propagates to some output z(q). The sequence of q tests, $x_1 x_2 x_3 \ldots\ldots x_q$, generated by the backward-trace

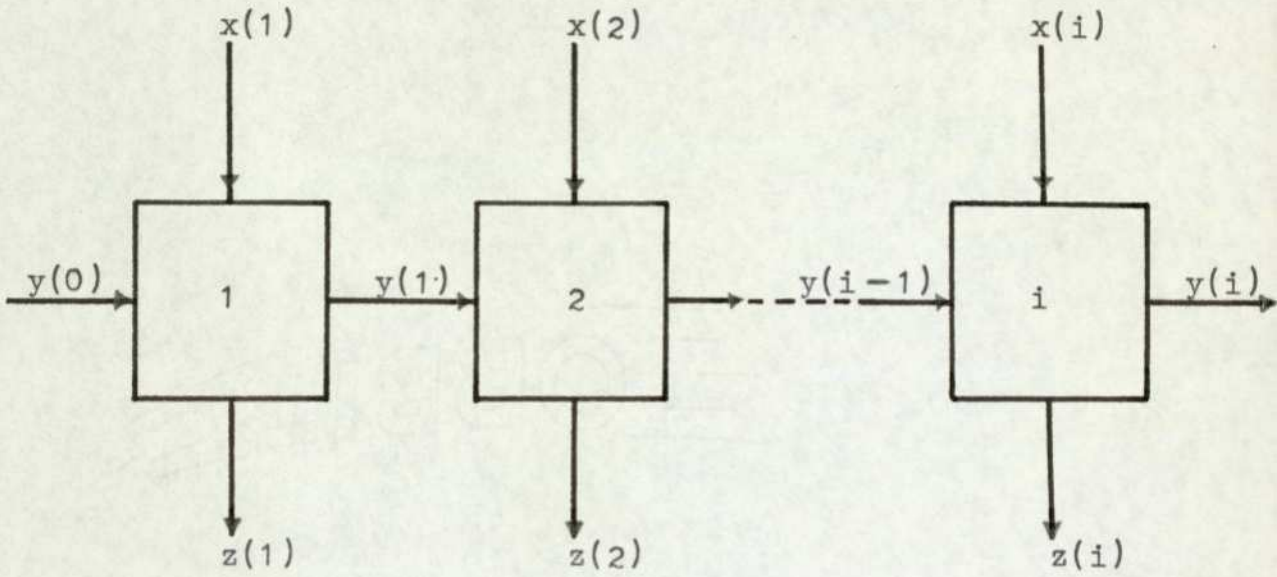


FIG. 4.2. CASCADE OF IDENTICAL COMBINATIONAL CIRCUITS

operation is then a test for the circuit fault. The sequences may not be of minimal length but if a fault is detectable this technique will find a detection sequence for it. Knowledge of the initial state of the circuit is required for test generation unless a homing sequence is defined. Calculation of such a homing sequence could be costly in terms of computer run-time for large circuits.

One way to simplify the process of test selection is to consider only a sub-set of the possible next input tests, chosen at random, and evaluate which of these give the best testing results. Breuer (91) has investigated such procedures using the d-Algorithm. The technique does not guarantee the detection of all fault conditions but does permit rapid construction of test sequences for a large percentage of circuit faults.

(c) Sequential Analyser

This technique is due to Seshu and Freeman (38). It uses simulation experiments on a model of the circuit to generate tests and testing sequences for fault detection and location. It is not an analytical method as the two previously described techniques are. Essentially, the sequential analyser consists of:

(i) a simulation model of the circuit under consideration.

(ii) a description of the various fault conditions.

(iii) a series of test generation heuristics.

It works by choosing series of input tests which can distinguish each fault from all others. The information

gain or distinguishability criteria are used as the basis for selecting the best test from a set of possible tests.

The initial partition of faults contains the complete circuit fault set. The circuit model is reset to known or predictable states. Each fault condition is injected into the circuit model in turn and simulated for each of the next possible input tests. One of these tests is selected as giving the best diagnostic resolution. The new partitions created by applying this test are determined. Each of these is then treated in turn, as above, to find the next best test. This continues until all faults are distinguished from each other or a required level of resolution is reached. When the process fails to find a suitable test, other heuristic strategies are tried. If all these fail the partition of faults under consideration is declared indistinguishable. In selecting the tests as described above a diagnostic testing schedule of the sequential type has been generated. The simulation process is more complicated than dealing with combinational circuits because at each stage in the process the circuit states associated with the faults in the partitions have to be recorded. These states also have to be correctly regenerated in the simulation model each time a particular fault is processed.

The Analyser can also be used in a slightly different mode of operation. The test sequences can be supplied by the design engineers or generated in whole or part by one of the previously described techniques and evaluated by

applying them to the simulated circuit. The resulting fault partitions can also be calculated.

In the automatic test schedule generation mode of operation, synchronous circuits would have too many next-test possibilities to be dealt with economically. Asynchronous circuits, with n inputs, have only n next-test possibilities. The Sequential Analyser is therefore only of real use with asynchronous or small synchronous circuits.

The heuristics which are generally used with the analyser were proposed by Seshu (34). They are as follows:

(i) NEXT-BEST-TEST - To generate the j^{th} input test, first compute all tests differing in one bit (only) from the $(j - 1)^{st}$ input test. Apply some selection criterion to calculate a figure of merit for each test. Discard those tests whose figure of merit falls below some defined threshold. Of the remining select the one with the largest merit value and call it the next (j^{th}) test. The remaining tests are placed in a pushdown list of good tests. If no j^{th} test can be found select from the good list a previously calculated test which may produce further distinguish-ability. Simply stated, this heuristic does not permit the wastage of potentially useful tests. It is a way of bypassing the limitation that the locally optimised method of choosing tests does not always give best results in a global sense.

(ii) WANDER-- If none of the next possible tests differing from the (j - 1)th test by one bit produce any further distinguishability one of them is selected at random to become the jth test. This is repeated for the (j + 1)st, (j + 2)nd, etc., tests until either a good test is found or the number of wandering steps is up to some defined limit.

(iii) RESET - This heuristic assumes that there are a number of permissible reset state (p.r.s.) that it is possible to momentarily force the circuit in to. When no j^{th} test can be found to give further distinguishability, each permissible reset is applied to the circuit in turn. The one giving the highest figure of merit becomes the j^{th} test.

(iv) COMBINATIONAL - This can be a useful heuristic for circuits which have a low ratio of feedback loops to numbers of gates. Essentially, the feedback loops in the circuit model are opened and tests are generated for each fault by one of the methods of section 3.2.1, such as the d-Algorithm. When a good test is found it is applied to the circuit, this time with the feedback lines connected. Useful partitioning may then occur. The heuristic is based on the assumption (and hope) that the feedback lines do not compromise the effectiveness of the selected good tests.

Heuristic (i) has proved the most effective in practice (28). When one heuristic fails the others are

tried in turn until a useful result is obtained. Data on the effectiveness of each is recorded to allow the Sequential Analyser control routine to select the best order of application, through a learning process.

Seshu and Freeman (38) implemented the Sequential Analyser using a compiled-code simulator and a special circuit race analysis routine. This special routine was named the Huffman analysis program and was used to determine whether a race condition was critical or non-critical when two or more feedback lines changed state simultaneously. If a particular test input causes a critical race, the final stable state being one of several possibilities, it is discarded.

The Sequential Analyser can deal with small pure synchronous circuits. No race analysis is required since all feedback lines are caused (purposely) to change state simultaneously by the circuit clock pulse. If a circuit is a mixture of synchronous and asynchronous structures, as many modern logic systems are, the Analyser cannot be used in this form.

Test selection techniques (a) and (b), described previously, will guarantee a sequence of tests for detecting a fault if such a sequence exists. This is not so with the Sequential Analyser and is a major drawback to this technique. The use of local optimisation in the test generation heuristics means that the test sequences will not be of minimal length.

Other techniques for generating tests for sequential

circuits have also been developed. Carter, et al, (92) have described a method for the IBM 360/50 computer system which uses special hardware to reset all (or some) of the processor to known states by a SCAN-IN procedure. The states of the processor, either immediately or after the partial execution of an instruction, can be recorded by a SCAN-OUT operation. Interpretation of the resulting data can lead to fault diagnosis.

Hsiao and Chia (93) have extended the Boolean Difference technique to deal with asynchronous sequential circuits.

One other approach is to physically open all feedback lines during testing to turn sequential into combinational circuits (28). In such circuits, faults in the feedback control logic must not affect the switching to open-circuit or it may still be sequential.

With the availability of the circuit simulator (see Section 2) the Sequential Analyser seemed to be the best technique to utilise in this project for generating circuit tests and forming testing schedules. The Successor Tree method would not be capable of effectively dealing with the size of circuit under consideration here, even though this is quite small. The d-Algorithm, although analytically more efficient and powerful than the Analyser, would require much more programming time and effort to implement.

4.2.2 Sequential Analyser Procedure For Solid Faults

The basic Sequential Analyser has been outlined in the previous sub-section, 4.2.1(c), and is reviewed in detail by Chang, et al, (28). The original implementation by Seshu and Freeman (38) used a complex race analysis routine and a comprehensive heuristic selection procedure based on analysis of previous performance capabilities. In other words the program learned, from past experience, to select those heuristics which tended to give best results. The heuristics themselves in the Seshu and Freeman implementation were also quite complex.

It was decided to simplify the implementation in this project by removing any race analysis and reducing the complexity of the heuristics. The reasons for this are:

(i) This reduces the time and effort (which are limited) required to perform the implementation.

(ii) Race analysis is only essential if the testing sequences of real circuits are being considered. In such cases simultaneous transition of two or more feedback loops could result in one of several possible stable states being reached depending on the normal statistical variations in propagation delays which occur in the real circuit components. This is known as a critical race. In this project only ideal circuits and components are being considered and calculations of circuit states by the simulation routine allow reproducable results to be obtained. In other words critical races do not exist in the

test by 1 bit only.

circuit simulation model and only need be considered if schedules to be used in real circuits are being constructed.

Only three heuristics are useful in this project, NEXT BEST TEST, WANDER and RESET (Section 4.2.1(c)). COMBINATIONAL requires some other method of test generation and is too complex to implement with the time and effort limitations of the project. It was therefore not included. The other three were modified to ease the problems of implementation:

(i) NEXT BEST TEST - To generate the j^{th} test, first compute the results of all tests differing from the (j - 1)st test by only one bit. Apply some selection criterion to calculate figures of merit for each. Discard those tests whose figure of merit falls below some defined threshold. Of the remaining select the one with the largest merit value and call it the j^{th} test. The remaining tests and their associated partitions are all stored for later analysis. If no j^{th} test can be found, switch to heuristic (ii).

(ii) WANDER - This is the same as it was originally defined in 4.2.1(c). When a good test is found switch back to (i). If the WANDER limit is exceeded switch to (iii).

(iii) RESET - When no j^{th} test can be found by one of the other heuristics, choose one of the permissible circuit reset states at random and apply it momentarily to the circuit. If this is a good test it becomes

the j^{th} test and return to (i). Otherwise, return to (ii) for another try at WANDER, or, if this has been tried more than a limited number of times, terminate the search and call this partition an indistinguishable fault set. Control can be switched back to (i) if other partitions are still to be processed.

In these modified heuristics all the NEXT BEST TESTS are evaluated. This allows a rather more global selection of tests than in the previous form. The final selection of tests for the schedule is then based on finding that mixture of locally optimised tests and NEXT BEST TESTS which give the best overall result.

The distinguishability criterion can be used for calculating the figures of merit for the various possible input tests (see Section 3.2.2). The data supplied to the procedure consists of the circuit description; the set of possible circuit faults; the set of permissible circuit reset states; and the threshold level and step number limits for controlling the switching between the various heuristics. The data produced by the procedure consists of a testing schedule of the same form described for combinational circuits (Section 3.2.3). To describe to the automatic tester that a particular test input is also a permissible reset state (p.r.s.), it is listed in the schedule at a negative decimal integer number of the form $-(T + 1)$. For an n-input and q-feedback connection circuit, T represents in binary the $(n + q)$ levels that are applied

momentarily to the appropriate circuit positions for the reset. The +1 permits a p.r.s. of all 0's to be described.

4.2.3 Sequential Analyser Implementation

A procedure for test selection and schedule construction designed for implementing the Sequential Analyser technique in this project is as follows:

(i) Define:

(a) the circuit to be tested.

(b) the set of circuit faults. These initially form the current partition of faults.

(c) the set of permissible reset states (p.r.s.).

(d) the threshold level and step limits for controlling heuristic switching.

(ii) Choose at random one p.r.s. Extract from this the input test value T_i and the reset states R_i. Using the circuit simulator apply T_i to the circuit for one of the current partition faults. Momentarily apply reset states R_i and record the output result and the associated circuit states. Repeat for each of the other faults. Form new partitions due to test T_i.

(iii) Select one of the new partitions and call it the current partition. If no new partition exists go to (xi).

(iv) Simulate each fault of the current partition by:

(a) Initialising the circuit to the appropriate fault and state conditions.

(b) Selecting test which differs from the previous test by 1 bit only.

(c) Performing the simulation process.

The circuit output results and the associated states are recorded in current storage.

(v) Repeat (iv) for all other possible tests which differ from the previous test by one bit. Eliminate any tests which cause circuit oscillation.

(vi) Apply distinguishability criterion to select which of the possible tests gives best diagnostic resolution. (Note: the current lists generated in (iv) and (v) are equivalent to the fault-matrix used for combinational circuits, but are only valid for these sets of states and inputs.)

(vii) If no T_i gives further distinguishability of the current partition faults, go to (viii). Otherwise, form the new partitions generated by the best test T_s, including the associated circuit states, and store for future processing. Other tests which produce figures of merit above some threshold level are stored in NEXT BEST TEST store, together with associated circuit states, to be analysed later. Repeat from (iii).

(viii) Apply WANDER heuristic. Select at random one of the tests from the possible set and define the fault partition and circuit state sets generated by it as the current partition, repeat from iv. If WANDER does not find a good test within a limited number of trials, go to (ix).

(ix) Apply RESET heuristic by repeating from (ii) using other p.r.s. values. If WANDER and RESET together

cannot find a good test within a limited number of reset trials, go to (x).

(x) The current set of faults is declared to be indistinguishable. This is recorded. Go to (iii).

(xi) Select one of the fault partitions, and the associated circuit state set, from the NEXT BEST TEST store and call it the current partition. Repeat from (iv). If all NEXT BEST TEST partitions have been analysed fully the schedule can be constructed by selecting those tests which give the best diagnostic resolution.

The above procedure was implemented in FORTRAN IV and named SEQ ANALY. The SIMULATION routine was used as a subroutine for this program. A flow diagram illustrating the program is shown in Fig.4.3. Many of the array and variable names have the same functions as in the program SCHED described in Section 3.2.3. Some new ones have the following names and uses:

LISTS - a one-dimensional array directly equivalent to LISTT for storing the circuit states associated with the faults stored in LISTT.

NSTORA - a two-dimensional array, the (i,j)th element of which is used for storing the circuit output response to test T_i in the presence of fault #j.

NSTORB - a two-dimensional array directly equivalent to NSTORB used for storing the associated circuit state.

MISTT - a one-dimensional array for storing the NEXT BEST TEST fault partition.

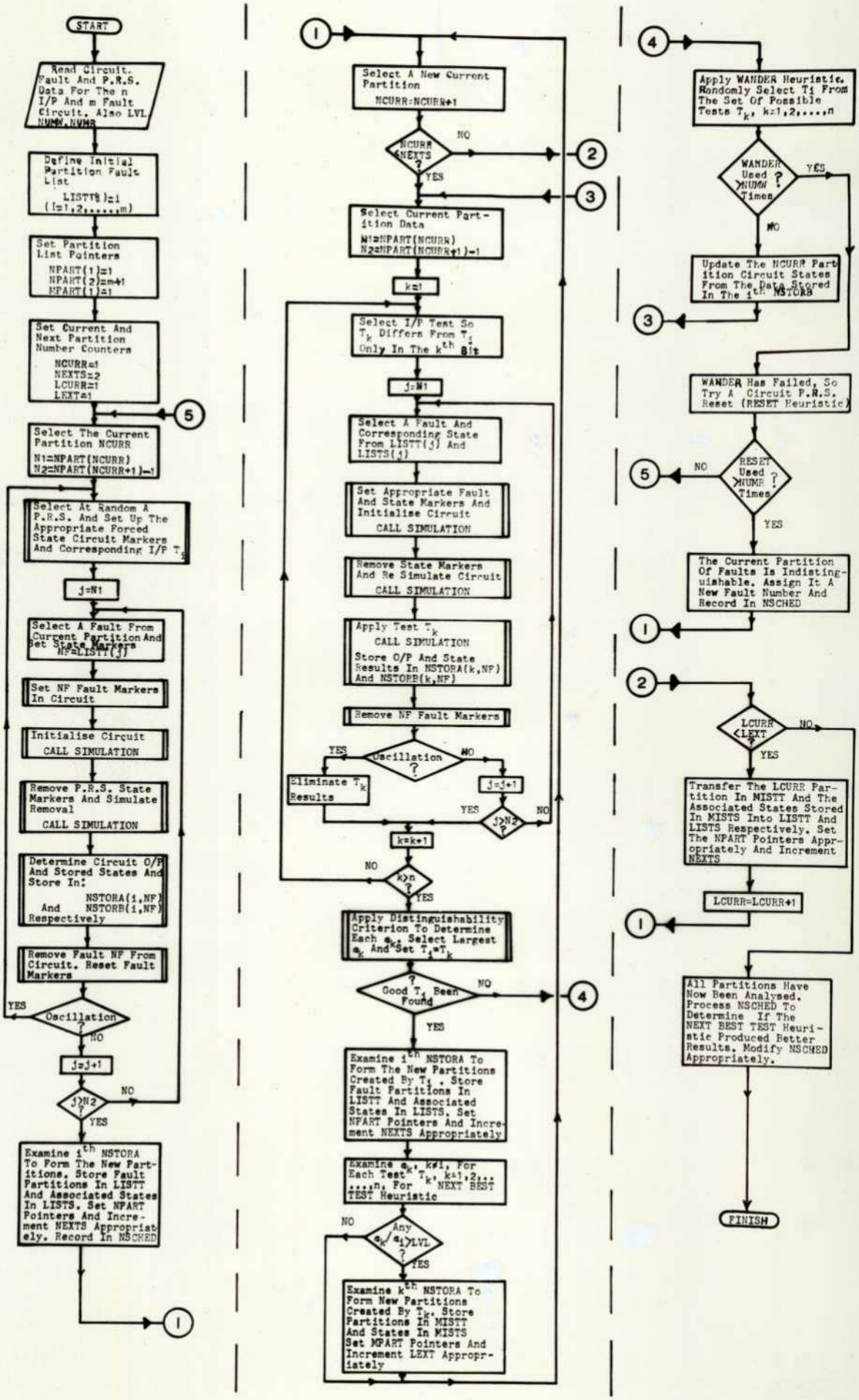


FIG. 4.3. FLOW DIAGRAM OF PROGRAM 'SEQ ANALY'

MISTS - an array directly equivalent to MISTT, used for storing associated circuit states.

MPART - a one-dimensional array to hold pointers for the addresses in MISTT and MISTS where the NEXT BEST TEST partitions are listed, i.e. MPART(x) stores the address in MISTT and MISTS where data on the xth partition is stored.

LCURR - a counter associated with NPART to store the number of the current NEXT BEST TEST partition under analysis.

LEXT - a counter associated with MPART to store the number to be given to the next newly generated NEXT BEST TEST.

NUMW - WANDER steps limit.

NUMR - RESET steps limit.

LVL - threshold level for selecting NEXT BEST TEST

The threshold value LVL is used in the program by calculating the greatest figure of merit test, T_i, and comparing the ratio α_k/α_i against the defined value of LVL for each of the n tests, $k = 1,2,.....,n$, but $k \neq i$. A test k is then selected for inclusion in NEXT BEST TEST only if $\alpha_k/\alpha_i > LVL$.

4.2.4 A Circuit Example

The circuit used by Seshu and Freeman (38) in their original description of the Sequential Analyser is used as an example of the program functioning, Fig.4.4. The limited fault set is the same as used by those authors, Fig.4.5.

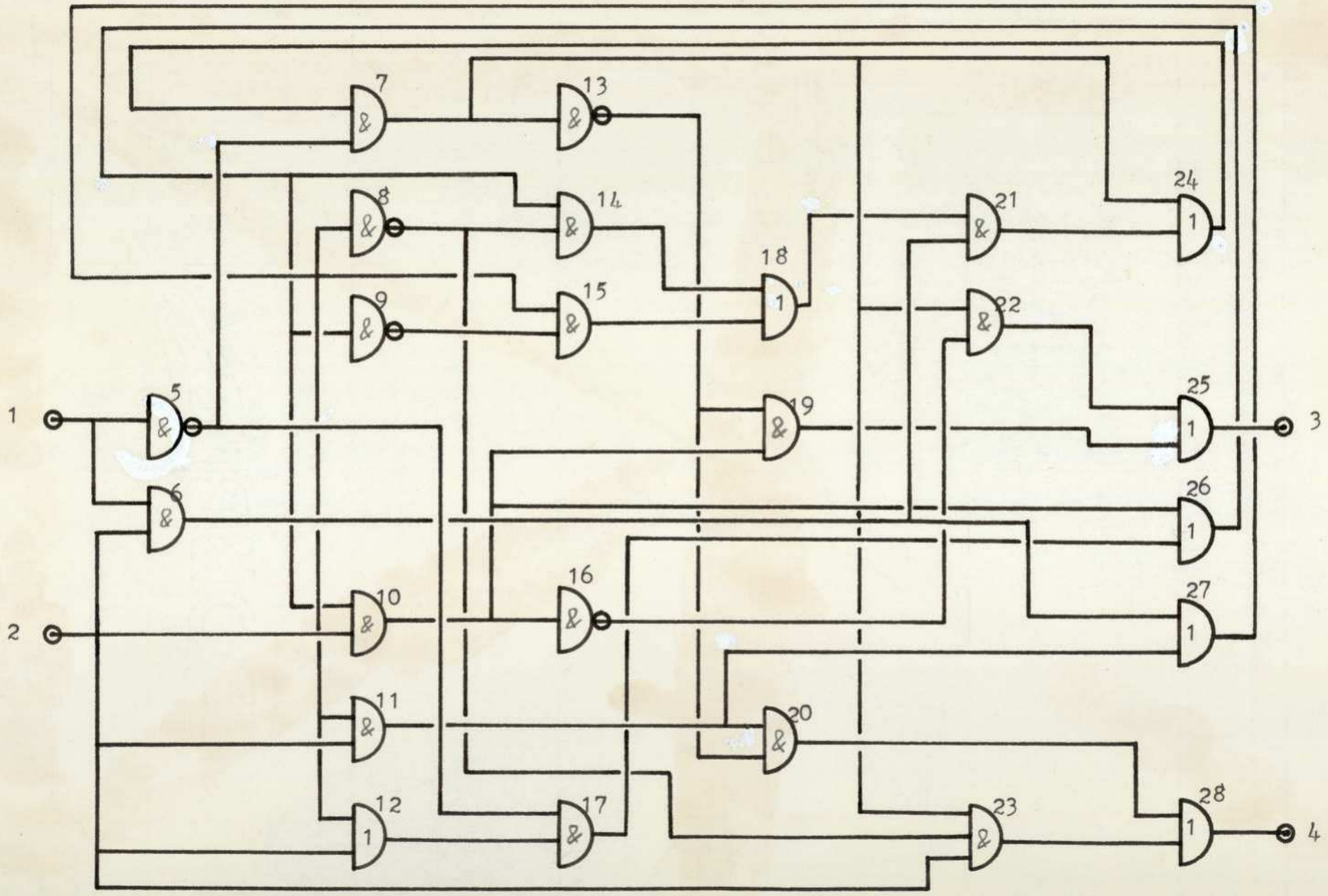


FIG. 4.4. A CIRCUIT EXAMPLE

FAULT DESCRIPTION

0	0	0	1→(absent)
6	0	0	2
6	1	1	3
6	2	1	4
7	0	0	5
7	24	1	6
7	5	1	7
10	0	0	8
10	2	1	9
10	26	1	10
11	0	0	11
11	2	1	12
11	27	1	13
12	0	1	14
12	2	0	15
12	27	0	16
14	0	0	17
14	26	1	18
14	8	1	19
15	0	0	20
15	27	1	21
15	9	1	22
17	0	0	23
17	5	1	24
17	12	1	25
23	0	0	26
23	2	1	27
23	8	1	28
23	7	1	29
27	0	1	30
27	6	0	31
27	11	0	32
19	0	0	33
19	13	1	34
19	10	1	35
20	0	0	36
20	13	1	37
20	11	1	38
22	0	0	39
22	7	1	40
22	16	1	41
18	0	1	42
18	14	0	43
18	15	0	44

(cont.

26	0	1	45
26	10	0	46
26	17	0	47
21	0	0	48
21	6	1	49
21	18	1	50
25	0	1	51
25	19	0	52
25	22	0	53
28	0	1	54
28	20	0	55
28	23	0	56
24	0	1	57
24	7	0	58
24	21	0	59

Fault No.

Fault Value

Connection From

Gate No.

FIG. 4.5. CIRCUIT EXAMPLE FAULT SET

The feedback connections of the circuit derive from elements #24,26 and 27. These, together with the circuit inputs, contain the full circuit state data required for the simulation and analysis. The three p.r.s. states used by Seshu and Freeman are the logic levels (11000), (01000) and (10001) applied to the outputs of element numbers (1,2,24,26,27). These are input to the program, using a format described in sub-section 4.2.2, as -4,-3 and -18 respectively

The effectiveness of the various heuristics can be evaluated by computing the results generated by SEQ ANALY for various settings of NUMW, NUMR and LVL. The important factors from which effectiveness may be gauged are:

(i) Program run-time.

(ii) Diagnostic resolution of the schedule.

(iii) Average test length of the schedule.

(iv) Total number of tests in the schedule.

The weight of importance associated with these factors for different users would vary. For example, keeping the total number of tests in the schedule down to a certain level but at the same time maximising diagnostic resolution could be the set of requirements necessary for production line testing using a low storage capability tester. Here, the program run-time expenses might have to be quite large to meet such specifications (if at all possible to satisfy).

For this circuit example NLV was initially set at 0.5 and NUMW varied from 0 to 5 (in integer steps) for settings of NUMR between 0 and 3. The value of 3 is the upper limit

possible in this example since there are only three p.r.s. The program run-times (including program compilation) are shown in Fig.4.6(a). The diagnostic resolution, measured in terms of number of fault-pairs resolved by the generated schedule, is shown in Fig.4.6(b). The average test length, calculated on the assumption of equal probability of fault occurrence, is shown in Fig.4.6(c). The total number of tests in the generated schedules is shown in Fig.4.6(d).

Examination of these plots show that the computation cost increases approximately linearly with increase in NUMW and NUMR. However, at the point NUMW = 2 and NUMR = 0, the diagnostic resolution has already reached a nearly maximum plateau. Although the average test length and total number of tests have risen above the NUMW = 0 and NUMR = 0 values, they are not excessive at this point.

A user interested in maximising diagnostic resolution (1647 fault-pairs) would have to accept a computation time of 168 seconds, an average test length of 5.78 and a total schedule length of 32 tests. This is found by setting NUMW = 5 and NUMR = 1. Near maximum resolution (1645 fault-pairs) is found using NUMW = 2, NUMR = 0, with a computation time of only 83 seconds. NUMW and NUMR were fixed at these values and LVL was varied between 0 and 1.0. The result is shown in Fig.4.6(e). In fact the use of NEXT BEST TEST failed to increase the diagnostic resolution at all. The plot shows the change of computation time, which is equivalent to computation cost, with LVL. There is a sharp increase in program run-time at a point just

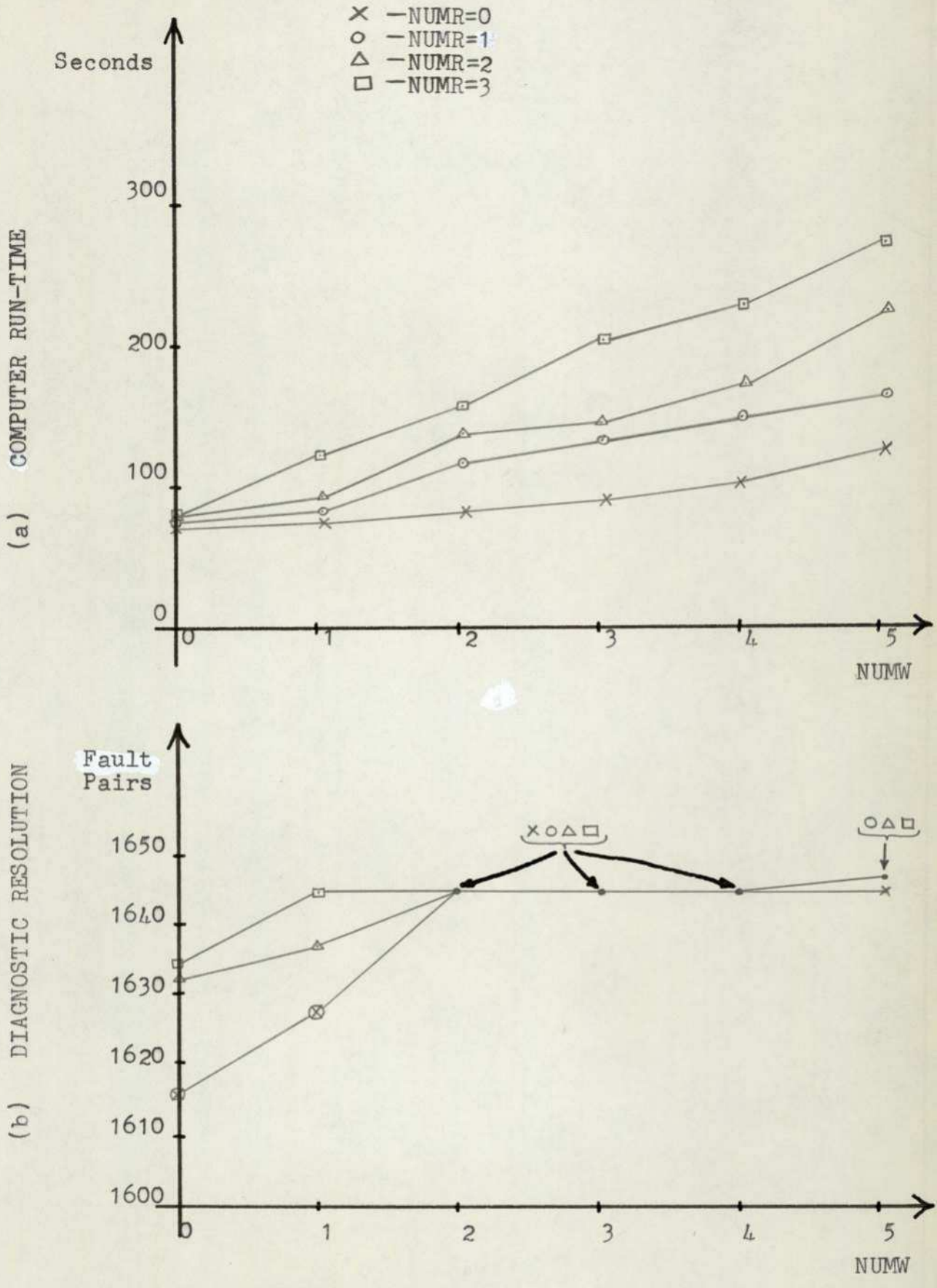


FIG. 4.6. EVALUATION OF HEURISTIC EFFECTIVENESS

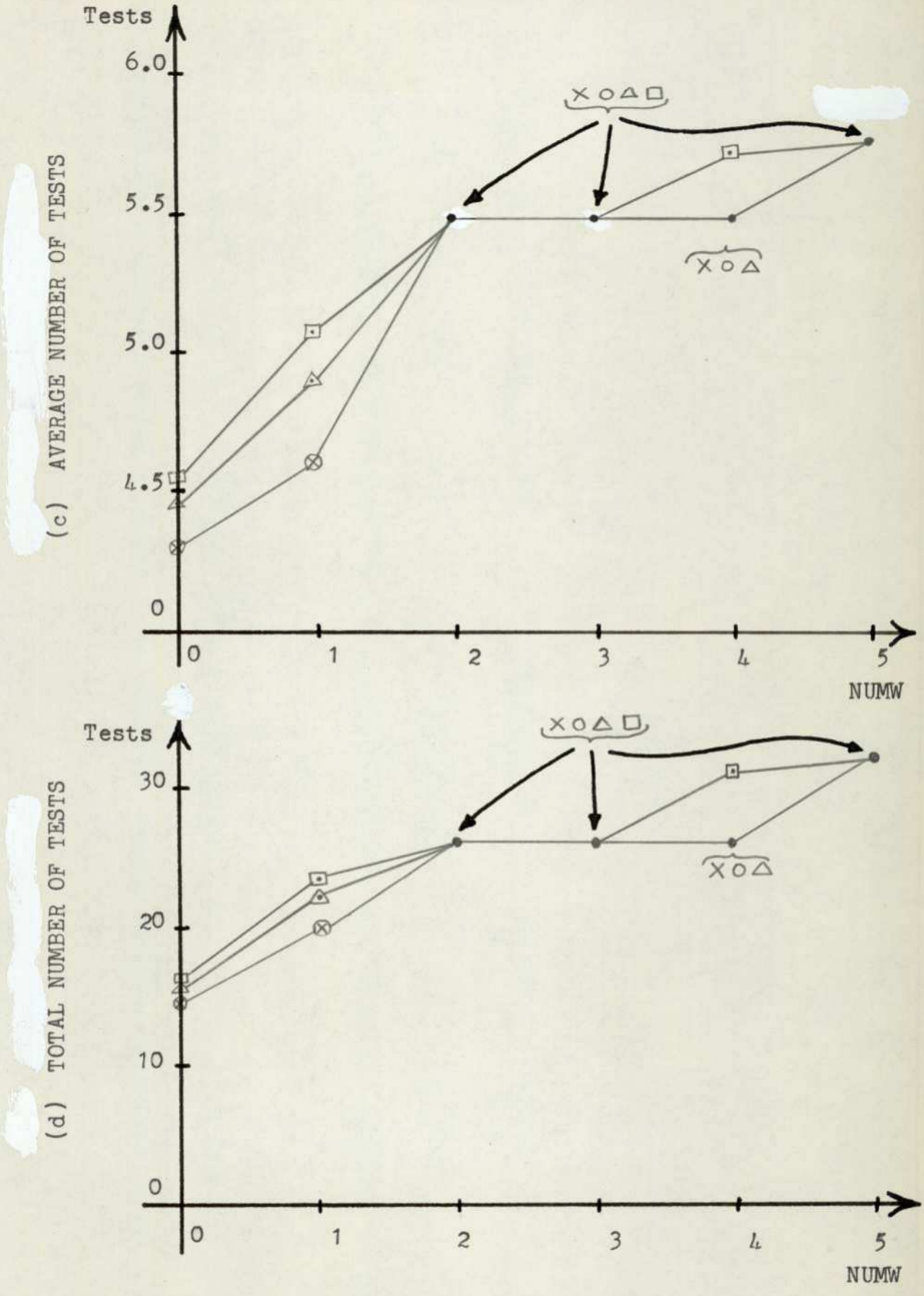

Tests
6.0
5.5
5.0
4.5
0
(c) AVERAGE NUMBER OF TESTS
X O △ □
X O △
0
1
2
3
4
5
NUMW
Tests
30
20
10
0
(d) TOTAL NUMBER OF TESTS
X O △ □
X O △
0
1
2
3
4
5
NUMW

FIG. 4.6. (cont.

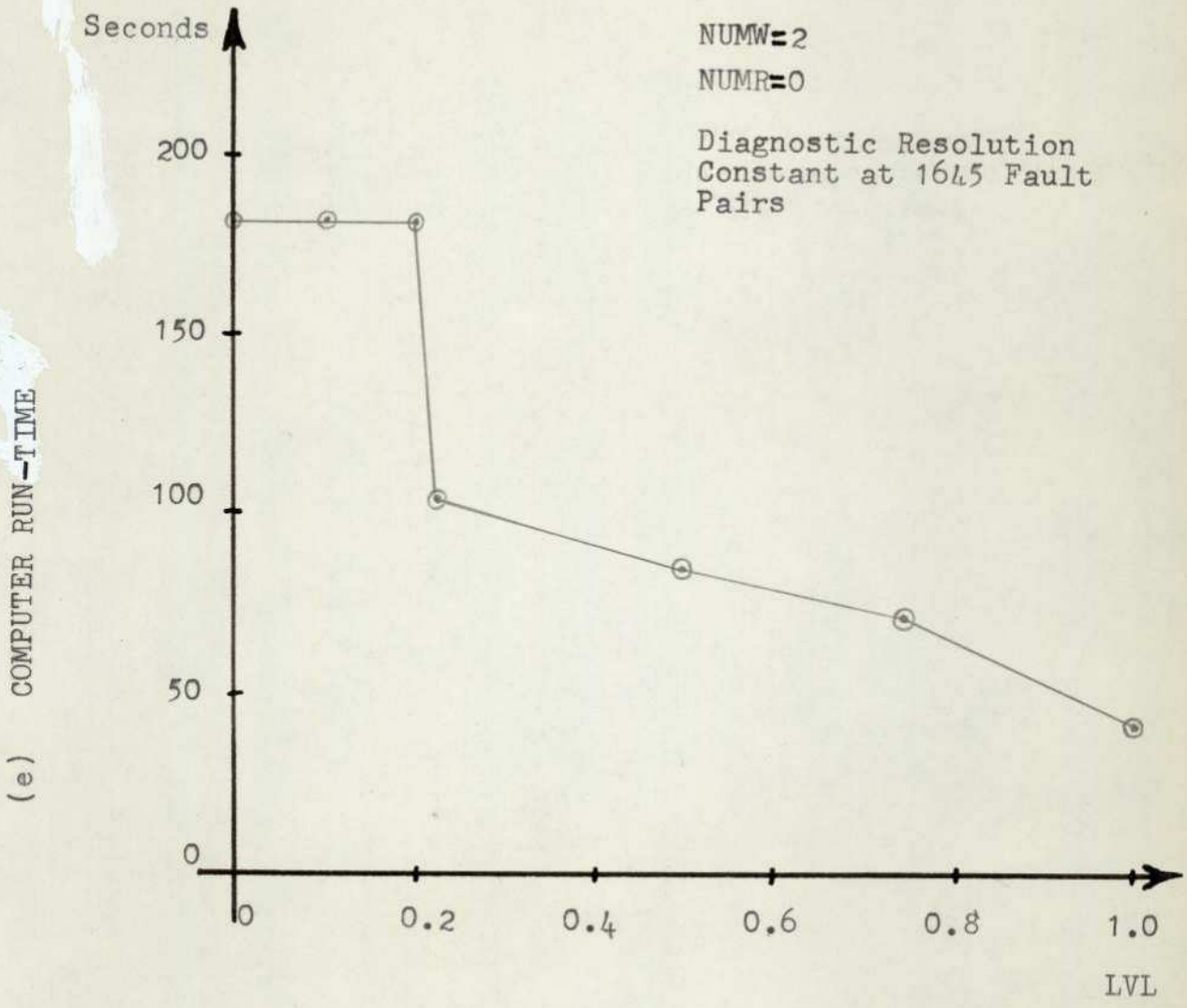

(e) COMPUTER RUN-TIME
Seconds
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1.0
LVL
NUMW=2
NUMR=0
Diagnostic Resolution Constant at 1645 Fault Pairs

FIG. 4.6. (cont.

below LVL = 0.25. This is caused by an increase in the number of tests which qualify for inclusion in NEXT BEST TEST. The minimum computation time for constructing this near maximum resolution diagnostic schedule is 43 seconds. The schedule is shown in Fig.4.7(a). The sets of indistinguishable faults, denoted by diagnosis numbers #60 to #73, are shown in Fig.4.7(b).

The schedule is somewhat longer than Seshu and Freeman's, 26 tests compared with 15. The average test length is also longer, 5.49 compared with 4.42. However, the diagnostic resolution is greater, 1645 fault-pairs to 1622. The differences are due to two major factors:

(i) The heuristic selection is rather different and not as efficient.

(ii) The lack of race analysis permits the use of certain tests that are not allowed in Seshu and Freeman's implementation. This gives greater diagnostic resolution.

So far only solid faults have been considered. The next stage is to find the properties of a schedule, such as the one described above, when used in the presence of intermittent faults.

4.3 INTERPRETATION OF SOLID FAULT DIAGNOSTIC TEST SCHEDULE DATA FOR DIAGNOSING INTERMITTENT FAULTS

4.3.1 The Effects Of Intermittent Faults On Schedule Data

During application of a testing schedule to a sequential circuit, intermittency of the fault state can

Test No.	Test Value	Output 0	1	2	3
1	-4	12	-7	2	15
2	2	3	16	17	-64
3	0	18	4	0	-27
4	2	19	0	5	-15
5	3	0	0	23	6
6	2	24	0	0	7
7	0	8	-49	-38	0
8	2	0	9	0	-29
9	3	0	0	0	10
10	1	11	0	0	0
11	0	-60	-4	0	0
12	2	-61	13	-11	0
13	3	0	14	0	0
14	2	0	-2	-31	0
15	2	-62	-51	0	-63
16	3	0	0	-65	-66
17	0	0	-67	0	-54
18	1	-68	-9	0	0
19	3	0	0	0	20
20	1	21	0	-12	0
21	0	22	0	0	0
22	2	-3	-70	0	-69
23	2	-46	0	-71	0
24	3	0	0	0	25
25	1	26	0	0	0
26	0	-73	-72	0	0

FIG. 4.7.(a). CIRCUIT EXAMPLE TEST SCHEDULE

New Diagnosis No.	Associated Indistinguishable Fault Sets
60	1,14,16,17,18,21,25,43
61	36,55
62	35,40
63	10,24,45
64	5,20,44,48,58,59
65	8,23,47
66	34,41
67	28,32,37
68	39,53
69	13,30
70	26,56
71	33,52
72	6,57
73	19,22,42,50

FIG. 4.7.(b). INDISTINGUISHABLE FAULT SETS

result in an incorrect diagnosis being reached. As with combinational circuits (Section 3.3.1) different patterns of intermittency can produce different diagnostic results. Each diagnostic exit of the schedule is therefore accessable to subsets of the complete fault set under such conditions. These subsets are the diagnostic-fault-profiles (d.f.p.). Such d.f.p.'s can be used to aid in the diagnosis of intermittent faults using the procedure discussed in Section 3.3.3.

The formation of d.f.p. in sequential circuits occur in a rather different way to those in combinational circuits. Here, the occurrence of a change in fault state can result in a transition of circuit state. This change might not be reflected at the circuit output until several test cycles later.

Consider a partition of p faults, numbered $(N_1, N_2, \ldots, N_p)$, and the associated circuit states, $(S_1, S_2, \ldots, S_p)$, existing in the presence of T_i. If, just before the next test T_{i+1} is applied to the circuit, fault N_k, $k = 1, \ldots, p$, can change state, the associated circuit states could also change, $S_k \rightarrow S_k^1$. This means that the actual fault partition is $(N_1, N_2, \ldots, N_p)$ with states $(S_1, S_2, \ldots, S_p)$ and $(N_1, N_2, \ldots, N_p)$ with $(S_1^1, S_2^1, \ldots, S_p)$. Application of test T_{i+1} to the circuit results in new state transitions $S_k \rightarrow S_k^2$ and $S_k^1 \rightarrow S_k^3$. If, just before the circuit output result is sampled, fault N_k changes state again, new circuit states could result, $S_k^2 \rightarrow S_k^4$ and $S_k^3 \rightarrow S_k^5$. The final partition of faults is $(N_1, N_2, \ldots$

$\ldots,N_p$) repeated four times over and the associated sets of circuit states ($S_1^2,S_2^2,\ldots\ldots,S_p^2$), ($S_1^3,S_2^3,\ldots\ldots,S_p^3$), ($S_1^4,S_2^4,\ldots\ldots,S_p^4$) and ($S_1^5,S_2^5,\ldots\ldots,S_p^5$).

Although the final partition is now rather more complex than at the time just prior to application of T_{i+1}, it can be simplified to some extent. Any fault N_k that has multiple entries having the same state value (i.e. $S_k^x = S_k^y$, where x = 2,3,4,5 and y = 2,3,4,5 but $x \neq y$), and which are in the same fault state, can be eliminated from the schedule since it is just duplicated data. Some of the resulting new partitions will still have multiple fault entires which must all be processed normally when subsequent testing and partitioning is performed. If the fault transition had been permitted to occur any number of times throughout the Test/Sample cycle, the number of states to be considered could be enormous. For this reason it was artificially restricted to only occur twice at most. This forms the justification of the well behaved fault restriction mentioned in Section 4.1.

If the partitioning described above is performed using a solid fault diagnostic schedule under the intermittent fault conditions, the final partitions contain the faults which make up the d.f.p.'s. Consider the final partition associated with diagnosis k. If this contained the fault set ($N_k,N_k,N_x,N_x,N_y,N_z,N_z,N_z$), with the associated states ($S_k^3,S_k^5,S_x^2,S_x^4,S_y^2,S_z^2,S_z^4,S_z^3$), the d.f.p. is ($N_k,N_x,N_y,N_z$) since these are the only faults capable of reaching diagnosis k under intermittent conditions.

The next step is to find an efficient method of calculating the d.f.p. sets for the solid fault schedules associated with sequential circuits.

4.3.2 Calculation Of Diagnostic-Fault-Profiles

The sequence of fault transitions possible during one Test/Sample cycle, within the limitations of the well-behaved-fault definitions, are:

0 - T - 0 - S

0 - T - 1 - S (T = TEST)

1 - T - 0 - S (S = SAMPLE)

and 1 - T - 1 - S

where 0 and 1 correspond to 'no-fault-transition' and 'fault-transition', respectively. For each fault these could result in four different circuit states. Each condition has to be evaluated to guarantee complete coverage in the d.f.p. calculation. Because of the possible multitude of circuit state conditions and the complexity of calculation it was decided to deal with each fault separately, find the diagnostic exits reached by each one and then combine the results after all faults have been evaluated. This means that p passes are required through the schedule rather than the single pass used for combinational circuits (Section 3.3.2).

A process for determining the d.f.p.'s is as follows:

(1) Define:

(a) the circuit to be tested.

(b) the set of circuit faults.

(c) the solid fault test schedule.

(ii) Initialise the simulation model of the circuit in the presence of T_1, the first schedule test, for the fault-absent (or fault-present) condition of fault N_k. Since the first test is also a p.r.s., variations in fault state prior to its application are irrelevant.

(iii) After the p.r.s. has been removed and the circuit settled, record the output result, the circuit states and the fault states in current lists. These results correspond to the 00 sequence fault transitions. Now simulate a fault transition and similarly record the output, circuit state and fault state in a current list. This is the 01 fault transition sequence.

(iv) Repeat from (ii) for the opposite initial fault state, so producing the sequences 00 and 01 for the other initial state.

(v) Examine the current storage lists to determine the partitioning. Each partition is examined to eliminate any duplicated fault that has:

(a) the same circuit state, and

(b) the same fault state.

If a new partition is one corresponding to a diagnostic exit, the fault N_k is recorded as being a member of that d.f.p. list. If this is a null exit (designated 0 in the schedule) the fault N_k is recorded as being a member of the d.f.p. list of this new exit. All other new partitions are stored for future processing.

(vi) Select a new partition as the current partition. If none is found go to (x).

(vii) Initialise the circuit to the circuit state and fault state defined by one of the current partition entries. Select the appropriate next test, T_i, from the schedule. Simulate the application of T_i to the circuit. Store the output, circuit state and fault state in the current lists. Change the fault state and simulate the transition. Store the circuit results as before. This corresponds to 0-0 and 1-1 fault transition sequences.

(viii) Re-initialise circuit to the same circuit and fault states as at the beginning of (vii). Now simulate a fault transition. Simulate the application of test T_i and record the appropriate results. Simulate a fault transition and record the appropriate results. This corresponds to the fault transition sequences 1-0 and 1-1.

(ix) Repeat from (vii) for each other circuit state and fault state defined by the current partition. Then go to (v).

(x) Repeat from (ii) for each other circuit fault N_k, $k = 1,2,\ldots\ldots, m$.

(xi) The set of faults reaching each diagnostic exit of the schedule is the d.f.p. for that exit.

A computer programmed version of the above process was written in FORTRAN IV and name DFP SEQ. The flow diagram is show in Fig.4.8. The named arrays and variables have the same meanings as those used in SEQ ANALY (Section 4.2.3). Other arrays and variables have the following

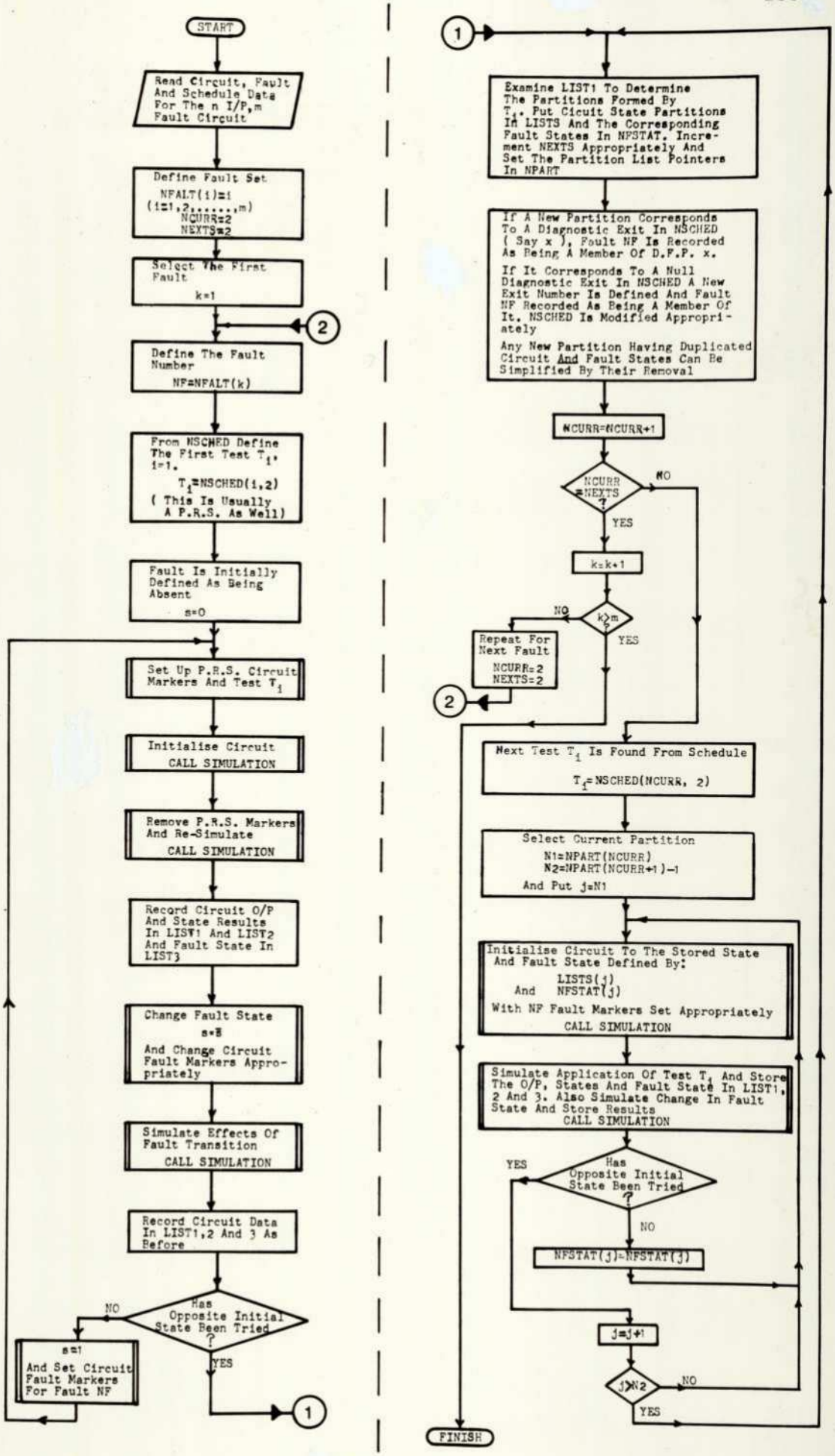


FIG.4.8. FLOW DIAGRAM OF PROGRAM 'DFP SEQ'

functions:

NFALT - a one-dimensional array for storing the m circuit faults.

NFSTAT - a one-dimensional array, directly equivalent to LISTT and LISTS, for storing the fault condition (absent/present) of the associated fault in LISTS.

LIST1, LIST2, LIST3 - three one-dimensional arrays for temporary storage of output results, circuit states and fault states respectively.

The program uses the SIMULATION routine as a subroutine. DFP SEQ automatically generates new diagnostic exit numbers for the null exits accessed by the faults under intermittent conditions, and the schedule is modified appropriately.

4.3.3 A Circuit Example

As an example of the d.f.p. calculation process consider the Seshu and Freeman circuit (38), Fig.4.4. The fault set is the same as used by those authors, Fig.4.5, and the schedule used is the one generated by SEQ ANALY, Fig.4.7(a). The computer run-time to determine the d.f.p.'s is 117 sec, including program compilation. This is a long run-time because of the number of calls in the SIMULATION subroutine which are made in the initialisation of the various circuit state conditions and the simulation of the fault state transitions required by the process.

A number of new exits are accessed under the

intermittent conditions. These are numbered -74 to -91, and are shown in the modified schedule, Fig.4.9. The calculated d.f.p. lists are shown in Fig.4.10.

The d.f.p. set, in conjunction with the modified solid fault testing schedule, can be used in an automatic circuit testing system to allow the diagnosis of intermittent faults. The process of repeated schedule application and the method of data interpretation, by intersecting the d.f.p.'s of the accessed diagnostic results, is the same as described for combinational circuits in Section 3.3.3. Diagnostic resolution is reduced compared with the case of solid faults, as with combinational circuits, through the occurrence of partial indistinguishability. In this circuit example sixteen solid faults can be uniquely diagnosed while only ten intermittent faults are completely resolved. Fig.4.11 shows the faults that are partially indistinguishable. For example, fault #4 cannot be distinguished from #6 (among others), whereas #6 can be distinguished from #4. Faults #14,16,17,18,21,25 and 43 are indistinguishable from all other intermittent faults and from the correctly working circuit.

Quotation of average test length is meaningless here since the point at which each fault is diagnosed is dependant on the particular pattern of fault occurrences which led up to the diagnosis. This of course is variable.

The effectiveness of this type of schedule at diagnosing intermittent faults is examined in Section 5.

Test No.	Test Value	Output 0	1	2	3
1	-4	12	-7	2	15
2	2	3	16	17	-64
3	0	18	4	0	-27
4	2	19	-79	5	-15
5	3	-81	-74	23	6
6	2	24	-85	-84	7
7	0	8	-49	-38	0
8	2	-78	9	-80	-29
9	3	-82	-75	-87	10
10	1	11	-88	-89	0
11	0	-60	-4	-91	0
12	2	-61	13	-11	0
13	3	0	14	0	-76
14	2	0	-2	-31	-77
15	2	-62	-51	0	-63
16	3	0	0	-65	-66
17	0	0	-67	0	-54
18	1	-68	-9	0	0
19	3	0	0	0	20
20	1	21	0	-12	0
21	0	22	0	0	0
22	2	-3	-70	0	-69
23	2	-46	-83	-71	-90
24	3	0	0	-86	25
25	1	26	0	0	0
26	0	-73	-72	0	0

FIG. 4.9. CIRCUIT EXAMPLE TEST SCHEDULE
(Modified by Intermittent Conditions)

Diagnostic Exit		Diagnostic Fault Profile
2	—	2
3	—	3
4	—	4,6,35,40,51,56
7	—	7
9	—	9
11	—	11
12	—	12
15	—	8,15,23,34,41,47,51
27	—	27,54
29	—	13,29,30,38,54
31	—	31
38	—	12,38,54
46	—	8,46
49	—	6,9,35,39,49,51,57
51	—	51
54	—	54
60	—	1,2,3, ... ,58,59
61	—	2,11,31,36,55
62	—	35,40,51
63	—	10,24,45
64	—	2,5,10,20,24,44,45,48,58,59
65	—	8,23,47
66	—	8,23,34,41,47,51
67	—	11,28,32,37,54
68	—	5,9,39,53,58
69	—	13,30
70	—	3,12,13,26,30,56
71	—	33,52
72	—	6,57
73	—	6,8,19,22,42,46,50,57
74	—	2,11,31,36,55
75	—	2,11,31,36,55
76	—	2,31
77	—	2,31
78	—	3,8,15,23,33,47,52
79	—	5,58
80	—	6,49,57
81	—	7
82	—	7
83	—	8
84	—	8,33,52
85	—	8,11,32,36,55
86	—	8,46
87	—	8,33,46,52
88	—	9,35,40,51
89	—	12,38,54
90	—	33,52
91	—	38,54

FIG. 4.10. CALCULATED DIAGNOSTIC-FAULT-PROFILES

Fault No.		Indistinguishable Faults
1	—	2,3,4, ... ,58,59
4	—	6,35,40,51,57
5	—	58
6	—	57
10	—	24,45
13	—	30
14	—	1,2,3, ... ,58,59
15	—	8,23,47
16	—	1,2,3, ... ,58,59
17	—	1,2,3, ... ,58,59
18	—	1,2,3, ... ,58,59
19	—	6,8,22,42,46,50,57
20	—	2,5,10,24,44,45,48,58,59
21	—	1,2,3, ... ,58,59
22	—	6,8,19,42,46,50,57
23	—	8,47
24	—	10,45
25	—	1,2,3, ... ,58,59
26	—	3,12,13,30,56
27	—	54
28	—	11,32,37,54
29	—	13,30,38,54
30	—	13
32	—	11
33	—	52
34	—	8,23,41,47,51
35	—	40,51
36	—	11,55
37	—	11,28,32,54
38	—	54
39	—	5,9,53,58
40	—	35,51
41	—	8,23,34,47,51
42	—	6,8,19,22,46,50,57
43	—	1,2,3, ... ,58,59
44	—	2,5,10,20,24,45,48,58,59
45	—	10,24
46	—	8
47	—	8,23
48	—	2,5,10,20,24,44,45,58,59
49	—	6,57
50	—	6,8,19,22,42,47,57
52	—	33
53	—	5,9,39,58
55	—	11,36
56	—	3,12,13,25,30
57	—	6
58	—	5
59	—	2,5,10,19,24,44,45,48,58

FIG. 4.11. INDISTINGUISHABLE FAULT SETS

4.4 TEST SCHEDULE CONSTRUCTION FOR INTERMITTENT FAULT DIAGNOSIS

4.4.1 A Philosophy For Efficient Testing

The philosophy developed for evaluating and selecting tests for intermittent fault location in combinational circuits, Section 3.4.1, applies to the case of sequential circuits as well. In the simplest terms this philosophy can be stated as:

(i) Taking maximum advantage of the time for which a fault is present.

(ii) At all stages of testing making maximum use of diagnostic information gained from previous stages.

These statements are discussed in more detail in Section 3.4.1.

Although the philosophy of testing and schedule construction, in a general sense, is the same for both sequential and combinational circuits, their fundamental differences in functioning will preclude the use of the same implementation techniques. This is a result of the way in which a fault-state transition can alter the circuit stored states when fault intermittency occurs. The construction process of the schedule and the form it takes must be capable of dealing with such effects. The effects of fault intermittency have been discussed in Section 4.3.1.

As in the construction of solid fault schedules, Section 4.2.2, the procedure consists of selecting tests which give best diagnostic resolution of the various partitions of faults. Initially the whole fault set of the circuit is in a single partition. When a sequence of test

has been chosen which allow each fault to be distinguished at least once from the correctly working circuit, these can be formed into a loop of tests around which the schedule can pass during periods of fault absence. The other partitions can be treated in a similar manner and a series of intersecting testing loops formed until maximum diagnostic resolution is reached. So far this is exactly the same as the combinational circuit schedule formation technique. One major difference lies in the fact that the intermittent faults, occuring within the restrictions of the well-behaved-fault definition, also cause circuit state changes which must be accounted for in the construction process. Partitions will contain multiple entries of those faults which cause circuit state changes.

To ensure that all of the fault conditions, including multiple entries, can be easily evaluated, the first test at the start of each testing loop can be a p.r.s. This forces the circuit into a limited number of know or predictable states and can be thought of as a process of initialising the circuit, as is done by the first test of the solid fault schedule in Section 4.2.2.

A duplicated fault number in a partition in the same fault state and the same circuit state conditions is redundant data. It can be removed from further consideration.

If a given test forms new partitions which contain the same numbers that followed the correctly working circuit (and all the fault-absent conditions), no new diagnostic data has been generated. These outputs can be

looped back to the start of the major loop. They are called secondary loops.

The next stage is to develop an implementation procedure for the construction of schedules conforming to the above stated philosophy and conditions.

4.4.2 Implementation Of Schedule Formation Process

The Sequential Analyser technique can be used as the basis for a schedule construction process. The distinguishability criterion permits maximum diagnostic resolution to be found when each test is evaluated for inclusion in the testing sequences.

The need to deal with intermittent faults, the different circuit states resulting from fault state transitions and the general complexity of programming the Sequential Analyser make implementation a very difficult task. For this reason it was decided to reduce the number of heuristics by removing parts of the NEXT BEST TEST. The lack of success of this heuristic in finding good tests in the SEQ ANALY program (Section 4.2.4) justify to some extent its removal. Only the best test is selected at each stage, other good ones are completely rejected.

A process, based on the injection of faults into a simulated model of the circuit under consideration, for constructing diagnostic test schedules for intermittent faults is as follows:

(i) Define:

(a) the circuit to be tested.

(b) the set of circuit faults. These initially form the current partition of faults. Make a copy of the current partition in COPY.

(c) the set of p.r.s.

(d) the step limits for controlling heuristic switching.

(ii) Choose at random one p.r.s. Extract from this the input test value T_i.

(iii) Initialise the simulation model of the circuit in the presence of the p.r.s. and T_i for the fault-absent (or fault-present) condition of each of the faults defined by the current partition. Since this is a p.r.s., variations in fault states prior to its application are irrelevant.

(iv) Simulate removal of p.r.s. for each fault and record the circuit output, circuit state and fault state in the i^{th} current list corresponding to T_i. Simulate a fault transition from fault-absent (-present) to fault-present (-absent) for each fault and similarly record appropriate circuit data.

(v) Repeat from (iii) for the fault-present (or fault-absent) initial fault state.

(vi) Examine i^{th} current list to determine the partitioning of circuit faults (including multiple entries). The new partition associated with the correctly working circuit is identified and the fault and circuit states are recorded. All other new partitions are processed to remove multiple entries (since the next test applied to them will be a p.r.s.). Any of these

which contain the same faults as the correctly working circuit are classified as secondary loops in the schedule and are not considered further. If a partition contains only one number, it is a unique diagnosis. This is recorded. Remove from COPY any faults which have now been distinguished from the correctly working circuit. Store all other new partitions.

(vii) If COPY is now empty, all faults of the current partition have been distinguished at least once from the correctly working circuit, go to (XV1). Otherwise, continue.

(viii) The partition corresponding to the correctly working circuit now becomes the current partition. Intialise the circuit to the state and fault conditions defined for the faults in the current partition. Select a possible next input test T_i ($i = 1 \rightarrow n$ for an n input circuit). Simulate the application of T_i for each fault case and record the fault numbers, circuit output, state and fault state in the i^{th} current lists associated with T_i. Simulate each fault changing state and store the appropriate data in the i^{th} current list.

(ix) Re-initialise the circuit to the states and fault conditions as defined at the beginning of (viii). Simulate the change of fault state for each one. Simulate the application of T_i for each fault condition. Store the results as before. Simulate the change of fault state for each fault condition. Store the results as before.

(x) Repeat from (viii) for each other T_i (i = 1,2,.....,n). Eliminate any test which results in a circuit oscillation.

(xi) By examining the data stored in the current lists apply the distinguishability criterion to select which T_i gives the best diagnostic resolution to those faults not yet distinguished from the correctly working circuit (i.e. those listed in COPY).

(xii) If each figure of merit, α_i (i = 1,.....,n) is zero then no relevant distinguishability is possible, so go to (xiii). Otherwise, the T_i with the largest α_i is the best test, go to (vi).

(xiii) Apply WANDER heuristic by randomly selecting T_i from the possible set of tests and repeating from (vi). If WANDER does not find a good test within a limited number of tests, go to (xiv).

(xiv) Apply RESET heuristic by repeating from (ii). If WANDER and RESET together cannot find a good test within a limited number of resets, go to (xv).

(xv) The faults remaining in COPY are indistinguishable. This data is recorded and the primary loop is closed.

(xvi) Select one of the new partitions as yet un-processed as the current partition. Make a copy of the current partition in COPY. Repeat from (ii). If all partitions have been processed, the procedure is finished. The schedule is formed from the recorded testing data.

A computer programmed version of the above procedure was written in FORTRAN IV and named SEQ ANALY(INT). The

flow diagram is shown in Fig.4.12. The named arrays and variables in the diagram have, in general, been previously defined (3.2.3, 3.4.2, 4.2.3 and 4.3.3). The only difference here is in the definitions of NSTORA and NSTORB. Also new arrays NSTORC and NSTORD are introduced:

NSTORA, NSTORB, NSTORC and NSTORD - four two-dimensional arrays for temporary storage of fault numbers, output results, circuit-states and fault-states respectively. The (i,j)th element of each contains the data on the appropriate circuit conditions defined for the j^{th} listed fault number when stimulated by the i^{th} test.

Data supplied to the program consists of a circuit description, a description of the fault set, the set of p.r.s. and the step limits for controlling WANDER and RESET heuristics.

4.4.3 A Circuit Example

As an example of the schedule construction process consider the Seshu and Freeman circuit (38), Fig.4.4. The fault set is the same as used by those authors, Fig.4.5.

The effectiveness of the heuristics can be evaluated by computing the results generated by SEQ ANALY(INT) for various settings of NUMW and NUMR, the heuristic step limits. The effectiveness of the construction process can be gauged from such factors as:

(i) Program run-time.

(ii) Diagnostic resolution of the schedule.

(iii) Total number of tests in the schedule.

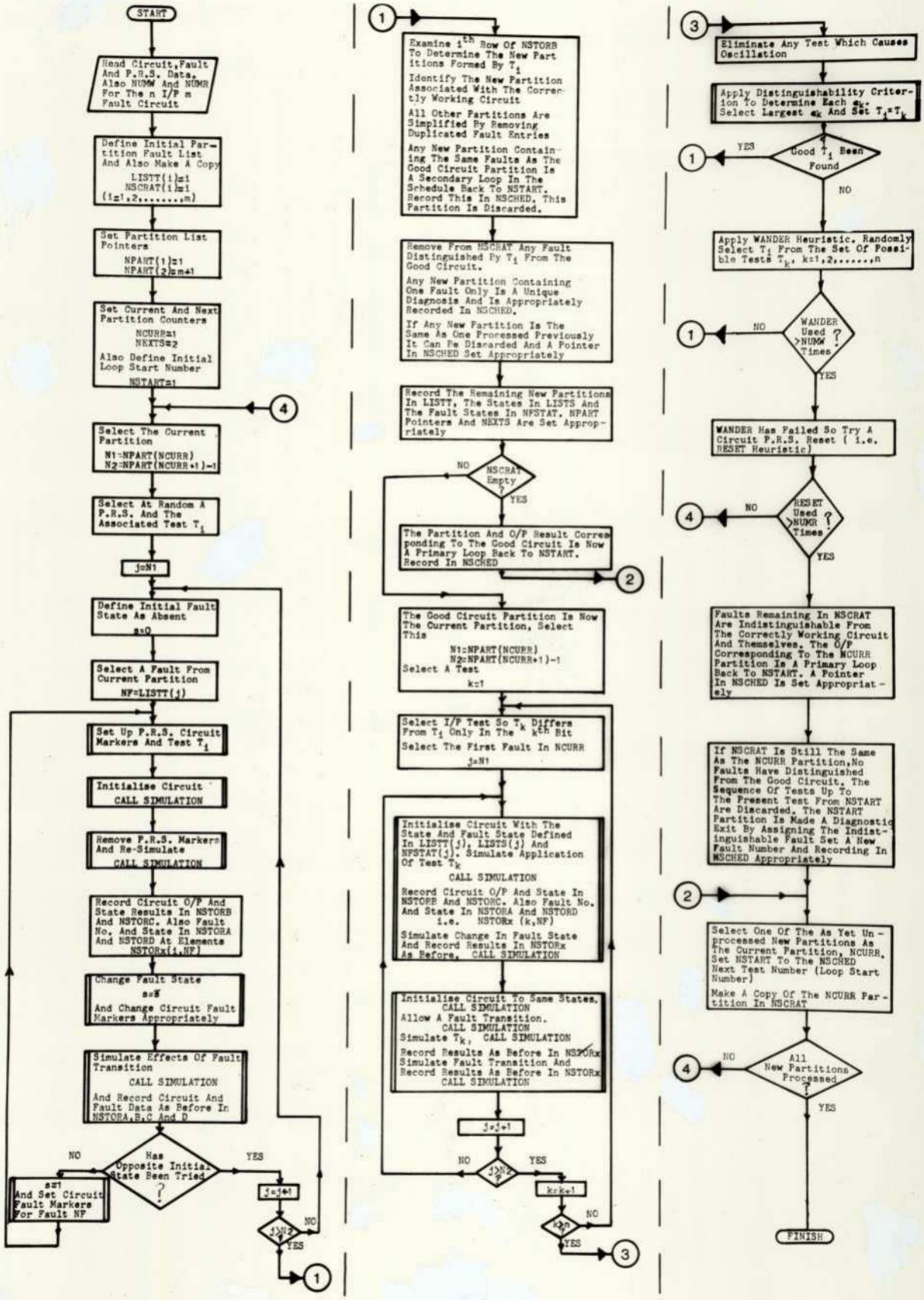


FIG. 4.12. FLOW DIAGRAM OF PROGRAM 'SEQ ANALY(INT)'

Since this is a schedule for intermittent faults, the average test length cannot be directly calculated. This aspect will be considered in more detail in Section 5 when the schedule is evaluated under various simulated intermittent conditions. The other factors listed above can be directly found by running the program with different heuristic control parameters. As with SEQ ANALY, Section 4.2.4, it is up to the user of the schedule to assign weights to the importance of each of these factors, depending on the needs.

NUMW was varied between 0 and 3 for settings of NUMR between 0 and 3. Because of the existence of partial indistinguishability the definition of diagnostic resolution in terms of numbers of distinguishable fault-pairs is no longer valid. Another possible measure of diagnostic resolution can be the sum of the total number of faults that are distinguishable from each other fault, or the number of 'fault-singles'. In a circuit with all faults completely distinguishable, where the fault-pair definition is valid, the number of fault-singles is exactly double the number of fault-pairs. The results for run-time, diagnostic resolution and the total number of tests in the schedule are shown in Fig.4.13(a),(b) and (c) respectively. The run-times required for NUMR > 1 are excessive and so the plots are not complete.

At NUMW = 2 and NUMR = 0 the diagnostic resolution reaches a value of 2890 fault singles. The number of tests in the schedule is 84. This took 742 seconds of computer

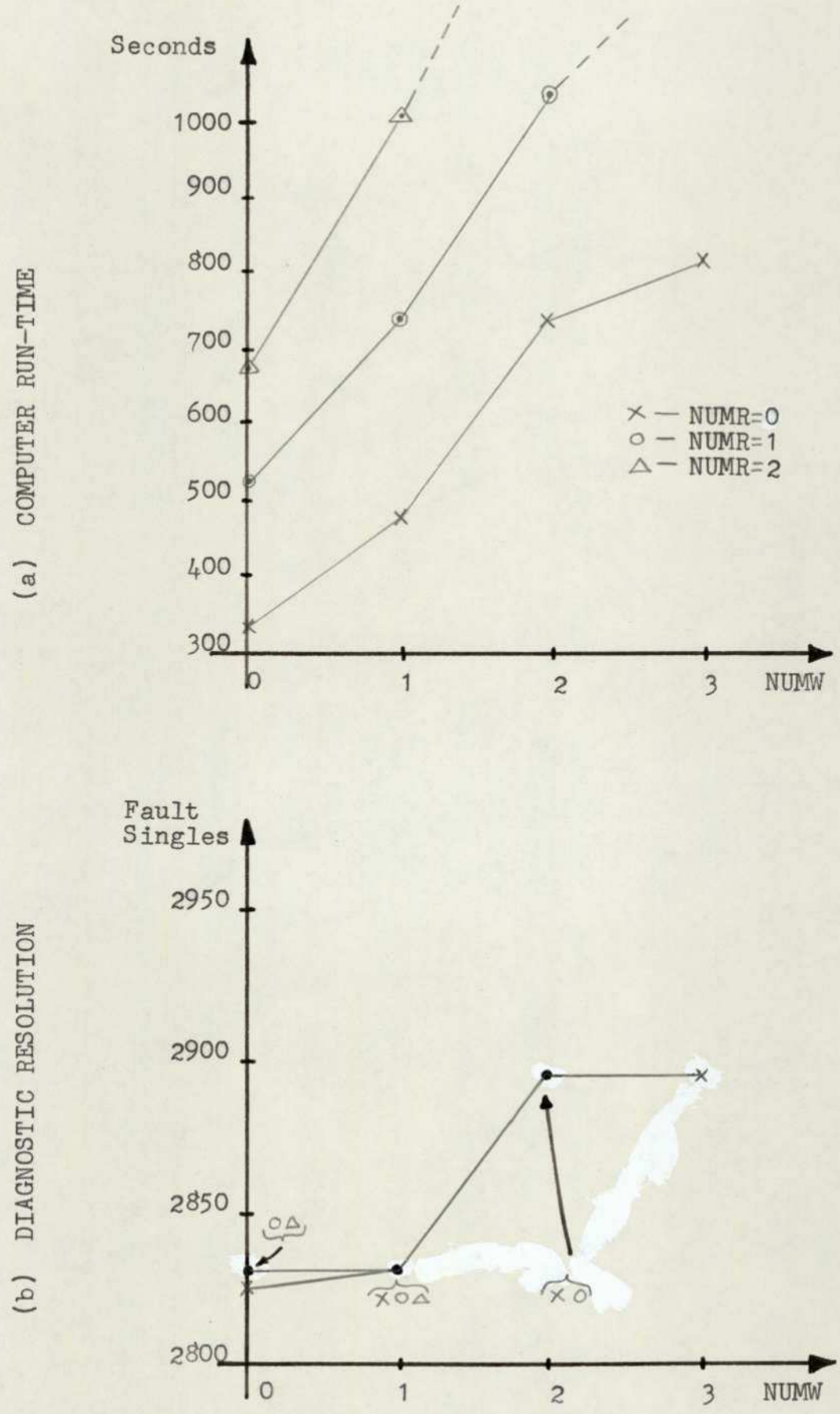


FIG. 4.13. EVALUATION OF HEURISTIC EFFECTIVENESS

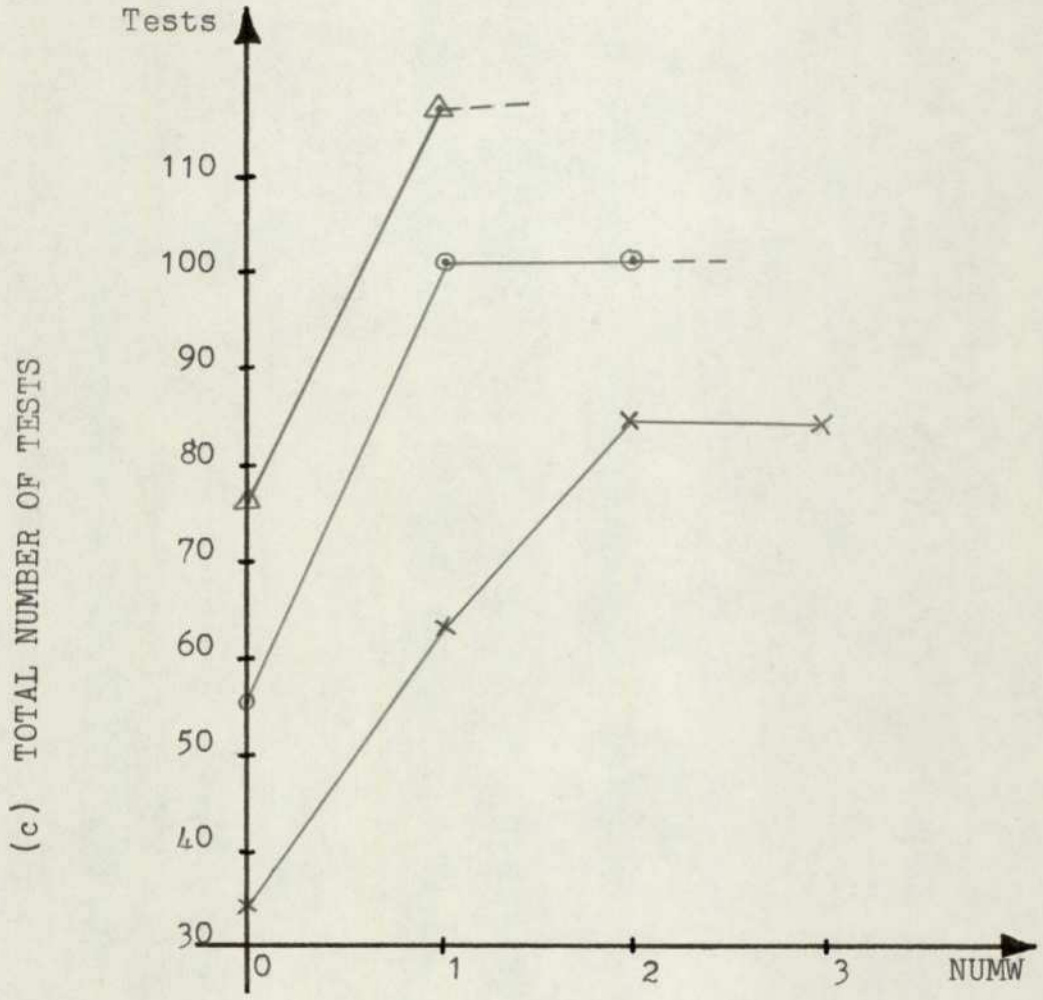

(c) TOTAL NUMBER OF TESTS
Tests
110
100
90
80
70
60
50
40
30
0
1
2
3
NUMW

FIG. 4.13. (cont.

run-time to evaluate. The actual schedule is shown in Fig.4.14. Diagnostic exists numbered -60 to -67 correspond to the sets of indistinguishable faults shown in Fig.4.15(a). The partially indistinguishable faults are listed in Fig.4.15(b). The numbers in the second column of this listing correspond to the loop-start test number at which the faults cannot be further distinguished.

These heuristics have failed to some extent to produce consistent results. An example can be seen in Fault #51. This has several unique diagnostic exits, while at another point in the schedule it is indistinguishable from faults #35 and #40, indicated by result -65. This apparent anomaly is a result of the local optimisation features of the technique. This does, however, indicate another heuristic that could reduce the problem. "If a fault has been distinguished previously, NUMW and NUMR can be extended above the defined limits until a good result is found, or up to some limit where further searching is uneconomical".

4.5 DISCUSSION

Consideration of only asynchronous sequential circuits in this section of the project is a basic limitation since most modern logic systems are synchronous or a mixture of both types. However, the extra time and effort to implement a more comprehensive analysis technique was not available.

The implementation of schedule construction procedures for both solid and intermittent faults are based on the heuristics of Seshu (34). These were simplified to some extent to ease the

Test No.	Test Val.	Output 0	1	2	3
1	-4	12	-7	2	13
2	2	3	14	15	16
3	0	17	4	0	18
4	2	19	-61	5	20
5	3	-7	12	21	6
6	2	22	24	23	7
7	0	8	25	26	0
8	2	27	9	-63	28
9	3	-7	12	21	10
10	1	11	29	26	0
11	0	1	30	-64	0
12	-18	33	0	0	0
13	-18	31	-65	0	0
14	-3	45	44	0	0
15	-3	0	41	0	-54
16	-3	0	36	0	0
17	-3	0	46	0	0
18	-4	0	0	50	0
19	-18	51	0	-12	0
20	-3	54	53	0	0
21	-18	55	0	0	0
22	-18	60	0	0	0
23	-3	23	65	0	0
24	-18	63	0	0	0
25	-4	0	0	67	-65
26	-18	72	0	26	0
27	-3	27	73	0	0
28	-4	0	0	74	0
29	-18	29	-65	0	0
30	-3	0	75	62	0
31	3	0	0	32	13
32	2	13	-51	0	-66
33	3	12	0	34	0
34	2	35	0	-11	-2
35	3	0	-67	0	12
36	3	0	-2	0	37
37	2	0	0	0	38
38	0	39	0	0	0
39	2	40	0	0	0
40	3	-2	0	16	-66
41	3	0	-11	0	42
42	2	0	43	0	15

(cont.

No.	Val.	0	1	2	3
43	-4	-11	0	43	0
44	0	14	-51	0	0
45	-18	77	0	0	0
46	0	47	-9	0	0
47	1	48	0	0	0
48	3	0	0	49	0
49	2	17	0	0	-61
50	2	18	0	-54	0
51	0	52	0	0	0
52	2	-3	19	0	-60
53	0	20	-51	0	0
54	-4	0	0	80	0
55	0	56	0	0	0
56	2	23	57	0	0
57	3	0	0	21	58
58	2	59	-8	23	21
59	-18	81	0	0	0
60	0	61	-62	0	0
61	2	-8	62	-62	0
62	3	0	0	59	22
63	3	-67	0	64	0
64	2	24	-8	43	0
65	3	0	0	23	66
66	2	-8	-8	23	23
67	2	68	-51	0	0
68	0	-9	69	0	0
69	2	0	0	70	-51
70	3	0	0	0	71
71	2	-62	0	0	25
72	0	26	0	-64	0
73	3	0	0	23	27
74	1	28	0	-64	0
75	0	30	76	0	0
76	-18	83	-65	0	0
77	0	78	0	0	0
78	2	45	79	0	0
79	3	0	0	-8	45
80	2	54	45	0	0
81	0	82	0	0	0
82	2	-8	59	0	0
83	0	84	76	0	0
84	2	0	76	-62	0

FIG. 4.14. CIRCUIT EXAMPLE TEST SCHEDULE FOR INTERMITTENT FAULT DIAGNOSIS

New Diagnosis No.		Associated Indistinguishable Fault Sets
60	—	13,30
61	—	5,58
62	—	6,57
63	—	6,49,57
64	—	38,54
65	—	35,40,51
66	—	10,24,45
67	—	11,36,55

FIG. 4.15.(a). INDISTINGUISHABLE FAULT SETS

Fault No.	Loop Start Test No.	Associated Indistinguishable Faults
1	1	2,3,4, ... ,58,59
3	27	15,23,47
4	30	6,57
5	16	2,10,20,24,44,45,48,58
12	26	38,54
13	28	29,30,38,54
14	1	1,2,3, ... ,58,59
15	27	3,23,47
15	54	8,23,47
16	1	1,2,3, ... ,58,59
17	1	1,2,3, ... ,58,59
18	1	1,2,3, ... ,58,59
19	22	6,8,22,42,50,57
20	16	2,5,10,24,44,45,48,58,59
21	1	1,2,3, ... ,58,59
22	22	6,8,19,42,50,57
23	45	8,47
23	27	3,15,47
25	1	1,2,3, ... ,58,59
26	19	3,12,13,30,56
27	18	54
28	15	11,32,37,54
29	28	13,30,38,54
30	28	13,29,38,54
31	12	2,11,36,55
32	43	11
33	23	52,8
34	14	41,51
34	20	41,51
37	15	11,28,32,54,
39	17	5,9,53,58
41	14	34,51
41	20	34,41
42	22	6,8,19,22,50,57
43	1	1,2,3, ... ,58,59
44	16	2,5,10,20,24,45,48,58,59
46	59	8
47	45	8,23
47	27	3,15,23
48	16	2,5,10,20,24,44,45,58,59
49	25	6,9,35,40,51,57
50	22	6,8,19,22,42,57
52	23	8,33
53	17	5,9,39,58
56	19	3,12,13,26,30
58	16	2,5,10,20,24,44,45,48,59
59	16	2,5,10,20,24,44,45,48,58

FIG. 4.15.(b). INDISTINGUISHABLE FAULT SETS

problems of implementation. The race-analysis used in the original Sequential Analyser of Seshu and Freeman (38) was not attempted at all. The justification for this is that the schedules are being generated for ideal components in ideal circuits and that critical races are not manifested. In other words, stable states in the simulated circuits are predictable and repeatable. A future development of the SIMULATION subroutine to three-level logic (see Section 2.2) would permit race-analysis to be performed in a much simpler way than Seshu and Freeman's Huffman analysis technique.

It is possible to conceive of a fault-state transition occurring more than twice in a Test/Sample cycle of the schedule. Such an event could possibly invalidate both the d.f.p. and schedule construction processes. This could be rectified by determining which fault transitions cause circuit state changes. Those that do not directly cause state changes can occur as often as possible without any effect. If the others are allowed to occur up to a limited number of times in the simulation model and the new circuit states recorded and analysed as before, the possibility of an occurrence that has not been catered for is then very small.

The computer run-times of the procedures are quite long and represent an important economic limitation to the use of the programs. The cause of this is related to the number of times each fault and circuit state need to be initialised into the circuit model and the large number of fault transitions that have to be simulated for each of the input tests evaluated. Each of these operations is a call on the SIMULATION subroutine. One

possible solution to reduce this run-time would be to use a lower level language and perhaps perform simultaneous simulation of a number of fault conditions by utilising all the bits in the computer word. This would certainly increase the computation speed and reduce storage problems.

Extensions of the schedule construction processes to deal with synchronous circuits could be managed by utilising a technique such as the d-Algorithm to select sets of possible tests. These could then be evaluated using the Sequential Analyser. This is a method mentioned previously in Section 4.2.1(c).

The heuristics, in general, have not proved too successful. NEXT BEST TEST has been eliminated from use in SEQ ANALY(INT). WANDER produced the best results of all three, as can be seen from the graphs in Section 4.2.4 and 4.4.3.

5. EVALUATION OF INTERMITTENT FAULT DIAGNOSTIC TESTING SCHEDULES

The operational characteristics of combinational and sequential circuit diagnostic testing schedules in the presence of intermittent faults can be evaluated through a simulation process. This requires simulation of both the circuit under test and the automatic testing system. Faults can be injected during simulation using a particular model of fault intermittency. This model permits different types of intermittency to be simulated.

Different schedules can be evaluated and compared under varying fault conditions.

5.1 INTRODUCTION

Having formulated various procedures for the construction of intermittent fault diagnostic testing schedules, the next stage is the evaluation of the effectiveness and capabilities of such schedules under actual intermittent fault conditions. This will also be a check on the validity of the construction procedures.

Tsiang and Ulrich (73) have described the method of physical injection of solid faults in circuits as one technique of test schedule evaluation used by Bell Telephone Laboratories. This involves the actual insertion of the fault condition into a working circuit. Such a technique requires special circuitry, can be very time consuming, and requires a commitment to a hardware design at an early state in the design process. The use of a simulation technique removes many of these problems. Here, the circuit, the fault and the automatic testing system can all be simulated in a single procedure.

Under intermittent fault conditions the requirement of flexibility for modelling intermittent fault injection makes simulation appear an even more effective approach than the physical technique. With the availability of the simulation subroutine SIMULATION (Section 2) it was decided to use this method for investigating the effectiveness of the various schedules.

Simulation of circuits and fault conditions has been discussed previously (Section 2). The automatic circuit tester can also be simulated. For different types of schedules the tester performs different operations. The d.f.p. process, described in Section 3.3.4, consists of applying the tests from an ordinary solid fault schedule, examining the output results

until a diagnosis is found, and then intersecting d.f.p. data to find a unique diagnosis. The schedule designed specifically for diagnosing intermittent faults is an ordinary schedule of the same form as the solid fault schedule but consisting of considerably more tests. Each of these testing procedures, and the associated testing and sampling operations, therefore require different simulation implementations.

Small sized combinational circuits can be simulated very easily and efficiently using a Fault-Matrix table look-up type of simulation process. For this reason different implementations of the evaluation techniques are also required for combinational and for sequential circuits.

Fault intermittency can manifest itself in a whole range of different forms, from sparse occurrences lasting short or long periods of time to common ocurrences also lasting short or long periods. Evaluation procedures must take this into account to enable meaningful results to be obtained. For this reason a satisfactory model of the intermittent fault must be developed which can be utilised during the simulation process.

Breuer (58) has suggested that a suitable statistical model can be based on a simple two-state first-order Markov process. By defining and varying the probabilities associated with a transition from one fault state to the other, and through use of a random number genertor, a method can be developed to produce the whole spectrum of intermittent occurrences during simulation. The fault model and its development into a useful evaluation tool is described in subsection 5.2.

Combinational circuits schedule evaluation is covered in sub-section 5.3. Sequential Circuit schedule evaluation is covered in sub-section 5.4. An extension to the use of the fault model is developed in 5.5. The evaluation results are discussed in sub-section 5.6.

5.2 A STATISTICAL MODEL OF INTERMITTENT FAULT OCCURRENCE

5.2.1 Markov Processes

Breuer (58) has suggested that intermittent fault occurrences can be suitably modelled by a two-state first-order Markov process.

Consider a process that takes on a finite number of possible values

$\{X_n,\ n = 0,1,2,.....\}$

denoted by the non-negative integers

$\{0,1,2,.....\}$.

$X_n = i$, states that the process is in state i at time n. Whenever the process is in state i let there be a fixed probability P_{ij} that the next state, at time n+1, will be j. Therefore, for all states $i_o,i_1,i_2,.....,i_{n-1},i,j$ and all $n \geqslant 0$.

$$P\{X_{n+1} = j \,|\, X_n = i,\ X_{n-1} = i_{n-1},\ X_{n-2} = i_{n-2},\ .....$$

$$...X_1 = i_1,\ X_o = i_o\} = P\{X_{n+1}\ j \,|\, X_n = i\} = P_{ij}.$$

This says that in such a process the condition of a future state X_{n+1}, given past states $X_o,X_1,.....,X_{n-1}$ and the

present state X_n, is independant of the past states and depends only on the present state. P_{ij} therefore represents the probability that the process will, when in state i, next make a transition to state j. Such a process is termed a first-order Markov Chain (or Markov Process). Given that the process must make some state transition (including $X_n = i \rightarrow X_{n+1} = i$) at time n+1:

$$\sum_j P_{ij} = 1, \quad i = 0,1,2,.....$$

A k^{th} order Markov process defines the conditions under which the probability of state $X_{n+1} = j$ is dependant on the k^{th} previous process states, that is

$$P\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, X_{n-2} = i_{n-2}, ...$$

$$.... X_1 = i_1, X_o = i_o\} = P\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1},$$

$$X_{n-2} = i_{n-2}, ..., X_{n-k} = i_{n-k}\}$$

The state transition probabilities, P_{ij}, for a first-order Markov process can be conveniently represented by a transition matrix P:

$$P = \begin{vmatrix} P_{oo} & P_{o1} & P_{o2} & ..... & P_{oj} & ..... \\ P_{1o} & P_{11} & P_{12} & ..... & P_{1j} & ..... \\ P_{2o} & P_{21} & P_{22} & ..... & P_{2j} & ..... \\ \vdots & \vdots & \vdots & & \vdots & \\ P_{io} & P_{i1} & P_{i2} & ..... & P_{ij} & ..... \\ \vdots & \vdots & \vdots & & \vdots & \end{vmatrix}$$

where P_{ij}, $i = 0,1,2,.....$ and $j = 0,1,2,.....$, is the probability of a transition to state j given that the present state is i. When referring to a specific physical process i and j are delimited appropriately. For example, in a two state system i and j are only 0 or 1.

5.2.2 Breuer's Model

Breuers' model (48) of fault intermittency allows the fault to be either present (1) or absent (0). The states 1 and 0 are assigned arbitarily. No other intermediate conditions are possible. The state of the fault at the next time step is declared to be dependant only on the present state. Such an intermittent fault can therefore be modelled by a two-state first-order Markov process. This is shown diagramatically in Fig.5.1, where s and r represent the probabilities of state transition $0 \rightarrow 1$ and $1 \rightarrow 1$, respectively. $\bar{s} = 1 - s$ and $\bar{r} = 1 - r$ are the probabilities of transitions $0 \rightarrow 0$ and $1 \rightarrow 0$.

In terms of the general Markov process representation $\{X_n, n = 0,1,2,.....\}$ can only take on the state values $\{0,1\}$. The transition matrix for the fault conditions is therefore reduced to:

$$P = \begin{vmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{vmatrix} = \begin{vmatrix} \bar{s} & s \\ \bar{r} & r \end{vmatrix}$$

If p_n $(\bar{p}_n = 1 - p_n)$ is the probability of being in state 1(0) at time t_n, then

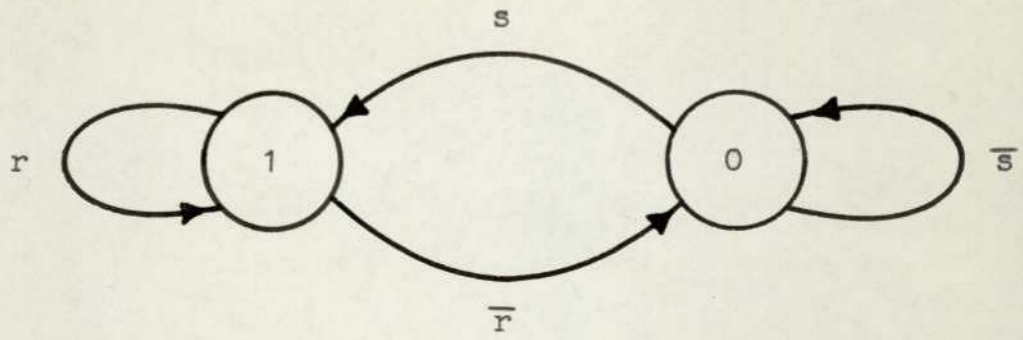


FIG. 5.1. TWO-STATE FIRST-ORDER MARKOV MODEL OF AN INTERMITTENT FAULT

$$\left(p_{n+1},\ \overline{p_{n+1}}\right) = \left(p_n,\ \bar{p}_n\right).\begin{vmatrix} \bar{s} & s \\ \bar{r} & r \end{vmatrix},$$

giving

$$p_{n+1} = p_n.r + \left(1 - p_n\right).s$$

$$= s + p_n(r - s) \qquad \ldots\ (1)$$

and

$$\overline{p_{n+1}} = \overline{p_n}.\bar{s} + \left(1 - \overline{p_n}\right).\bar{r}$$

$$= \bar{r} + \overline{p_n}.(\bar{s} - \bar{r}). \qquad \ldots\ (2)$$

The solid-fault and no-fault conditions are special cases of this model defined by $r=p_o=1$ and $\bar{s}=\overline{p_o}=1$, respectively, where $(p_o,\overline{p_o})$ is the initial fault state probability distribution at time t_n, $n=0$.

The model can now be used to consider the probabilities of being in a fault state 1(0) given that at Δ previous time steps the state was 1(0) or 0(1). Element ${}^{\Delta}P_{oo}$ of the matrix

$${}^{\Delta}P = \begin{vmatrix} {}^{\Delta}P_{oo} & {}^{\Delta}P_{o1} \\ {}^{\Delta}P_{1o} & {}^{\Delta}P_{11} \end{vmatrix}$$

is the probability that the fault will be in state 0 given that Δ time steps earlier it was in state 0. Similar definitions apply for ${}^{\Delta}P_{o1}$, ${}^{\Delta}P_{1o}$ and ${}^{\Delta}P_{11}$. The formulation of ${}^{\Delta}P$ can be performed in the following manner:

Consider ${}^{\Delta}P_{oo}$.

Equation (2) is:

$$\bar{p}_{n+1} = \bar{r} + \bar{p}_n.(\bar{s} - \bar{r}) = {}^1P_{oo}.$$

It follows that:

$$\bar{p}_{n+2} = \bar{r} + \bar{p}_{n+1}.(\bar{s} - \bar{r}) = \bar{r} + \bar{p}_n.(\bar{s} - \bar{r})^2 + \bar{r}(\bar{s} - \bar{r}) = {}^2P_{oo}$$

and

$$\bar{p}_{n+3} = \bar{r} + \bar{p}_{n+2}.(\bar{s} - \bar{r}) = \bar{r} + \bar{p}_n.(\bar{s} - \bar{r})^3 + \bar{r}(\bar{s} - \bar{r})^2 + \bar{r}.(\bar{s} - \bar{r}) = {}^3P_{oo}$$

etc., and in general:

$$\bar{p}_{n+\Delta} = \bar{r} + \bar{p}_{n+\Delta-1}.(\bar{s} - \bar{r}) = \bar{r} + \bar{p}_n.(\bar{s} - \bar{r})^\Delta + \bar{r}(\bar{s} - \bar{r})^{\Delta-1} + \bar{r}.(\bar{s} - \bar{r})^{\Delta-2} + \ldots\ldots + \bar{r}.(\bar{s} - \bar{r}) = {}^\Delta P_{oo}$$

So,

$$\bar{p}_{n+\Delta} = \bar{p}_n.(\bar{s} - \bar{r})^\Delta + \sum_{m=0}^{\Delta-1} \bar{r}.(\bar{s} - \bar{r})^m = {}^\Delta P_{oo}.$$

State at time n is defined as being 0, therefore $\bar{p}_n = 1$, giving:

$$\bar{p}_{n+\Delta} = (\bar{s} - \bar{r})^\Delta + \sum_{m=0}^{\Delta-1} \bar{r}.(\bar{s} - \bar{r})^m$$

Now, $\sum_{m=0}^{\Delta-1} \bar{r}.(\bar{s} - \bar{r})^m$ is a geometric series of the form:

$$a + az + az^2 + \ldots\ldots + az^{m-1}$$

which generalises to:

$$\frac{a(1 - z^m)}{(1 - z)} .$$

Therefore,

$$\bar{P}_{\Delta+n} = (\bar{s} - \bar{r})^{\Delta} + \frac{r\left[1 - (\bar{s} - \bar{r})^{\Delta}\right]}{1 - (\bar{s} - \bar{r})}$$

$$= \frac{(\bar{s} - \bar{r})^{\Delta} + r\left[1 - (s - r)^{\Delta}\right]}{1 - (\bar{s} - \bar{r})}$$

$$= \frac{(r - s)^{\Delta} s - (r - 1)}{1 + s - r} \qquad \ldots\ (3)$$

$$= {}^{\Delta}P_{oo}.$$

Equation (3) is the same as stated by Breuer (58).

Similar calculatiions give:

$${}^{\Delta}P_{o1} = \frac{(s - r)^{\Delta} \bar{s} + r}{1 - s + r} \qquad \ldots\ (4)$$

$${}^{\Delta}P_{1o} = \frac{(s - r)^{\Delta} r - (s - 1)}{1 + r - s} \qquad \ldots\ (5)$$

$${}^{\Delta}P_{11} = \frac{(r - s)^{\Delta} \bar{r} + s}{1 + s - r} \qquad \ldots\ (6)$$

Another solution can be obtained through repeated matrix multiplication because

$${}^{\Delta}P = (P)^{\Delta},$$

which comes from the Chapman-Kolmogorov equation (94).

The initial matrix is:

$$P = \begin{vmatrix} P_{oo} & P_{o1} \\ P_{1o} & P_{11} \end{vmatrix}$$

The elements of the final matrix $^{\Delta}P$ then contain the appropriate probabilities after Δ steps, as before.

Given that Δ times steps earlier the probability of being in state 0 or 1 was α and β respectively, the probability of being in state0 at the present time is

$$\bar{p}_{\Delta} = \alpha.^{\Delta}P_{oo} + \beta.^{\Delta}P_{1o} \quad \ldots\ (7)$$

and of being in state 1 is

$$p_{\Delta} = \alpha.^{\Delta}P_{o1} + \beta.^{\Delta}P_{11} \quad \ldots\ (8)$$

where $\alpha + \beta = 1$.

Examination of equations (3),(4),(5) and (6) indicate that $^{\Delta}P_{ij}$, i=0,1 and j=0,1, tends to a limit as Δ tends to ∞, because 0 < (r and s) < 1. In other words a limit is reached such that $^{\Delta+1}P_{ij} = {}^{\Delta}P_{ij}$. These values represent the stationary probabilities that the fault will be in state 1(0) at a given time, and also equal the long-run proportion of time that the fault is in state 1(0). These stationary probabilities can be defined as β_{st} and α_{st} for states 1 and 0 respectively where $\alpha_{st} + \beta_{st} = 1$. Equation (1) therefore becomes

$$\beta_{st} = \beta_{st}.r + \alpha_{st}.s.$$

Then

$$\beta_{st} = \beta_{st}.r+(1 - \beta_{st}).s$$

since $\alpha_{st} + \beta_{st} = 1$.

This leads to

$$\beta_{st} = \frac{s}{1 + s - r}. \qquad \ldots\ (9)$$

From equation (2) we can similarly obtain

$$\alpha_{st} = \frac{1 - r}{1 + s - r}. \qquad \ldots\ (10)$$

If the intermittency has been occurring for a period of time such that stationary conditions exist, α and β of equations (7) and (8) can be replaced by α_{st} and β_{st}.

The reader will note that the model of fault intermittency discussed above fits the well-behaved fault restrictions used during schedule construction in Section 3 and 4. In the case of combinational circuits each time step of the Markov process can correspond to a single Test/Sample cycle of the test schedule. For sequential circuits two steps of the Markov process can occur in each cycle.

Breuer (58) has used this fault model to investigate some of the theoretical aspects of generating fault detection testing sequences for intermittent faults. The probability calculations are used to aid in the choice of those tests which give best detection capabilities for the highest probability patterns of fault occurrence.

Several limitations to this model are identifiable:

(i) no data exists (at present) for relating physical properties of circuit components, faults or environment to the values of r and s.

(ii) particular kinds of faults may not be accurately modelled by a first-order Markov process. For example, the probability of transition from states 0 to 1 may vary due to thermal stimulus resulting from previous intermittent occurrences. Such a fault may be modelled more satisfactorily by a k^{th} order process.

5.2.3 Implementation Of The Model

The fault model described in the previous section (5.2.2) was chosen as the tool for evaluating the intermittent fault schedules. The limitations of the model do not restrict its use here because no assumptions are made as to the physical nature and relationships of the parameters which define r and s except that it is a first-order Markov process. If the evaluation takes place over a range of r and s values then all classes of intermittency have been covered to some extent.

Given that the fault is in state 1, for example, at time t_n the model could be used to calculate the state at time t_{n+1} if the values of r and s are defined. This could be implemented using a random number generator. Assume that the generated number x is $0 \leqslant x \leqslant 1$. If $x \leqslant r$ then a fault transition to state 0 occurs, otherwise it stays in state 1. Assuming that the random generator produces numbers between 0 and 1.0 with a rectangular probability distribution, the probability of staying in state 1 is equal to r and of changing to state 0 is $\bar{r}$. Similarly, if the fault is in state 0 at time t_n, the state at t_{n+1} is 0 if $x \leqslant \bar{s}$ and 1 otherwise. The initial state

of the fault when the circuit is switched on can be determined in a similar manner using α and β. Initial fault state is 1 if $x \leqslant \beta$ and 0 otherwise.

This can be formalised into a procedure which can be accessed whenever the next or initial fault state is to be calculated.

(i) Generate a random number x.

(ii) If the present fault state has been previously defined go to (iv). Otherwise, calculate the initial fault state by going to (iii).

(iii) If $x \leqslant \beta$, set initial fault state to 1 (present). Otherwise, set initial fault state to 0 (absent). Go to (vi).

(iv) If present fault state is 0 go to (v). Present fault state is therefore 1, so if $x \leqslant \bar{r}$ set next state to 0, otherwise no change. Go to (vi).

(v) Present fault state is 0. If $x \leqslant \bar{s}$ there is no state change, otherwise next state is set to 1.

(vi) Procedure finished.

The above procedure was implemented as a FORTRAN IV routine using the I.C.L. FPMCRV function to generate pseudo-random numbers. FPMCRV is used by initially defining a seed number XRAN of value $0 \leqslant$ XRAN $\leqslant 1.0$ (but XRAN $\neq 0.5$). Thereafter a random number is generated by calling the function XRAN = FPMCRV(XRAN). The value of XRAN is used to determine the next or initial fault state. XRAN is also the seed for the next random number to be generated. Different initial seeds generate different sequences of

random numbers and conversely the same sequence of (pseudo) random numbers is generated by the same initial seeds. This is a useful feature which enables different schedules to be evaluated under exactly similar intermittent fault occurrences.

The flow diagram of the computer programmed implementation of the fault model is shown in Fig.5.2. The named variables have the following definitions:

NFSTAT - an integer variable holding the state condition of the fault under present consideration.

XRAN - a real variable holding the random number generated by FPMCRV. Also this is the initial seed to start the sequence of random numbers.

BETA, R, S - real variables holding the defined probabilities of β, r and s.

5.3 EVALUATION OF COMBINATIONAL CIRCUIT INTERMITTENT FAULT DIAGNOSTIC INTERMITTENT FAULT DIAGNOSTIC TESTING SCHEDULES

5.3.1 Evaluation Parameters

Those people concerned with the economics of implementing an automatic testing system must evaluate the various schedules and choose the most cost-effective approach. Various parameters affect the cost of diagnosing faults:

(i) Capital cost of the automatic testing system.

(ii) Capital cost of the schedule generation programs.

(iii) Cost of computer time for schedule construction to gain a required level of diagnostic resolution

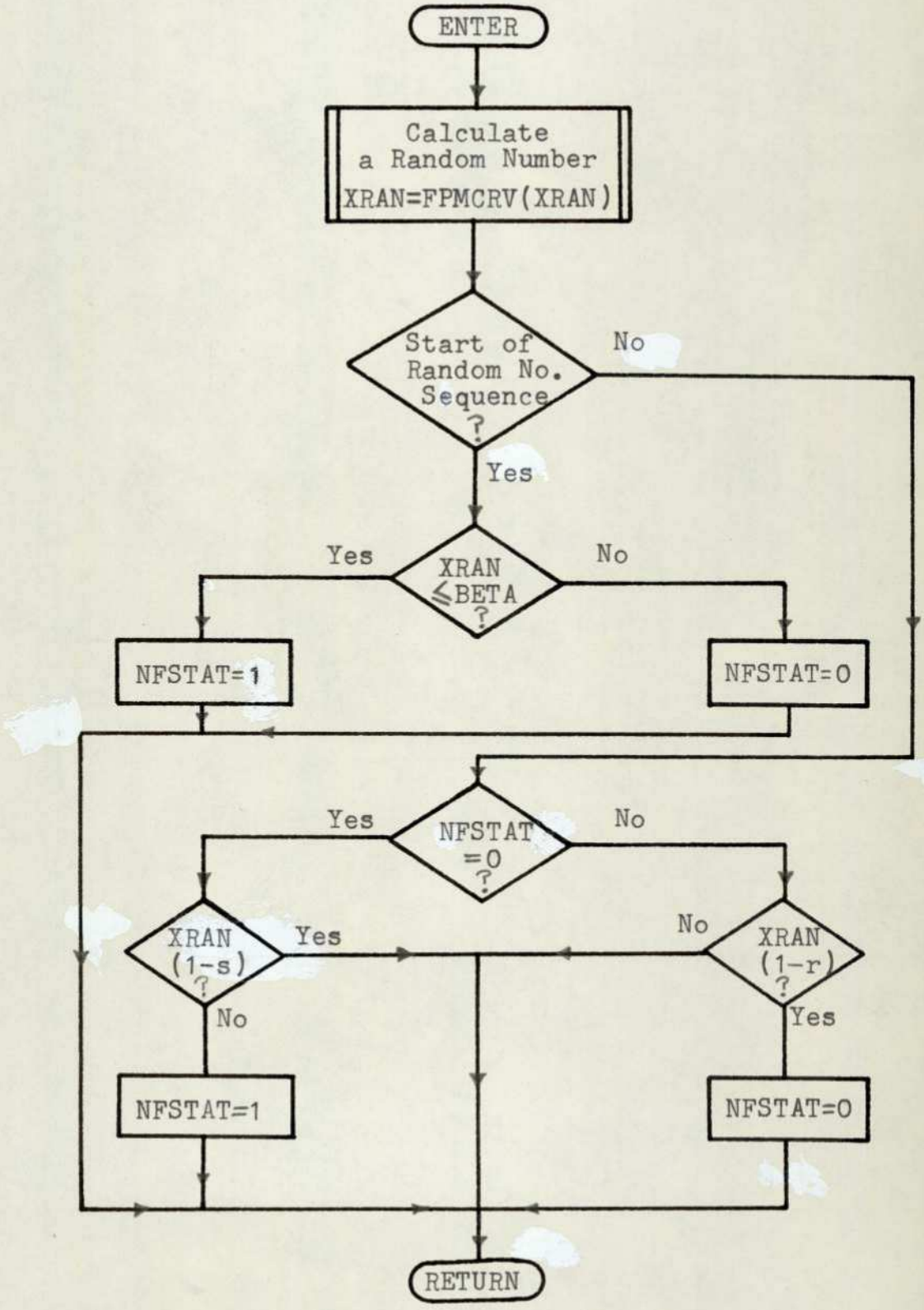


FIG. 5.2. FLOW DIAGRAM OF FAULT MODEL IMPLEMENTATION

(iv) The total number of tests in the schedule (this affects the storage requirements of the automatic tester).

(v) The average test length (time) required by a given type of schedule to diagnose the faults. (For intermittent faults this is related to the nature of the intermittency itself.)

Some of these costing parameters have been considered previously in Sections 3 and 4, and were concerned with program run-times, fault diagnostic resolution and total number of tests in the schedules. Given that levels of capital investment, schedule construction costs and fault resolution have been defined, it still remains to evaluate the costs (in terms of time) of diagnosing intermittent faults for the different types of available schedule.

Direct comparisons of the operational capabilities, such as the average time required to diagnose a fault, of each type of schedule can be made if each is evaluated under similar intermittent fault conditions. Variations in capabilities can be examined by altering the intermittency (different r and s values).

If it is assumed that all faults are equally probable the capabilities of a schedule can be evaluated for given types of intermittency, (at given values of r and s), by finding the mean time to diagnosis ($MTD_i^{r,s}$) for each fault (i) at each setting of r and s, and the calculating the mean time over all faults ($MTDF_{r,s}$) for each r and s. So, for a given setting of r and s

$$MTDF_{r,s} = \frac{\sum_{i=1}^{n} MTD_i^{r,s}}{n}$$

for the n circuit faults. Each $MTD_i^{r,s}$ is found by generating m different sequences of intermittent fault occurrence, for the same values of r and s, and calculating the mean of the time to diagnosis ($TD_i^{r,s}$). So,

$$MTD_i^{r,s} = \frac{\sum_{j=1}^{m} TD_{i,j}^{r,s}}{m}.$$

This gives

$$MTDF_{r,s} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} TD_{i,j}^{r,s}}{n \times m}.$$

The value of m, the sample size from which the mean is calculated, has to be large enough to give statistically significant results. For example, if m is made too small the error in $MTD_i^{r,s}$ can be large. Error calculations are shown later when the results are presented.

$MTDF_{r,s}$ can be plotted over a range of r and s values. If this is repeated for a different schedule, direct comparisons can be made from such graphs. In practice the m different intermittent sequences can be obtained using m different seed numbers in the random number generator of the fault model simulation.

The parameter descriptions in this sub-section apply to both combinational and sequential circuits.

5.3.2 A Procedure For D.F.P. Schedule Evaluation

The techniques of interpreting the diagnostic data produced by a solid fault schedule in the presence of intermittent faults using d.f.p.'s have been described in Section 3.3. Because of the very nature of such faults the user of the d.f.p. technique can never be completely certain if, at any given time, maximum resolution has already been reached. This is a result of loss of distinguishability under intermittent conditions. Only when a unique diagnosis is found (if at all possible) has the process guaranteed the correct result. In other words the user of the schedule may have actually reached maximum diagnostic resolution for a particular indistinguishable fault condition but can only be absolutely certain if he goes on applying the schedule for (virtually) infinite time. In practice, after a period of no further increase in resolution the diagnostic result is accepted as being of maximum resolution.

Under the experimental conditions of this project, faults are definable and the level at which they are resolved can be calculated (see Section 3.3). When the d.f.p. intersection process reaches this level the fault is maximally resolved and diagnostic effort can be terminated. For a particular fault (i) the time, or number of Test/Sample cycles, to reach this point is TD_i. Although this does not necessarily reflect the time taken under real working conditions, as long as each schedule being evaluated is treated in a similar manner,

direct comparisons of operational capabilities are valid. To aid in the evaluation process, data on the maximum resolution of each fault is used.

The variables under which the evaluation takes place are:

(i) the n circuit faults.

(ii) the m different intermittent sequences to allow mean values to be calculated.

(iii) the fault model probabilities r and s.

Another possible variable would be β, the inital fault state probability, but it was decided to define this as a constant because of the large number of variables already requiring to be evaluated. Since β is a constant for all cases the effects on the evaluation results will be minimal.

The overall process of schedule evaluation requires simulation of the circuit and the automatic testing system:

(i) Define:

(a) the circuit under test.

(b) the n fault set.

(c) the test schedule and d.f.p. sets.

(d) the levels of resolution associated with each fault.

(e) the variables r, s and m and the constant β.

(f) the m different seed values for the random number generator.

(ii) Select the i^{th} circuit fault, i=1,2,.....,n.

(iii) Select values of r and s, $0 < r < 1$ and $0 < s < 1$.

(iv) Select the j^{th} random number seed, $j=1,2,\ldots,m$.

(v) $k=1$, the Test/Sample cycle counter.

(vi) $\ell=1$, the number of the first schedule test.

(vii) Access the fault model routine and calculate the fault state.

(viii) Simulate the circuit in the presence of test T_ℓ with fault # i in the defined state.

(ix) Use the resulting circuit output response and the schedule data to determine the next step in the process.

(x) If the schedule indicates a diagnostic result go to (xi). Otherwise, the schedule defines a new test number, ℓ. Increment the cycle counter, $k=k+1$, and repeat from (vii).

(xi) Intersect the d.f.p. of the new diagnosis with previous diagnostic data (unless this is the first result). If this does not equal maximum resolution for fault # i set $\ell=1$, $k=k+1$, and repeat from (vii). Otherwise, the time to reach maximum resolution is k, $TD_{i,j}^{r,s} = k$.

(xii) Repeat from (iv) until all m seeds have been used and all $TD_{i,j}^{r,s}$ have been calculated.

(xiii) Repeat from (iii) for the other defined values of r and s to calculate all other $TD_{i,j}^{r,s}$ for the i^{th} fault.

(xiv) Repeat from (ii) for each other circuit fault.

(xv) For each defined values of r and s the mean time to diagnosis over all faults can be calculated:

$$MTDF_{r,s} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} TD_{i,j}^{r,s}}{n \times m} .$$

5.3.3 A Procedure For Intermittent Fault Schedule Evaluation

A schedule constructed specifically for diagnosing intermittent faults, such as the one described in Section 3.4, requires less supervision from the automatic testing system than the d.f.p. type. The loop structure of the schedule acts as a waiting area when effective diagnosis ceases during fault absent periods. Some intermittent faults are not uniquely diagnosable and will cause the schedule to loop in a waiting area indefinately once the maximum resolution has been obtained.

Under the experimental conditions of this project data can be supplied to the evaluation procedure to allow the testing to terminate when a defined fault is maximally resolved. This data consists of the loop-start numbers at which maximum resolution is reached for the indistinguishable faults (see Section 4.4).

The same evaluation variables used for the d.f.p. schedule, sub-section 5.3.2, are used in this procedure to permit direct comparisons of operational capabilities.

The process of evaluation is very similar to that described for the d.f.p. schedule in sub-section 5.3.2. Only steps (x) and (xi) are different. The modified parts of the procedure are as follows:

(x) If the schedule indicates a diagnostic result or if test ℓ is the loop-start test at which fault i is maximally resolved, go to (ix). Otherwise, the schedule defines a new test number ℓ. Increment the

cycle counter, k=k+1, and repeat from (vii).

(xi) The time to reach maximum resolution is k, $TD^{r,s}_{i,j} = k$.

In step (i)(c) only the schedule need be defined.

5.3.4 Implementation Of The Schedule Evaluation Procedures

The common features of the two procedures described in sub-sections 5.3.2 and 5.3.3 simplified the implementation problems to a point where a single program could be written.

As with previous combinational circuit programs the simulation is most efficiently performed by a table look-up technique using the circuit Fault-Matrix. A flow diagram illustrating the implementation of the combined evaluation procedures is shown in Fig.5.3. Some of the arrays and variables have the same names and uses as described for previous programs. The arrays and variables are:

NSCHED - a two-dimensional array for storing the diagnostic testing schedule.

NTYPE - an integer variable to indicate the type of schedule, 0 for d.f.p., 1 for intermittent fault schedule.

NFMAT - a two-dimensional (pxn) array containing the circuit Fault-Matrix for the p possible input tests and the n circuit faults (or indistinguishable fault sets).

NDFP - a two-dimensional (txq) array for listing the d.f.p. (of length q) for each of the t

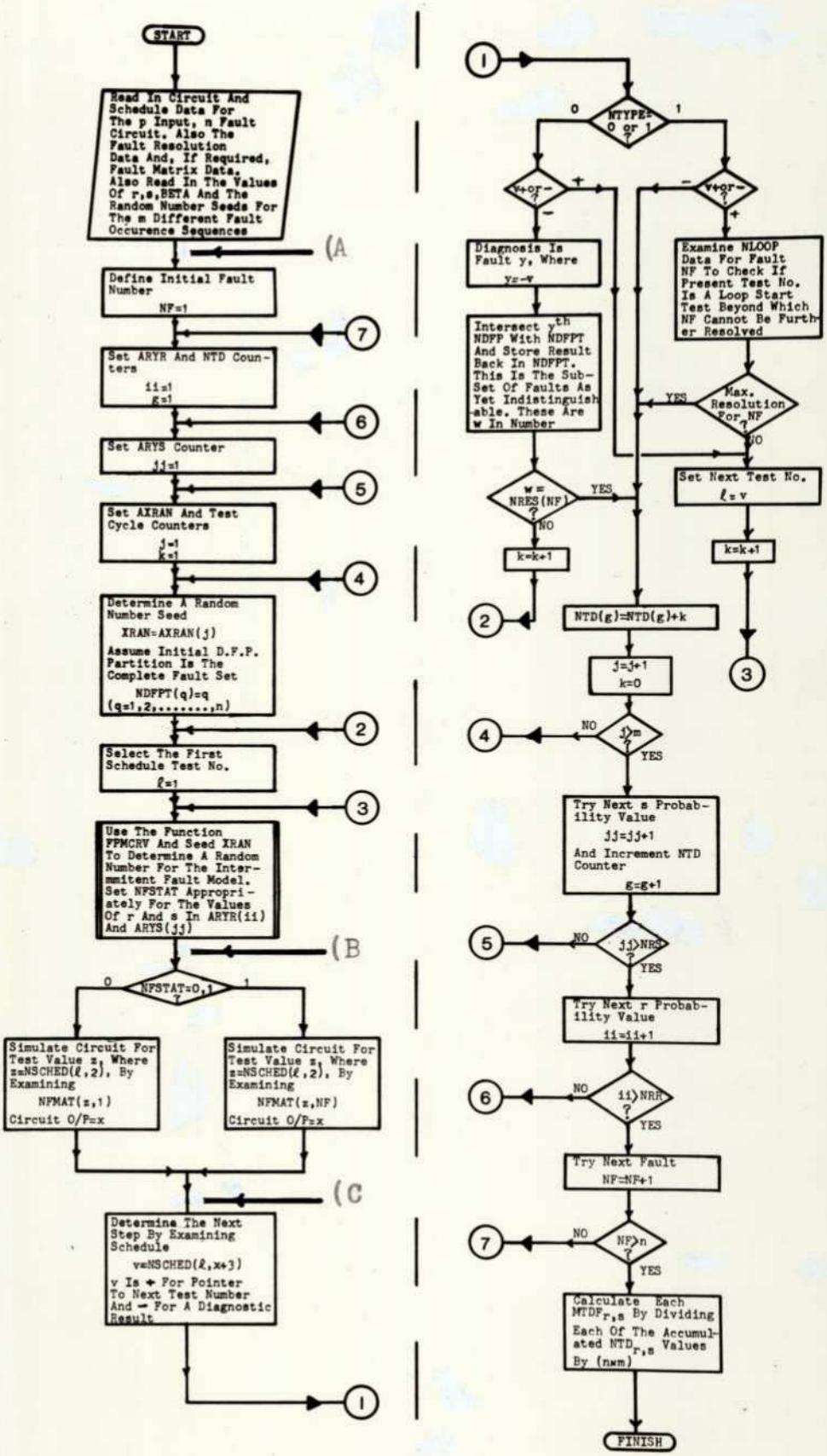


FIG. 5.3. FLOW DIAGRAM OF PROGRAM 'SCHED CHECK(COMB)'

schedule diagnostic exits (only used when NTYPE = 0)

NRES - a one-dimensional array of length n, the i^{th} element of which states the number of other faults from which the i^{th} fault is indistinguishable.

NLOOP - a two-dimensional (nxq) array listing the (q) numbers of the loop-start tests at which the i^{th} fault, i=1,2,.....,n, reaches maximum resolution.
(This is used when NTYPE = 1 only.)

NF - an integer variable containing the fault number under present consideration.

NRR, NRS - integer variables holding the number of r and s values to be considered.

ARYR, ARYS - one-dimensional arrays of length NRR and NRS, each element containing the different r and s values to be considered.

AXRAN - a one-dimensional array of length m, the j^{th} element, j=1,2,.....,m, containing the j^{th} random number seed.

NDFPT - a one-dimensional array for storing the d.f.p. intersection results.

NTD - a one-dimensional array of length (NRSxNRR) for storing the combined time for diagnosis for each fault, at each r and s value (each element finally contains the $MTDF_{r,s}$ values when divided by (nxm)).

XRAN, NFSTAT, R, S and BETA have the same meanings as before (see sub-section 5.2.3).

The computer program was written in FORTRAN IV and named SCHED CHECK(COMB).

5.3.5 A Circuit Example

Two different schedules were generated for the full-adder circuit considered in Section 3, one a d.f.p. schedule (3.3), the other a schedule specifically for diagnosing intermittent faults (3.4). These can be used to illustrate the evaluation results of the program SCHED CHECK (COMB).

Two problems arise in the evaluation procedure:

(i) The time to reach a diagnostic result is dependent on the type of intermittency (the values of r and s) and so, therefore, is the computer run-time required to compute this. The values of r and s have to be chosen so that the computation times are not too unreasonable.

(ii) To give statistical significance to the results generated by the program, the sample size (m) of $TD_i^{r,s}$ used in calculating $MTD_i^{r,s}$ must be large enough.

The bigger m is, the longer the computation time. These problems are overcome by limiting to some extent the range of r and s values under consideration and keeping m small enough such that the calculated errors still give statistically significant results.

In calculating each $MTD_i^{r,s}$ from a sample of m values

of $TD_i^{r,s}$, $MTD_i^{r,s}$ is the best available estimate of the true mean. The standard deviation of these m values is:

$$s = \sqrt{\frac{1}{m} \sum_{j=1}^{m} \left(TD_{i,j}^{r,s} - MTD_i^{r,s} \right)^2} .$$

It follows (95) that the best estimate afforded by the sample of the true standard deviation is:

$$\sigma_i = \sqrt{\frac{1}{m-1} \sum_{j=1}^{m} \left(TD_{i,j}^{r,s} - MTD_i^{r,s} \right)^2}$$

Now, the distribution of $MTD_i^{r,s}$ for all possible samples of size m has a variance:

$$\frac{\sigma_i^2}{m}$$

or standard deviation:

$$\frac{\sigma_i}{\sqrt{m}} .$$

This is the r.m.s. error to be feared in accepting $MTD_i^{r,s}$ as the estimate of the true mean time to diagnosis for fault i under r and s intermittent conditions.

Assuming all faults occur equally probably, $MTDF_{r,s}$ is calculated by:

$$MTDF_{r,s} = \frac{\sum_{i=1}^{n} MTD_i^{r,s}}{n} .$$

The error in $MTDF_{r,s}$ can be found from the error in each $MTD_i^{r,s}$ from which it is calculated. It can be shown (95) that for a quantity u, which is a function of a finite number of independant variables,

$$u = f(x,y,z,.....),$$

the error in u is given by:

$$\sigma_u^2 = \left(\frac{\partial f}{\partial x}\right)^2 \alpha_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 \sigma_y^2 + ...$$

where α_x, α_y, α_z,....., correspond to the errors in x,y, z,..... . If u corresponds to $MTDF_{r,s}$ and x,y,z,....., to the n fault conditions, then the r.m.s. error in $MTDF_{r,s}$ is:

$$\sigma_{r,s} = \sqrt{\frac{1}{n^2}\left\{\frac{\sigma_1^2}{m} + \frac{\sigma_2^2}{m} + \frac{\sigma_3^2}{m} + ... \frac{\sigma_n^2}{m}\right\}}$$

or

$$\sigma_{r,s} = \frac{1}{n}\sqrt{\sum_{i=1}^{n} \frac{\sigma_i^2}{m}}.$$

These error calculations were included in the computer-program implementation of the evaluation procedure.

Values of r and s were selected between 0.05 and 0.9 and the value of β set at 0.5. The sample size was set at m = 100. In fact m could well have been set much lower since the errors were minimal in most cases. The results are shown in Fig.5.4(a),(b),(c),(d) and (e). Each graph shows $MTDF_{r,s}$ plotted against s for fixed values of r. The units of MTDF are in numbers of tests. The units of r and s are those of probability between 0 and 1.0. The two lines numbered #1 and #2 in each graph correspond to the d.f.p. and the intermittent fault schedule, respectively. As would be expected the MTDF increases as the sparsity of fault occurrence increases (as r and s decrease in value).

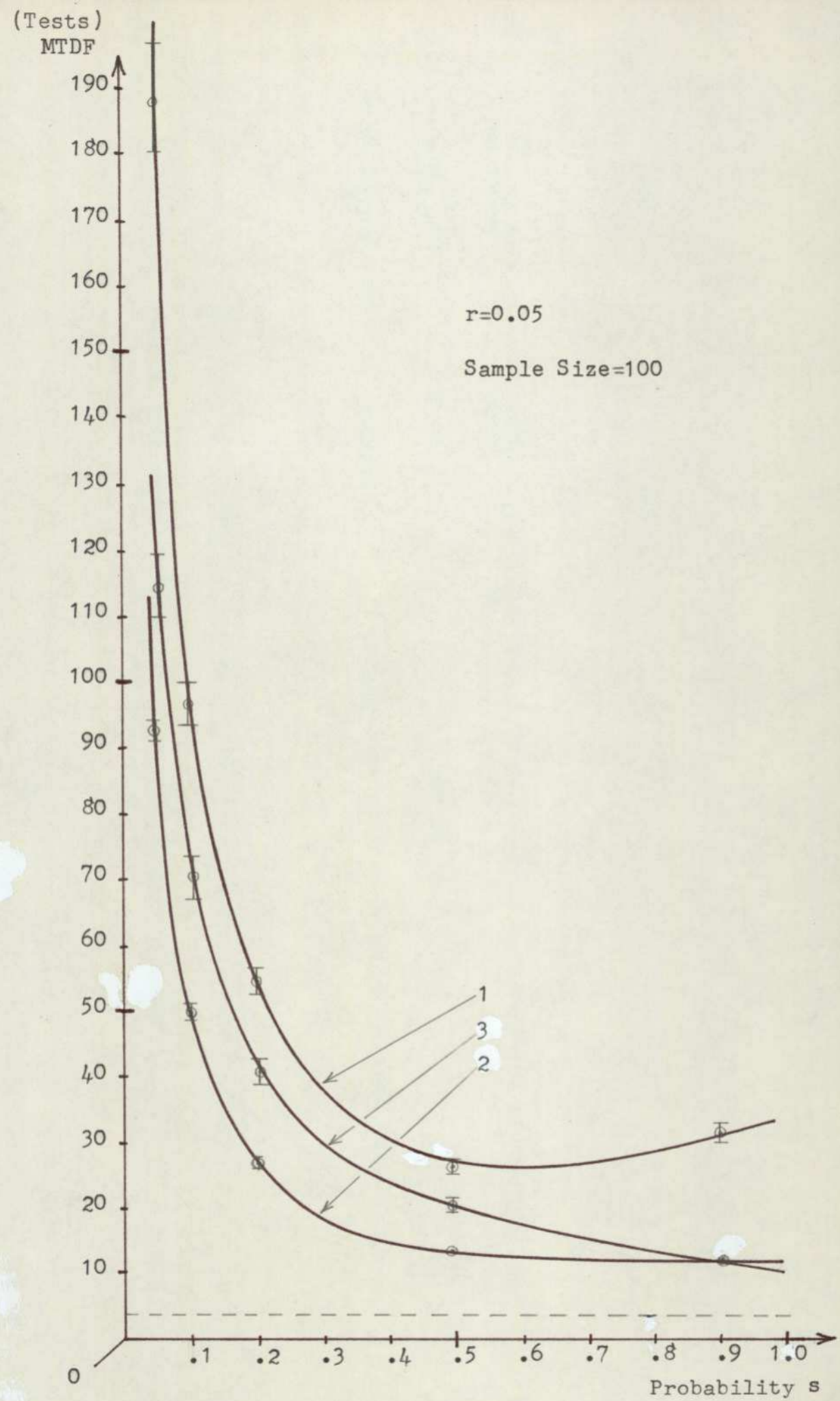


FIG. 5.4.(a). MTDF Plotted Against s at r=0.05

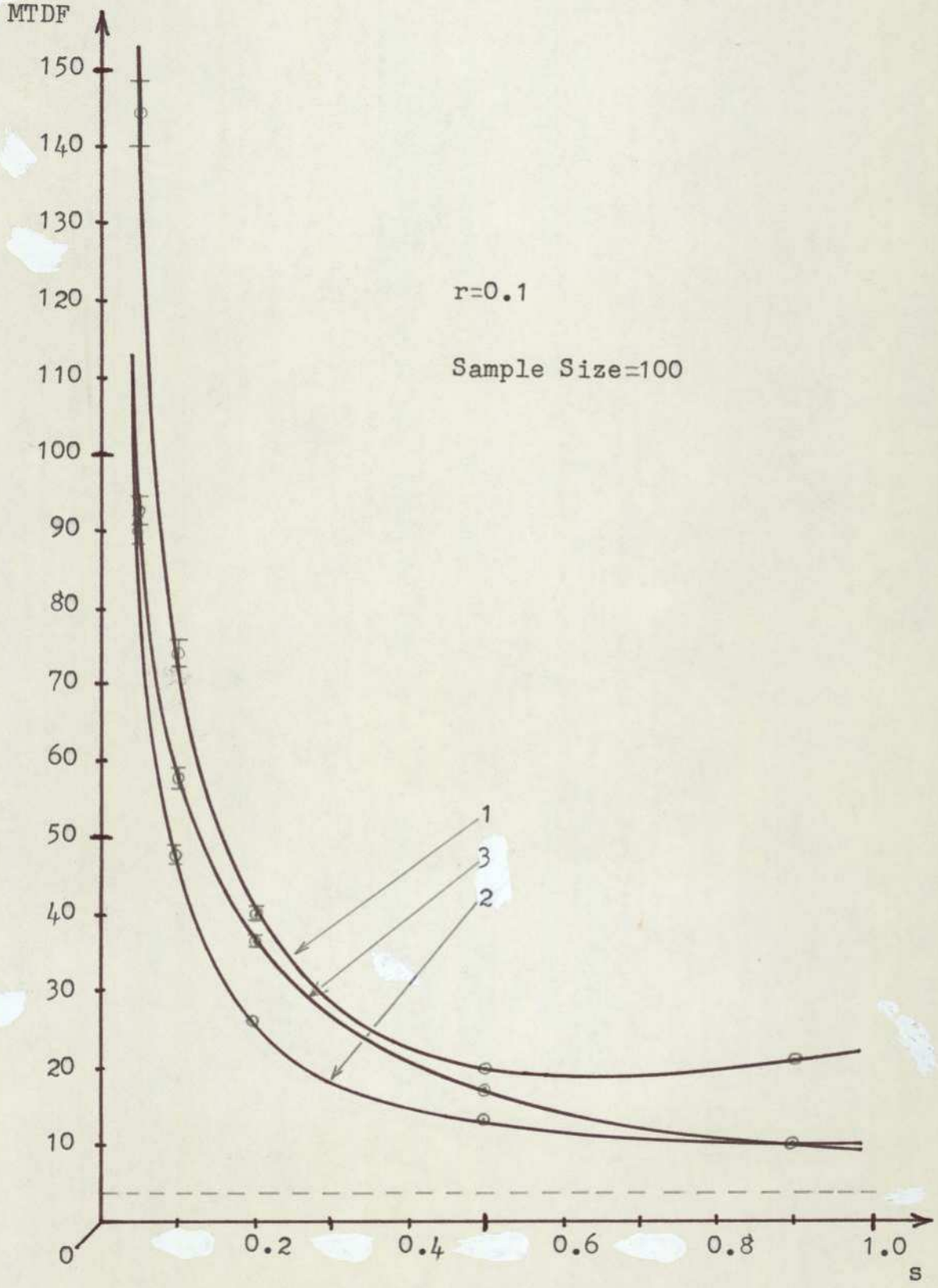


FIG. 5.4(b). MTDF Plotted Against s at r=0.1

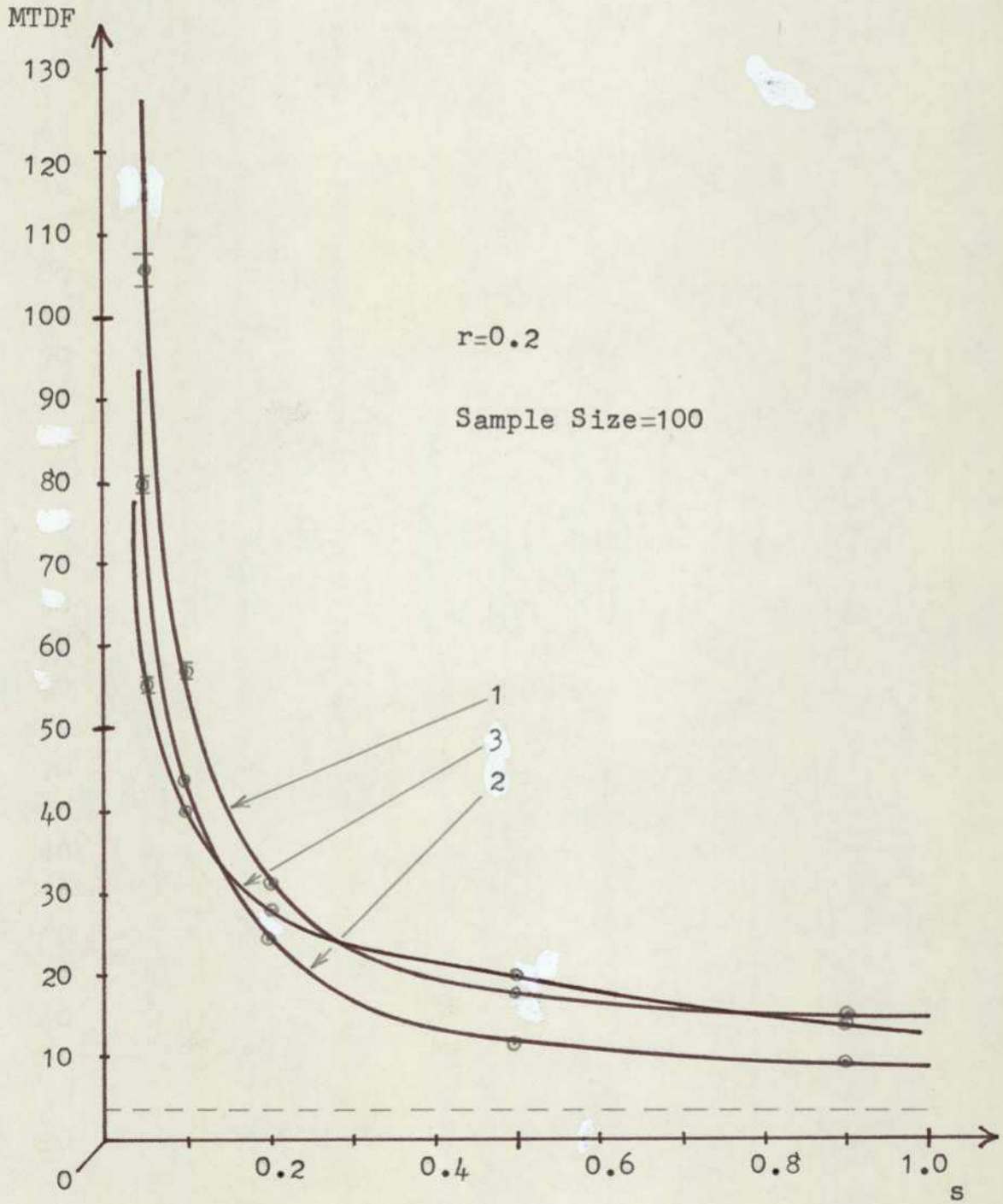


FIG. 5.4.(c). MTDF Plotted Against s at r=0.2

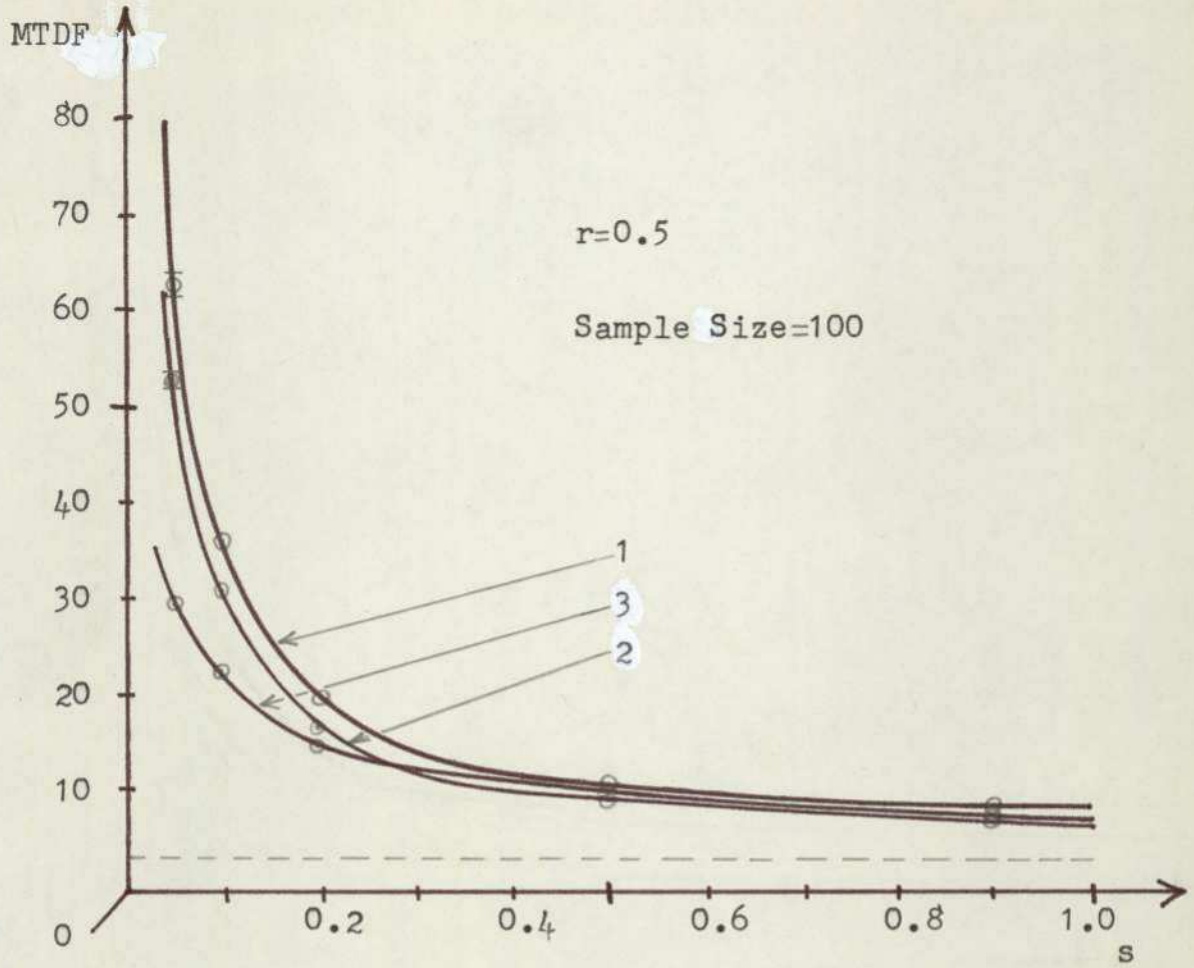


FIG. 5.4.(d). MTDF Plotted Against s at r=0.5

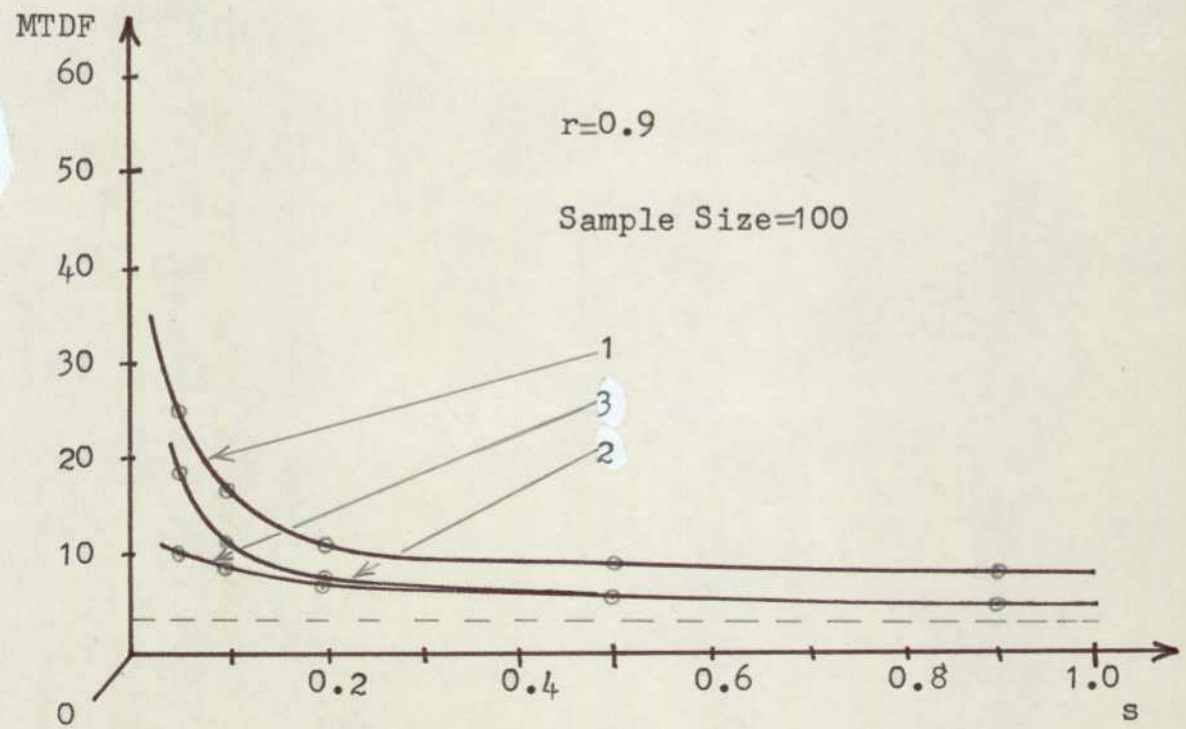


FIG. 5.4.(e). MTDF Plotted Against s at r=0.9

At low values of r the intermittent schedule takes approximately half the time on average to diagnose faults. The error bars, corresponding to $\pm \sigma_{r,s}$ calculated as shown previously, indicate excellent statistical significance of the results. Where errors are not shown, they are within the graph point size.

The broken line on each graph shows the mean time to diagnosis ($\overline{TD}$) obtained for solid faults using the solid fault schedule (Section 3.2). As would be expected the values of $MTDF_{r,s}$ approach this level as the fault intermittency tends to become more like a solid fault (as r and s values approach 1·0).

One method of comparing the operational capabilities of each schedule against the other is to examine the ratio of the calculated $MTDF_{r,s}$. This is shown in Fig.5.5 with the ratio MTDF #1/MTDF #2 plotted against s for each value of r. An obvious conclusion from this comparison is that the intermittent fault schedule makes much better use of the time the fault is present than the d.f.p. schedule. This is an indication of the success of the schedule construction philosophy formulated in Section 3.4.

Using the same circuit example a further testing method can be considered. Bennetts (60) has suggested that the normal approach to the problem of intermittent fault diagnosis is to re-cycle the diagnostic test set through each test in turn until the fault recurs and is successfully diagnosed. This method works by testing over a long enough period so that the fault has occurred often enough

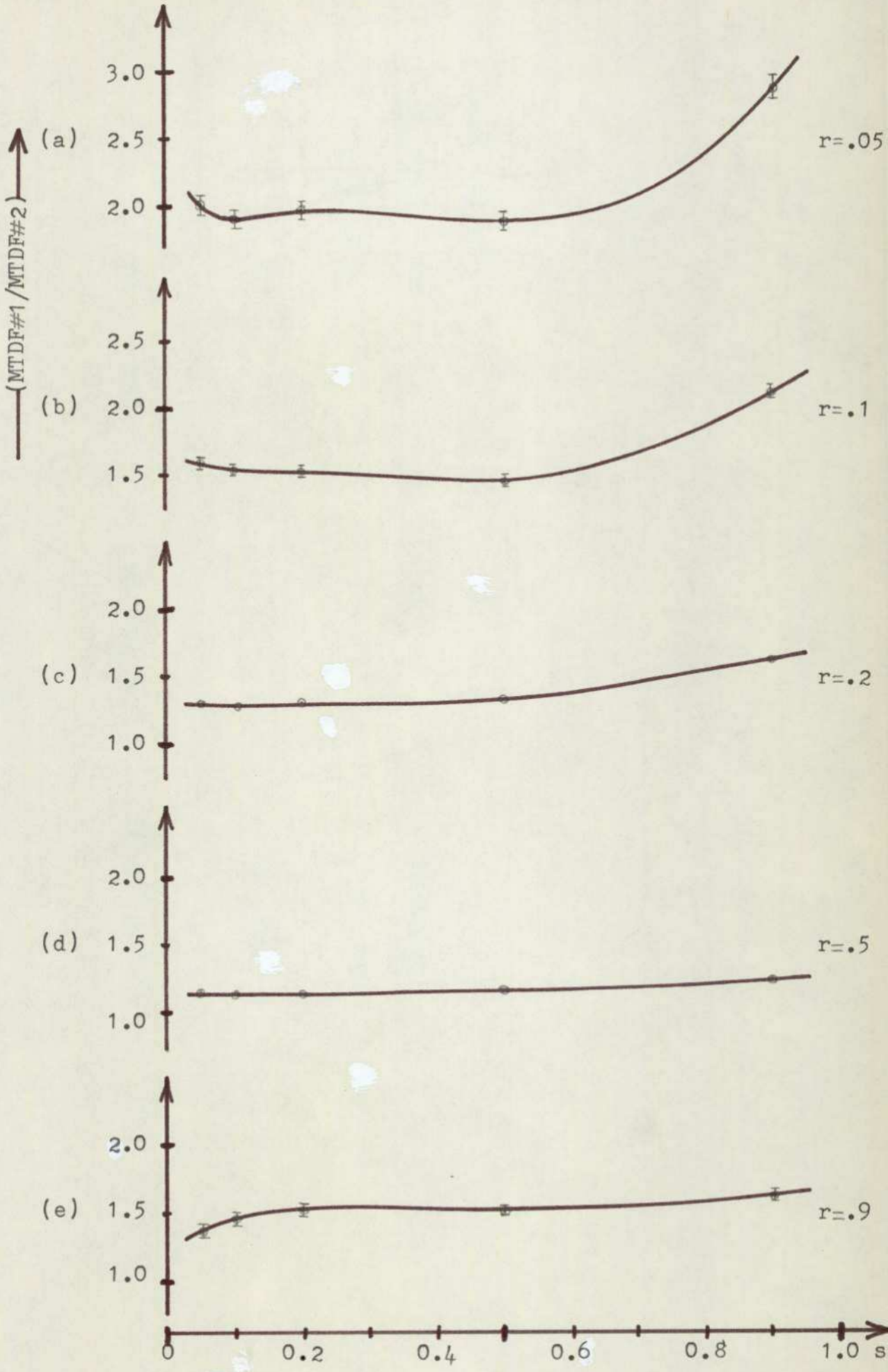


FIG. 5.5. RATIO (MTDF#1/MTDF#2) PLOTTED AGAINST s

for each of the relevant tests (of the particular fault) to have been covered at least once by the fault-present condition. This produces a combined set of output responses corresponding to the uniquely resolved column in the Fault-Matrix. This was implemented in SCHED CHECK(COMB) by supplying a schedule consisting of a continuous loop of tests (0 to 7 and then back to 0, etc.). The program was modifed to check when the fault had been present at least once during the application of each of the relevant tests. When this point was reached a correct diagnosis was obtained. Normal statistical checks were performed by using a sample size m = 100, as before. The results are shown as line #3 in the graphs of Fig.5.4.

At higher values of r and s this is obviously a more effective technique. At lower values of r and s, that is with sparse fault occurrences, the schedule designed specifically for diagnosing intermittent faults makes much better use of the fault present condition and is more effective.

Although the re-cycling of tests produces reasonable results and can be used as a standard for comparison, there are certain problems with its implementation as a part of an automatic testing system not found to such an extent with the other types of schedule:

(i) The full Fault-Matrix would have to be stored for the analysis to be successfully performed. This would mean large storage requirements even for quite small circuits.

(ii) In working towards a correct diagnosis (x) an incorrect result (y) could be obtained (but the user would not know this). This is caused by all the relevant tests of fault #y being the same as some of the relevant tests of fault #x. An example is fault #14 of the full-adder circuit. It is possible that fault #13 could be incorrectly diagnosed as the fault since the relevant tests for #13 are 1 and 2, while those for #14 are 0,1,2 and 4. This is the partial indistinguishability discussed in Section 3.

In forming the results of the graphs in Fig.5.4 for schedules numbered #1, #2 and #3, over 7 million individual test input simulations were performed. This required one hour and forty minutes of computer time, working at the rate of about 1200 simulations per second, on the ICL 1905 computer.

5.4 EVALUATION OF SEQUENTIAL CIRCUIT INTERMITTENT FAULT DIAGNOSTIC TESTING SCHEDULES

5.4.1 Evaluation Parameters

The parameters for evaluating different types of testing schedule discussed in sub-section 5.3.1 apply to the case of sequential circuits. Evaluation of the mean time to diagnosis over all faults (MTDF) permits comparisons of operating capabilities to be made as before.

5.4.2 Procedures For Schedule Evaluation

The two procedures for schedule evaluation described in sub-sections 5.3.2 and 5.3.3 are, with some minor

changes, satisfactory for the case of sequential circuits. The changes are associated with the fact that the intermittent fault model permits two changes in fault state for each Test/Sample cycle. Also, some of the tests in the schedule are p.r.s. and require momentary application of appropriate resets to the circuit.

The modifications occur at steps (vii) and (viii):

(vii) (a) Access the fault model routine and calculate the next fault state.

(b) Simulate any change in fault state in the circuit model.

(viii) (a) Simulate the circuit in the presence of test T_ℓ with fault #i in the defined state.

(b) If test T_ℓ is a p.r.s. momentarily apply reset states to circuit.

(c) Access the fault model routine and calculate the next fault state.

(d) Simulate any change in the fault state.

5.4.3 Implementation Of The Schedule Evaluation Procedures

As with the evaluation procedures for combinational circuit schedules, the d.f.p. and intermittent fault schedules for sequential circuits can be evaluated using a single computer program. The main differences in the implementation come in the use of the SIMULATION subroutine for calculating the circuit reactions to the input test and fault-state transition stimulii. The simple Fault-Matrix table look-up approach is not possible for sequential circuits.

The program was written in FORTRAN IV and named SCHED CHECK(SEQ). The basic flow diagram is the same as for SCHED CHECK(COMB) in Fig.5.3 (sub-section 5.3.4). Differences in the flow diagram occur at point {A}, where an addition is made, and between points {B} and {C}. These are shown in Fig.5.6. The program uses the same array and variable names as in SCHED CHECK(COMB).

5.4.4 A Circuit Example

Two different schedules were generated for the Seshu and Freeman circuit considered in Section 4, one a d.f.p. schedule (4.3), the other a schedule specifically for diagnosing intermittent faults (4.4). These can be used to illustrate the evaluation results of the program SCHED CHECK(SEQ).

Values of r and s used in the evaluation have to be suitable to give reasonable computation times. The sample (m) of calculated values of $TD_i^{s,r}$ used in determining each $MTD_i^{r,s}$ has to be large enough to give statistically significant results, but must be small enough to give reasonable computation times. The use of the much slower SIMULATION routine for sequential circuits only aggravates these run-time problems. For these reasons ranges of r and s were kept at the same levels as before, while m was reduced drastically to m = 5. The same error calculations discussed in sub-section 5.3.5 apply in this case.

The results are shown in Fig.5.7(a),(b),(c),(d),(e) and (f). Each graph shows $MTDF_{r,s}$ plotted against s for

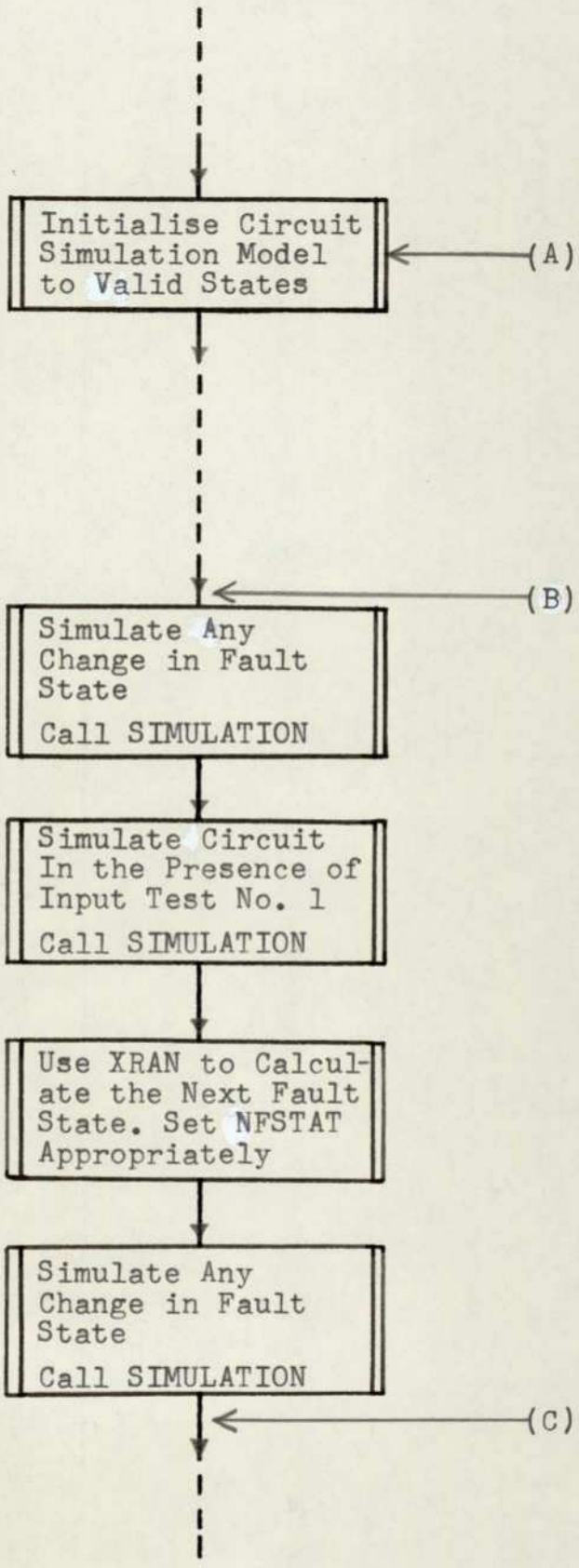


FIG. 5.6. MODIFICATIONS TO FLOW DIAGRAM OF FIG. 5.3.

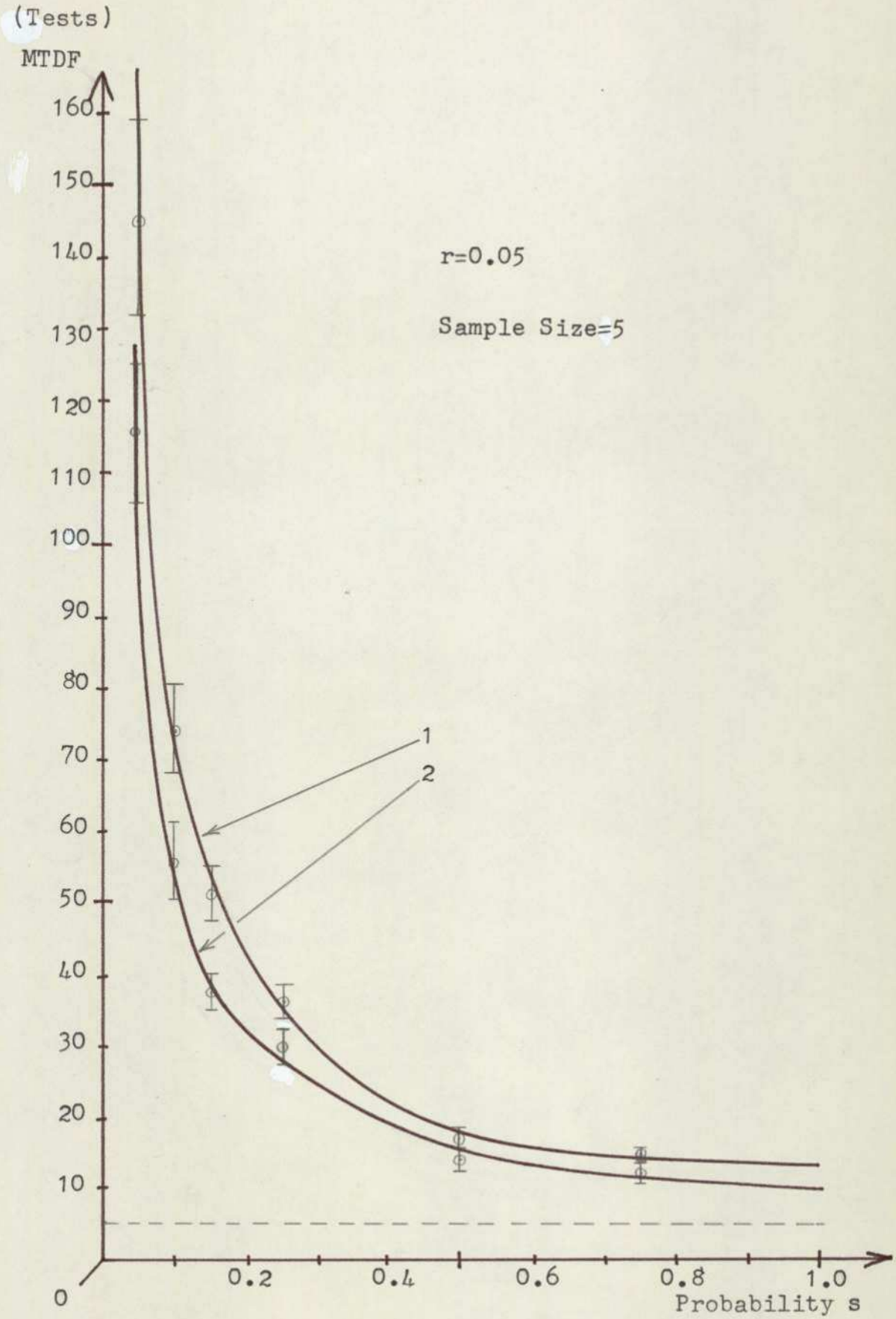


FIG. 5.7.(a). MTDF Plotted Against s at r=0.05

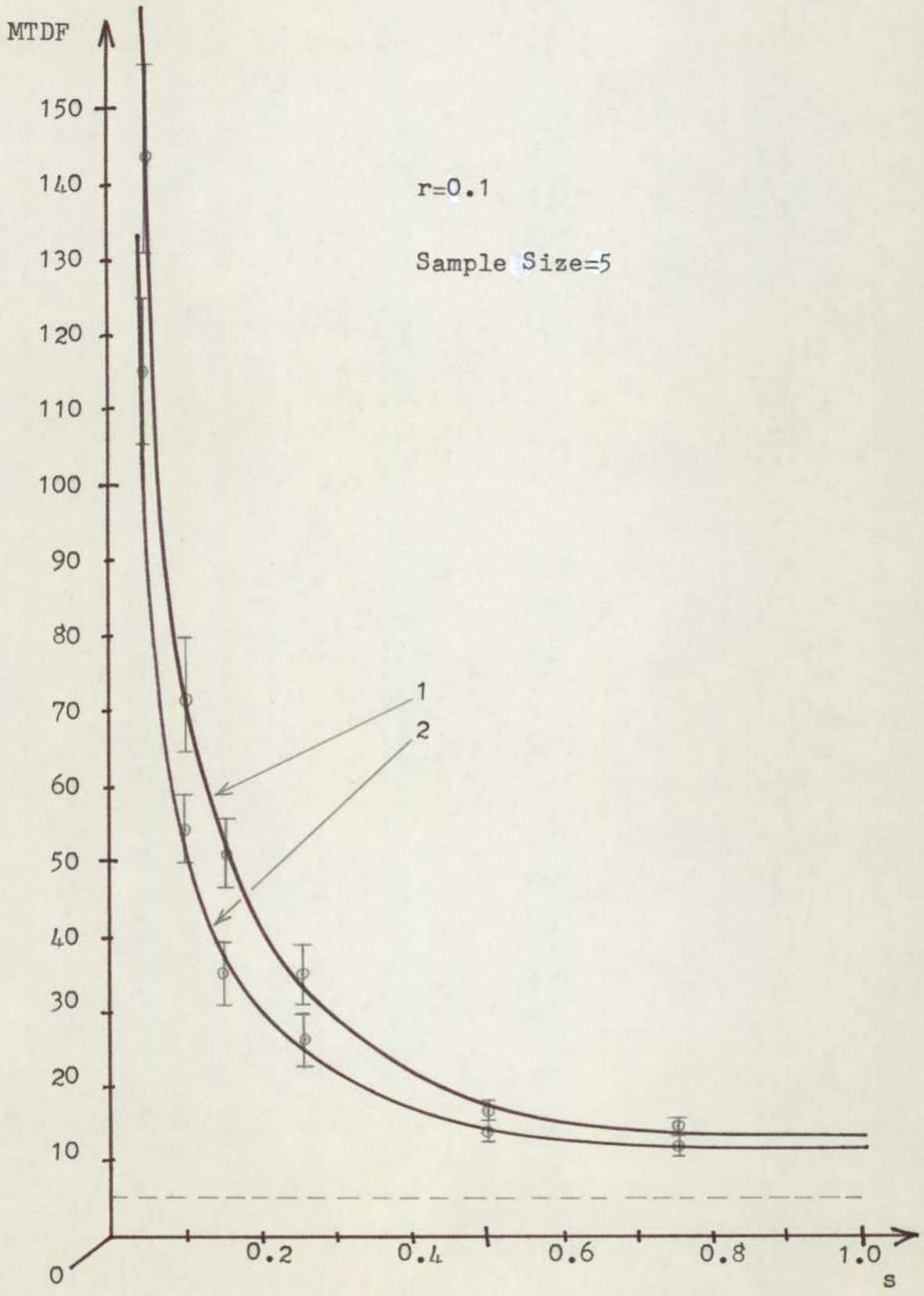


FIG. 5.7.(b). MTDF Plotted Against s at r=0.1

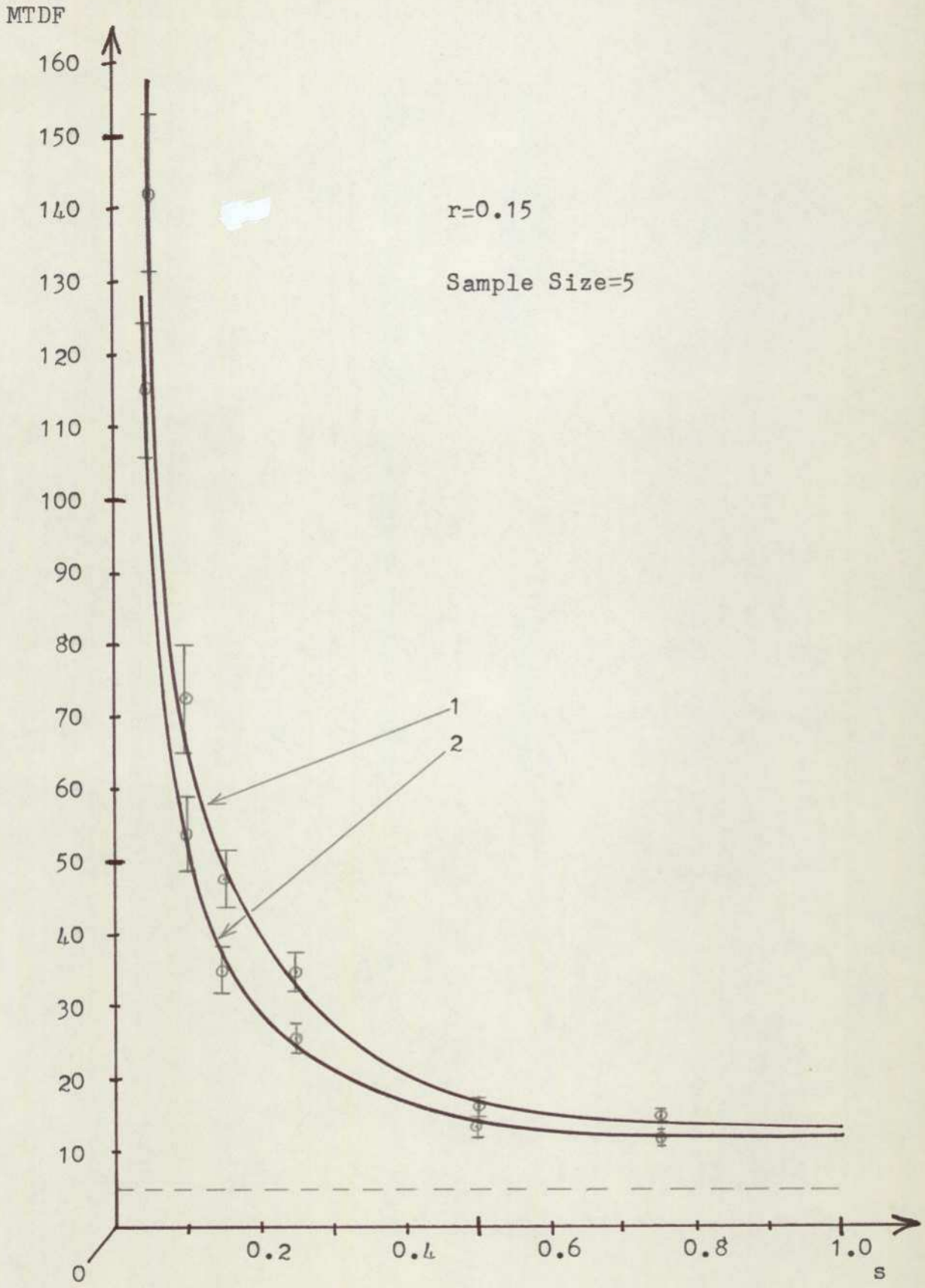


FIG. 5.7.(c). MTDF Plotted Against s at r=0.15

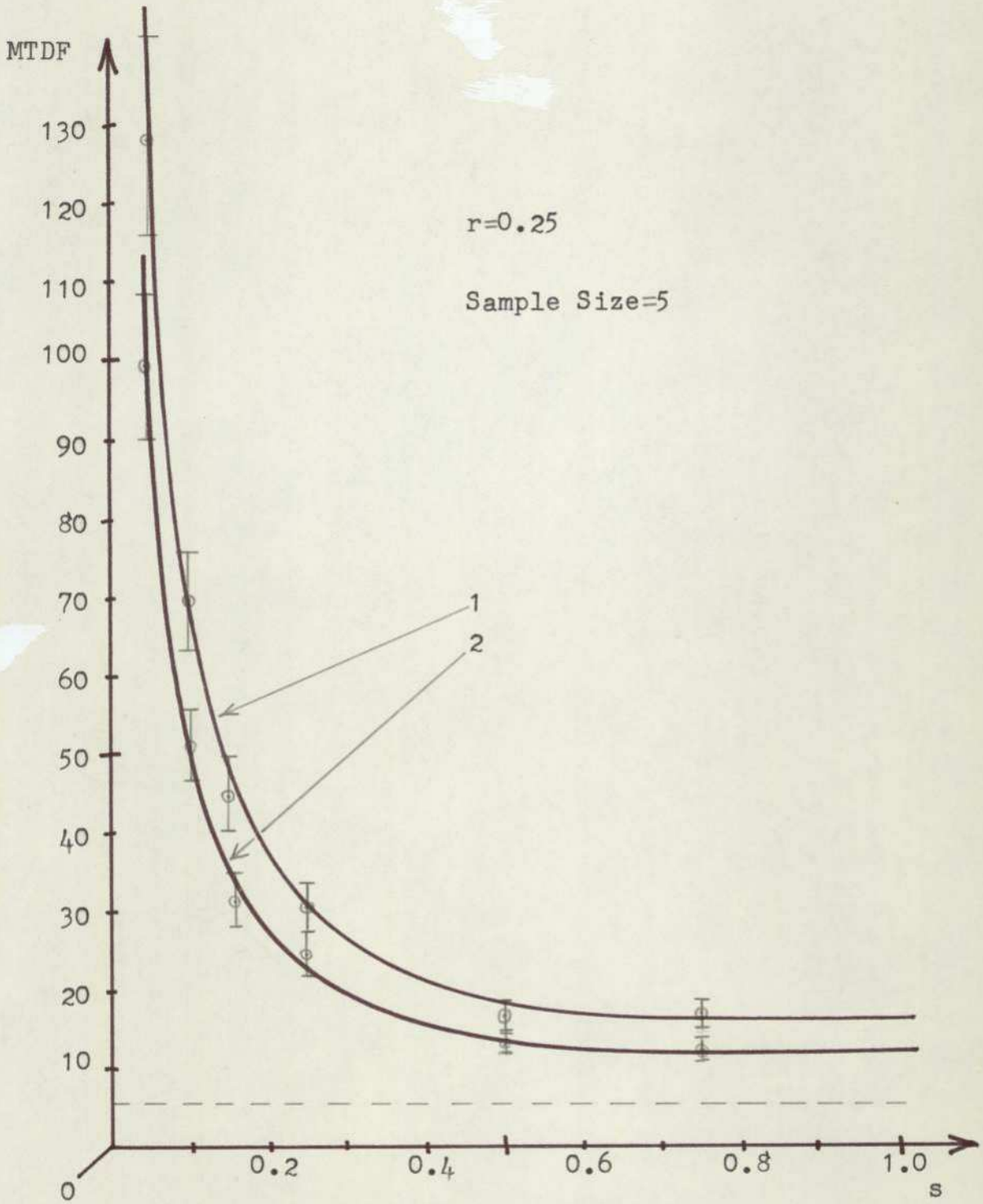


FIG. 5.7.(d). MTDF Plotted Against s at r=0.25

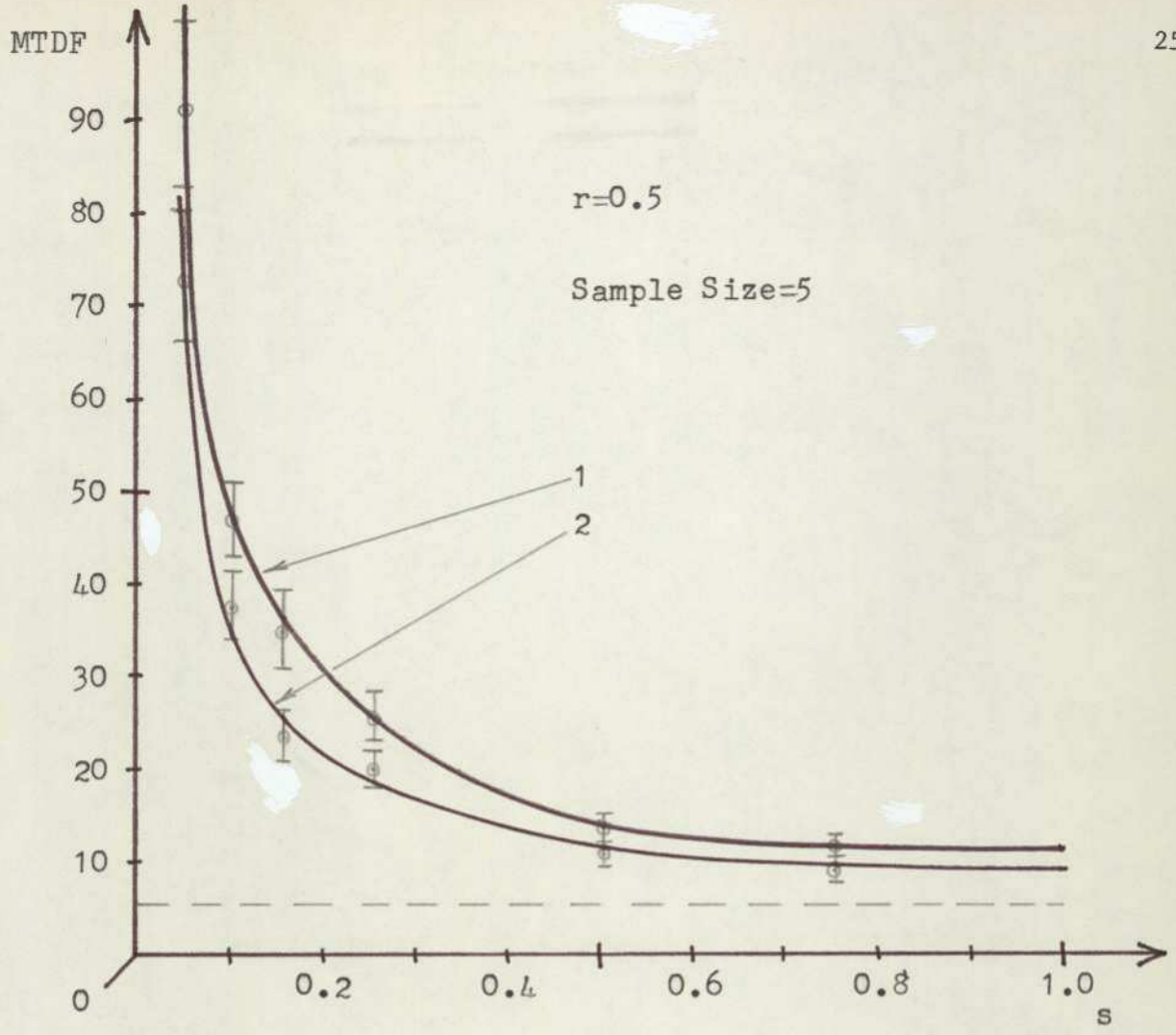


FIG. 5.7.(e). MTDF Plotted Against s at r=0.5

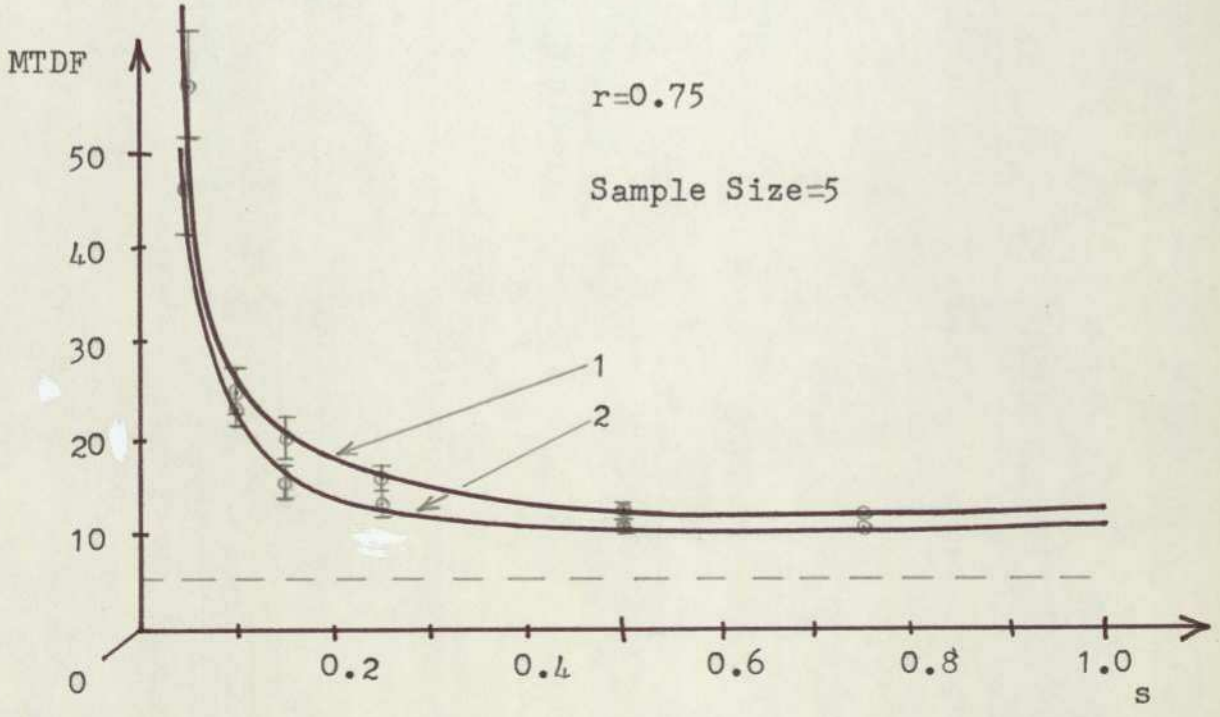


FIG. 5.7.(f). MTDF Plotted Against s at r=0.75

fixed values of r. The units of MTDF are in numbers-of-tests. The units of r and s are those of probability between 0 and 1.0. The two lines numbered #1 and #2 in each graph correspond to the d.f.p. and the intermittent fault schedule respectively. As would be expected the MTDF increases as the sparsity of fault occurrence increases.

The error bars represent the r.m.s. error in $MTDF_{r,s}$, $\pm \sigma_{r,s}$, calculated during the evaluation procedure and indicate that the results are statistically significant, although much less so than in the combinational circuit case (sub-section 5.3.5) where the sample size was m = 100.

The broken line on each graph shows the mean time for diagnosis obtained for solid faults using the solid fault schedule (Section 4.2). As would be expected the values of $MTDF_{r,s}$ approach this level as the fault intermittency tends to become more like a solid fault (as r and s values approach 1.0). The gaps here are larger than in the combinational circuit case (sub-section 5.3.5). This is due to the number of extra p.r.s. reset tests required to initialise the circuit to known or predictable state values at each loop start-test of the schedule.

A comparison of the operational capabilities of each schedule against the other can be made by forming the ratio of the calculated $MTDF_{r,s}$. This is shown in Fig.5.8 with the ratio MTDF # 1/MTDF # 2 plotted against s for each value of r. An obvious conclusion from this comparison is that the intermittent fault schedule makes

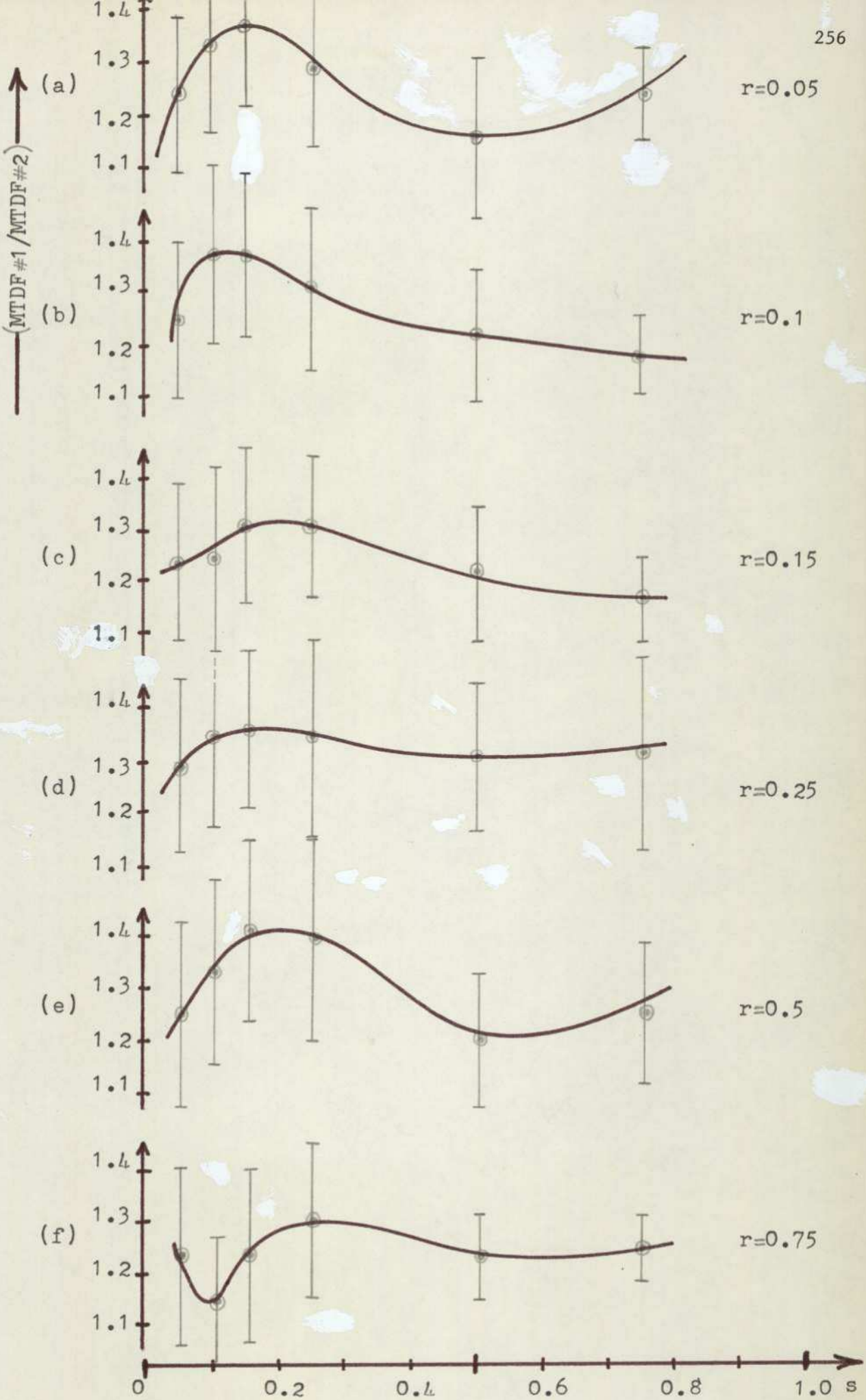


FIG. 5.8. RATIO (MTDF#1/MTDF#2) PLOTTED AGAINST s

much better use of the time the fault is present than the d.f.p. schedule. This is an indication of the success of the schedule construction philosophy formulated in Sections 3 and 4. However, the ratio of results is lower than that obtained for combinational circuits. This again is due to the number of extra p.r.s. reset tests in the intermittent fault schedule, which are not used so extensively in the d.f.p. technique.

The present output response of a sequential circuit is a function of the present input test and previous input tests. Under intermittent fault conditions the output is also a function of previous fault state transitions. A simple technique of re-cycling through the test set until the fault has occurred often enough to be uniquely identified is therefore not possible. This means that there is no standard technique against which a comparison can be made of the results of the two schedules described previously. These results must therefore stand on their own to indicate that the most effective diagnostic testing technique for intermittent faults in sequential circuits yet found is that of the schedule designed specifically for locating such faults.

The results shown in Fig.5.7 represent a total of about 650,000 individual test simulations performed in a time of one hour and fifty minutes on the ICL 1905 computer. This is a work rate of about 100 simulations per second.

5.5 AN EXTENSION TO THE USE OF THE FAULT PROBABILITY MODEL

A diagnostic-fault-profile is a list of faults which under intermittent conditions can cause the same diagnostic result. In the sequence of tests leading up to a diagnosis the different fault members of the d.f.p. have different relevant tests (i.e. tests for which a given fault must be present to cause this diagnosis). The pattern of fault occurrences required by one fault for it to be included in the d.f.p. is therefore different to that for another. The probabilities of such patterns occurring may be different. If this is so then the probability distribution of each d.f.p. member can be calculated. From this distribution a list of the faults in their order of decreasing probability can be formed. Such a list can act as a guide to which faults are most probable and therefore which to try to repair first. This would be particularly useful for undistinguishable faults.

Given that a sequence of tests T_1, T_2, T_3, T_4, T_5 and T_6 uniquely diagnose the presence of the solid fault #x, and that T_3 and T_5 are the only relevant tests for #x, the probability of #x being correctly diagnosed under intermittent conditions can be calculated. Assume values of s,r,α and β for the intermittent fault probability model. Equation (8) of 5.2.2 gives the probability of the fault being present at T_3 as

$$ {}^{3}P_x = \alpha . {}^{\Delta=2}P_{01} + \beta . {}^{\Delta=2}P_{11} $$

${}^{\Delta}P_{01}$ and ${}^{\Delta}P_{11}$ are defined by equations (4) and (6) of 5.2.2. The value of Δ is 2 because this is the number of steps between the first and third test. The state of the fault at T_1 is defined by α and β.

The probability of fault #x also being in the fault present condition at T_5 (a requirement for diagnosis) is given by

$$^5p_x = {}^3p_x \cdot {}^{\Delta=2}P_{11}.$$

Since the fault state at T_6 is irrelevant, the fault is correctly diagnosed with a probability

$$^6p_x = {}^5p_x.$$

A fault #y, which under intermittent conditions is a member of d.f.p. x, can similarly be evaluated by identifying the required relevant tests in the sequence $T_1 \rightarrow T_5$ for it to be diagnosed as fault #x and using the $^{\Delta}P_{01}$ and $^{\Delta}P_{11}$ equations appropriately.

In a similar way the probabilities of each fault member in each d.f.p. of reaching that diagnosis can be calculated for the particular values of s,r,α and β (or α_{st} and β_{st} if stationary conditions are assumed to exist). Since fault #x must reach some diagnostic result when the test schedule is applied, including perhaps the 'no fault' results,

$$\sum_{j=1}^{n} p_x^j = 1$$

for the n diagnostic exits.

A suitable point to calculate these probabilities is during the d.f.p. construction phase. This feature was added to the program FAULTS. An example is the diagnostic result -8 for the full-adder circuit. This has a d.f.p. consisting of the faults #8,9,23 and 25. With s=0.3, r=0.7 and α=β=0.5, the probabilities associated with these four faults are 0.5, 0.15, 0.15 and 0.21 respectively. The relative probabilities (given that we now know

the fault is either #8,9,23 or 25) are 0.495, 0.1485, 0.1485 and 0.2079. Fault #8 is the most probable fault and faults #9 and 23 the least probable.

If the schedule is re-applied to the circuit and a new diagnostic result obtained, the probability distribution changes, particularly if the set of possible faults is reduced. This can be calculated by a simple iterative procedure.

Let P_i^j represent the probability of fault #i causing the diagnostic result j to be obtained, where i=1,2,.....,m for the m circuit faults and j=1,2,.....,n for the n diagnostic results. n is not always equal to m since some null schedule exits may be accessed during intermittency. Each P_i^j is calculated as shown previously, $P_x^x = {}^6p_x$, etc. Any fault #i not a member of the d.f.p. of j has $P_i^j = 0$. This data can be stored in a (nxm) matrix, where

$$\sum_{j=1}^{n} P_i^j = 1.0$$

as indicated previously.

Let the initial relative probability distribution of faults be $\{{}^oQ_i\}$, i=1,2,.....,m, where $\sum_{i=1}^{m} {}^oQ_i = 1$. If all faults are equally probable, ${}^oQ_i = 1/m$.

The probability of obtaining result j after the first pass through the schedule is

$${}^1p_j = \sum_{k=1}^{m} \left({}^oQ_k . P_k^j\right).$$

Then the relative probability of fault i after one pass through the schedule is

$$ {}^{1}Q_i = \frac{{}^{o}Q_i . P_i^j}{\sum_{k=1}^{m} \left({}^{o}Q_k . P_k^j \right)} . $$

Similarly, after w passes through the schedule, with various diagnostic results (j) being obtained,

$$ {}^{w}Q_i = \frac{{}^{(w-1)}Q_i . P_i^j}{\sum_{k=1}^{m} \left({}^{(w-1)}Q_k . P_k^j \right)} $$

is the probability of the fault being #i.

At each step in the process the ${}^{(w-1)}Q_k$, k=1,2,.....,m, probability values have to be calculated and recorded to be used in the subsequent step. $\{{}^{w}Q_k\}$ is the probability distribution of the faults. If at the w^{th} step any ${}^{w}Q_i$ becomes 1.0 (the others must then all be zero) a unique diagnosis has been reached.

The probability distribution of an indistinguishable fault set can be used to guide the maintenance engineer to the most probable site of the malfunction. If the intermittency is very sparse and full diagnosis is taking a long time, the probability data can be used to initiate repair work at the most probable locations.

The main drawbacks to this technique are related to the fact that the values of s,r,α and β are not definable at present. However, this could prove a most powerful and useful aid for overcoming the diagnostic problems associated with intermittent faults.

Because of the lack of time, simulation experiments to

investigate an implementation of this technique were not performed prior to the writing of this thesis. Such an investigation is the next area to be examined.

5.6 DISCUSSION

The first-order two-state Markov model of fault intermittency developed by Breuer (58) has proved most useful as a tool for evaluating the operational characteristics of the various types of schedule developed throughout this project. The fault model has certain limitations which have been recognized and accepted. The limitations are acceptable because both types of schedule are evaluated under exactly similar conditions, making any comparisons valid.

The cost of diagnosing faults can be divided into several areas, as described in sub-section 5.3.1, of which the parameter of average-test-length is one. The others have been considered to some extent in previous sections. The average test length (or time) to obtain diagnosis can be used for determining and comparing the operational effectiveness of the various schedules. A number of schedule evaluation procedures were developed and implemented, as described in sub-sections 5.3 and 5.4.

For combinational circuits the schedule constructed specifically for diagnosing intermittent faults proved very much more effective than the d.f.p. schedule, especially at lower levels of r and s. In terms of the economics of fault-finding the intermittent fault schedule would prove more cost-effective if the extra costs of greater schedule construction computer run-times and greater storage requirements in the automatic testing system are acceptable. This is up to the user to decide.

Re-cycling of the complete test set produced reasonable results, especially for higher values of r and s where the fault occurs frequently. Use of such a schedule has certain drawbacks

associated with excessive storage requirements in the automatic tester and in the uncertainty of the correctness of the diagnostic results. The schedule designed specifically for intermittent fault diagnosis proved more effective at the lower values of r and s corresponding to sparse fault occurrences.

For sequential circuits the results of the evaluation procedures have shown that the differences between the two schedule techniques are not as marked, although the intermittent fault schedule is obviously more effective. No standard technique, such as re-cycling the test set, could be found to give a comparison against this technique of diagnosing intermittent faults in sequential circuits.

The sample sizes used in calculating the results for both combinational and sequential circuit schedules have proved satisfactory. All results indicate good statistical significance.

The main limitation of the evaluation processes is in the fact that the real-life situation is not always modelled satisfactorily. When a given fault reached maximum diagnostic resolution testing ceased and the time to reach that point was recorded. In reality the automatic test system or the test engineer would only be certain that the intermittent fault was maximally resolved if testing went on for a much longer time. However, this is the same for all types of schedule, so the simplification in the evaluation processes is reasonable when comparisons are being made.

Large amounts of computer-run time were used during evaluation. This results from the time to reach a diagnosis being dependant on the very nature of the fault intermittency.

The intermittency itself is modelled by the random number routine in association with the fault model and therefore affects the computation time quite considerably.

The use of the fault probability model has been extended to allow the prediction of the probability distribution associated with a set of indistinguished faults in the d.f.p. process. This can tell the maintenance engineer the most probable fault to start repair work on. This is potentially a most powerful diagnostic aid, but suffers from the inability at present to define the fault probability parameters.

6. GENERAL DISCUSSION AND CONCLUSIONS

6.1 PROJECT FORMULATION

The complex problems of designing, constructing and maintaining modern digital systems have led many computer and digital system manufacturers to utilise the power of computers. This is generally termed Design-Automation. Of particular interest to manufacturers and users of digital equipment is the maintenance of system integrity. One approach to this problem is to make systems inherently more reliable through the use of duplicated system or circuit components. Another approach is to detect malfunctions and then reconfigure the system to allow system processing to continue (although perhaps at lower efficiency). A third method is to test the system off-line to locate the position of the malfunctioning component and repair or replace it.

Much work in the past was concerned with dealing with solid faults. The lack of techniques for locating intermittent faults has been noted by several authors (see Section 1.3.3). This represented the justification for proceeding with the project described in this thesis.

A general statement of intent for this project was - the investigation of techniques for deriving efficient testing schedules for the diagnosis of intermittent faults in digital circuits. This was divided into three main phases.

(a) Investigation of the effectiveness of present solid fault schedules at diagnosing intermittent faults

and the derivation of techniques for efficiently utilising such diagnostic information for faults in both combinational and sequential circuits.

(b) Development of efficient test schedule construction techniques for specifically diagnosing intermittent faults.

(c) Evaluation and comparison of the performance of the different types of schedules under actual intermittent fault conditions.

6.2 PROJECT RESTRICTIONS

In common with many other published works in the field of fault diagnosis, various limitations and restrictions were applied to the type of fault and the class of circuitry considered in the project. These restrictions were necessary to enable solutions to the problem at hand to be derived within the limited bounds of time and effort available.

The various restrictions were:

(a) Faults were logical and of the s-a-0 or s-a-1 variety.

(b) No circuit redundancy existed, so each fault affected the circuit behaviour in some way.

(c) Only one fault occurred at any given time.

(d) Intermittent faults were well behaved (see Section 3).

(e) Diagnosis was limited to the circuit gate level.

(f) Only small sized circuits of the combinational and asynchronous sequential class were considered.

(g) No circuit race analysis was included in the test selection procedures.

6.3 PROJECT IMPLEMENTATION

Various techniques exist for determining the effects of a particular fault occurrence on the operational characteristics of a circuit to permit the construction of diagnostic testing schedules. Simulation by computer program was adopted for this project. This offered good flexibility for the injection of intermittent faults.

A simulation routine was written as a FORTRAN IV subroutine using the table-driven simulation technique. As a subroutine it was used by other programs in the construction and evaluation of testing schedules. A more simple and efficient technique, that of table look-up, was used for simulating combinational circuits.

Various analytical techniques exist for selecting and evaluating tests for inclusion in testing schedules. The approach used for combinational circuits was based on the Fault-Matrix, which was used as the table for the look-up simulation process. The technique used for asynchronous sequential circuits was the Sequential Analyser. This used the simulation subroutine to determine the effects of the various fault conditions on the circuit operation when stimulated by input tests.

Combinational and Sequential circuits were each considered separately in the following manner:

(a) A schedule construction program was written for diagnosing solid faults.

(b) The operation of the solid fault schedule, (a) above, was evaluated under intermittent fault conditions using the simulation routine to generate diagnostic-fault-profiles (d.f.p.)

(c) The d.f.p. and the solid fault schedule together were used to diagnose intermittent faults.

(d) Using a simple philosophy for test selection and schedule formation, a procedure was developed and a program written for constructing efficient testing schedules specifically for diagnosing intermittent faults. This involved the formation of testing loops in the schedule where time was spent waiting for the fault to re-appear ready for relevant diagnosis to quickly continue.

(e) The different schedules were evaluated by performing extensive simulation experiments. Evaluation was performed under exactly similar intermittent conditions using a computer program which enabled faults to be injected into the simulated circuit during application of the schedule tests. Fault intermittency was modelled using a first-order two-state Markov process and a pseudo-random number generation routine.

(f) The results of the evaluation procedures, (e) above, permitted comparisons to be made of the operational effectiveness of each schedule. For combinational circuits, comparisons of the results against a standard intermittent fault location technique was also performed.

6.4 PROJECT RESULTS

Circuit operation, both in the fault-absent and fault-present conditions, was successfully modelled by the selected simulation processes. Using the Fault-Matrix, simulation rates of 1200

test applications per second were obtained in the schedule construction and evaluation programs. For sequential circuits the FORTRAN IV simulation subroutine produced a simulation rate of approximately 100 tests per second.

Solid fault diagnostic testing schedules were constructed. The sequential circuit construction procedure, the Sequential Analyser, was examined in detail to determine the effects of altering the usage of the various heuristics. This showed that the WANDER heuristic was the most successful at increasing the effectiveness and capabilities of the derived schedules.

Diagnostic-fault-profiles were determined from the solid fault schedules. These were used in conjunction with the schedules to diagnose intermittent faults for both combinational and sequential circuits. The diagnostic resolution decreased in comparison with the case of solid faults because of the existence of partial indistinguishability. This was determined to be an inherent feature related to the intermittency, not the diagnostic technique.

Schedules with characteristic loop structures were derived for specifically diagnosing intermittent faults. Partial indistinguishability again caused a loss in diagnostic resolution in comparison with the solid fault case. Schedules consisted, in total, of a much larger number of tests than the solid fault schedules for the same circuit and required much larger construction program run-times. The effects of varying the usage of the various heuristics in the sequential circuit Sequential Analyser were investigated. WANDER again proved to be the most sensitive heuristic. Excessive program run-times limited the extent of this investigation.

The different types of schedule were evaluated under various intermittent fault conditions. Although the nature of the physical causes of fault intermittency and the exact form of real occurrences were unknown, each schedule was evaluated under exactly similar simulated patterns of fault occurrence, thus allowing valid comparisons to be made. Since this was a formal experiment, even though only a simulation, the statistical significance of the results had to be calculated. Each determination of mean time to diagnosis was calculated from a sample of large enough size, and the appropriate r.m.s. errors calculated. The results showed, in general, that the schedule designed specifically for diagnosing intermittent faults was much more effective than the d.f.p. technique, especially with very sparse fault occurrences. This was found to be true for both combinational and sequential circuits, although the differences for the latter type of circuit were not so marked. A comparison of these results against a standard technique used for diagnosing intermittent faults in combinational circuits showed that for sparse occurrences, the schedule designed specifically for diagnosing intermittent faults was the most effective technique.

An interesting and potentially powerful diagnostic aid was developed based on the fault probability model. This allowed the probability distributions of indistinguished sets of faults to be calculated to aid the maintenance engineer in repairing the most probable fault.

6.5 CONCLUSIONS

The main objectives of this project, to investigate techniques for diagnosing intermittent faults in logic circuits, have been successfully attained.

Two different technuques, one based on interpreting

diagnostic data from solid fault schedules and the other on deriving schedules from first principles specifically for diagnosing intermittent faults, were developed and evaluated.

The results from the evaluation procedures enables the conclusion to be made that the schedule designed specifically for diagnosing intermittent faults is the most effective.

These results must be considered in the light of the assumptions, limitations and restrictions imposed during the project. In general faults do not always occur singly, are not well-behaved, and are not always logical and of the s-a-1 or s-a-0 variety. Modern electronic logic systems are generally large and complex with the smallest replaceable components themselves being complicated assemblies of circuit element gates. Commonly used circuits are usually not pure asynchronous sequential, but rather a mixture of synchronous and asynchronous.

As with most published work in this field, such assumptions are essential if the problem is to be of size suitable for investigation. This has been a research rather than development project and the general techniques of solving the problems of fault intermittency have been of more concern than the problems of finding immediately applicable practical solutions and implementations.

The limitations and restrictions indicate areas for investigations in the future.

(a) Location of intermittent faults to sub-system or replaceable package level.

(b) Location of intermittent faults which are <u>not</u> well-behaved.

(c) Application of some of the more analytical test selection and evaluation techniques to find those tests sensitive to intermittent fault occurrences in all classes of circuitry.

(d) Use of a more realistic simulation technique, such as three-value simulation, to permit circuit race analysis.

(e) Dealing with multiple faults.

(f) Development of better fault models.

(g) Determination of the physical parameters which control the nature of the intermittency. Use this information, together with an appropraite fault model, to predict the fault probability occurrence patterns and to calculate fault probability distributions.

Certain practical limitations were encountered during implementation of the techniques developed during this project. These were related to the use of a high level language, FORTRAN IV, for the various computer programs. More efficient implementations for the future would be obtained through the use of a basic machine language. This would permit the circuit functions and operations to be modelled with instructions working at a level approaching that of the circuit itself. This would results in an increase in size of circuit capable of being analysed and in the speed of simulation.

A further advantage was identified for schedules designed specifically for intermittent fault diagnosis (p.149, 3.5). Such schedules could be used in operational systems to allow normal operations to continue during fault absent conditions. When a fault is detected the schedule is accessed at the appropriate loop. When looping occurs, the fault has disappeared and normal operations continued. Eventually diagnosis is reached.

REFERENCES

(1) PREISS,R.J.: 'Introduction, Chapter (i), "Design Automation of Digital Systems"'(Ed. BREUER,M.A.); Prentice-Hall, 1972, pp.1-19.

(2) BREUER,M.A.: "Recent Developments in the Automated Design and Analysis of Digital Systems"; Proc. IEEE, Jan 1972, Vol.60, No.1, pp.12-27.

(3) CRAY,S.R., and KISCH,R.N.: "A Progress Report on Computer Applications in Computer Design"; Proc. WJCC, 1956, pp.82-85.

(4) HERSKOWITZ,G.J. (Ed.): "Computer-Aided Integrated Circuit Design"; McGraw-Hill, 1968.

(5) BREUER,M.A. (Ed.): "Design Automation of Digital Systems"; Prentice-Hall, 1972, pp.21-100.

(6) DIMSDALE,B., and MARKOWITZ,H.M.: "A Description of the Simscript Language", IBM Systems J., 1964, Vol.3, p.57-67.

(7) HERSCOUTCH,H., and SCHNEIDER,T.: "GPSS III an Expanded General Purpose Simulator"; IBM Systems J., 1965, Vol.4, pp.174-183

(8) PROCTOR,R.M.: "A Logic Design Translator Experiment Demonstrating Relationship of Language to Systems and Logic Design"; IEEE Trans. E.C.; 1964, EC13, pp.422-430.

(9) FALKOFF,A.D., IVERSON,K.E., and SUSSENGUTH,E.H.: "A Formal Description of System/360"; IBM Systems J., 1964, Vol.3(2).

(10) DULEY,J.R., and DIETMEYER,D.L.: "Translation of a DDL Digital System Specification to Boolean Equations"; IEEE Trans. on Computers, 1969, Vol.C-18, pp.305-313.

(11) FRIEDMAN,T.D., and YANG,S.C.: "Methods Used in an Automated Logic Design Generator (ALERT)"; IEEE Trans. on Computers, 1969, Vol.C-18, pp.593-613.

(12) McCLUSKEY,E.J.: "Minimization of Boolean Functions"; BSTJ, 1956, Vol.35, pp.1417-1444.

(13) LEWIN,D.: "Logical Design of Switching Circuits"; Nelson, 1968.

(14) LEWIN,D.W., and WATERS,M.C.: "Computer Aids to Logic Design"; Computer Bulletin, 1969, Vol.13, No.11, pp.382-388.

(15) DIETMEYER,D.L., and SCHNEIDER,P.R.: "A Computer-oriented Factoring Algorithm for NOR Logic Design"; IEEE Trans. E.C., 1965, Vol.EC-14, pp.868-875

(16) RAMAMOORTHY,C.V.: "A Structural Theory of Machine Diagnosis"; Proc. SJCC, 1967, pp.743-756.

(17) LAWLER,E.L., LEVITT,K.N., and TURNER,J.: "Module Clustering to Minimize Delay in Digital Networks", IEEE Trans. on Computers, 1969, Vol.EC-18, pp.47-57.

(18) KODRES,U.R.: 'Partitioning and Card Selection, Chap.(iv), "Design Automation of Digital Systems", (ed. BREUER,M.A.); Prentice-Hall, 1972, pp.173-212.

(19) KERNIGHAN,B.W., and LIN,S.: "An Efficient Heuristic Procedure for Partitioning Graphs"; BSTJ, 1970, Vol.49, pp.291-307.

(20) NOTZ,W.A., et al.: "Large Scale Integration: Benefitting the System Designer"; Electronics, 1967, pp.130-141.

(21) HANAN,M., and KURTZBERG,J.M.: 'Placement Techniques, Chap.(v), "Design Automation of Digital Systems", (Ed. BREUER, M.A.); Prentice-Hall, 1972, pp.213-282.

(22) AKERS,S.B.: 'Routing, Chap.(vi), "Design Automation of Digital Systems", (Ed. BREUER,M.A.); Prentice-Hall, 1972, pp.283-333.

(23) MARCONI COMPANY: "Electronic Production Testing in British Industry"; Marconi Company, Research Division, 1969, Vol.1-12.

(24) LEWIN,D.: "Theory and Design of Digital Computers"; Nelson 1972.

(25) FOWLER,F.H.: "Salvaging Diagnostic Fault Dictionaries and Fault Tallies After a Minor Logic Change in the Tested Device"; Digest 6th Ann. IEEE Computer Soc. Int. Conf., 1972, pp.251-254.

(26) LEWIN,D.W., PURSLOW,E.J., and BENNETTS,R.G.: "Computer Assisted Logic Design - the CALD System"; IEE Conf. on Computer Aided Design, 1972, Pub.No.86, pp.343-351.

(27) RAMAMOORTHY,C.V., CHANG,L.C.: "System Modelling and Testing Procedures for Microdiagnostics"; IEEE Trans. on Computers, 1972, Vol.C-21(11), pp.1169-1183.

(28) CHANG,H.Y. MANNING,E.G., and METZE,G.: "Fault Diagnosis of Digital Systems"; Wiley-Interscience, 1970.

(29) FRIEDMAN,A.D., and MENON,P.R.: "Fault Detection in Digital Circuits"; Prentice-Hall, 1971.

(30) BREUER,M.A.: "Automatic Digital Test Generation for Sequential and Combinational Logic"; Proc. National Electronics Packaging Conf., 1972.

(31) ARMSTRONG,D.B.: "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets"; IEEE Trans. E.C., 1966, Vol.EC-15, pp.66-73.

(32) ROTH,J.P.: "Diagnosis of Automation Failures: A Calculus and a Method"; IBM Journ. of R. and D., 1966, pp.278-291

(33) SELLERS,F.F., HSIAO,M.Y., and BEARNSON,C.L.: "Analyzing Errors With the Boolean Difference"; IEEE Trans. on Computers, 1968, Vol.C-17, pp.676-683.

(34) SESHU,S.: "On An Improved Diagnosis Program"; IEEE Trans. E.C., 1965, Vol.EC-14, pp.76-79.

(35) POAGE,J.F.: "Derivation of Optimum Tests to Detect Faults in Combinational Circuits"; Proc. Symposium on Mathematical Theory of Automata, 1963, pp.483-528.

(36) GALEY,J.M.,NORBY,R.E., and ROTH,J.P.: "Techniques for the Diagnosis of Switching Circuit Failures", IEEE Trans. on Comm. Elect., 1964, pp.509-514.

(37) CHANG,H.Y.: "An Algorithm for Selecting an Optimum Set of Diagnostic Tests"; IEEE Trans. on E.C., 1965, Vol.EC-13, pp.706-711.

(38) SESHU,S., and FREEMAN,D.N.; "The Diagnosis of Asynchronous Sequential Switching Systems"; IRE Trans. E.C., 1962, Vol.EC-11, pp.459-465.

(39) BRULE,J.D., JOHNSON,R.A., KLETSKY,E.J.: "Diagnosis of Equipment Failures"; IRE Trans. on Reliab. and Quality Control, 1960, Vol.RQC-9, pp.23-34.

(40) CHANGE,H.Y.: "A Distinguishability Criterion for Selecting Efficient Diagnostic Tests"; AFIPS Proc. SJCC, 1968, Vol.32, pp.529-534.

(41) KAUTZ,W.H.: "Fault Testing and Diagnosis in Combinational Digital Circuits"; IEEE Trans. on Computers, 1968, Vol.EC-17, pp.352-366.

(42) VON NEUMANN,J.: "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components"; Automatic Studies, Annals of Math. Studies No.34, Princeton University Press, 1956, pp.43-98.

(43) TRYON,J.G.: "Quadded Logic", Redundancy Techniques for Computing Systems (Ed. WILCOX and MANN); Spartan Books, 1962, pp.205-228.

(44) KLASCHKA,T.F.: "Two Contributions to Redundancy Theory"; Proc. Eighth Annual Symp. on Switching and Automata Theory, 1967, pp.175-183.

(45) ARMSTRONG,D.B.: "A General Method of Applying Error Correction to Synchronous Digital Systems"; BSTJ, 1961, Vol.40, pp.577-593.

(46) RAY-CHAUDHURI,D.K.: "On the Construction of Minimally Redundant Reliable System Designs"; BSTJ, 1961, Vol.40, pp.595-611.

(47) AVIZIENIS,A., et al.: "The STAR (SELF-TESTING-AND-REPAIRING) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design"; IEEE Digest International Symp. on Fault-Tolerant Computing, 1971, pp.92-96.

(48) DOWNING,R.W., NOWAK,J.S., and TUOMENOKSA,L.S.: "No.1 ESS Maintenance Plan"; BSTJ, 1964, Vol.43, pp.1961-2019.

(49) AVIZIENIS,A.: "An Experimental Self-Repairing Computer"; Proc. IFIP Congress, 1968, pp.29-33.

(50) KOHAVI,Z., and LAVALLEE,P.: "Design of Diagnosable Sequential Machines"; AFIPS Proc. SJCC, 1967, Vol.30, pp.713-718.

(51) HALL,A.D.: "Synthesis of Double-Rank Sequential Circuits", Princeton Univ. Tech. Report No.53, 1966.

(52) FRIEDMAN,A.D., and MENON,P.R.: "Design of Generalized Double-Rank and Multiple-Rank Sequential Circuits"; Information and Control, 1969, Vol.15, pp.436-451.

(53) IEEE COMPUTER SOCIETY: "1971 International Symposium on Fault-Tolerant Computing"; 1971, Pasadena, California.

(54) BENNETTS,R.G., and LEWIN,D.W.: "Fault Diagnosis of Digital Systems - A Review"; The Computer Journal, 1971, Vol.14(2), pp.199-206

(55) PREISS,R.J.: 'Fault Test Generation, Chapter (vii), "Design Automation of Digital Systems" (Ed. BREUER,M.A.)'; Prentice-Hall, 1972, pp.335-409.

(56) BALL,M., and HARDIE,F.: "Effects and Detection of Intermittent Failures in Digital Systems"; Proc. FJCC, 1969, Vol.35, pp.329-335.

(57) KINGSNORTH,R.L.: "Some Interactions of Aluminium and Silicon"; Reliability Physics of Electronics Components (Institute of Physics-Electronics Group), 1972.

(58) BREUER,M.A.: "Generation of Fault Detection Tests for Intermittent Faults in Sequential Circuits"; Digest of the International Symp. on Fault-Tolerant Computing, 1972.

(59) ROTH,J.P., et al.: "Programmed Algorithms to Compute Tests to Detect Between Failures in Logic Circuits"; IEEE Trans. on Computers, 1967, Vol.EC-16, pp.567-580.

(60) BENNETTS,R.G.: "The Diagnosis of Logical Faults", Wireless World, July 1971, pp.325-328.

(61) SCHEFF,B.H., and YOUNG,S.P.: 'Gate-Level Logic Simulation, Chapter (iii), "Design Automation of Digital Systems" (Ed. BREUER,M.A.)'; Prentice-Hall, 1972, pp.101-172.

(62) KATZ,J.H.: "Optimizing Bit-Time Computer Simulation"; Comm. ACM, 1963, Vol.6(11), pp.679-685.

(63) YETTER,I.H.: "High Speed Fault Simulation for UNIVAC 1107 Computer Systems"; Proc. ACM Nat. Conf., 1968, pp.265-277.

(64) HARDIE,F.H., and SUHOCKI,R.J.: "Design and Use of Fault Simulation for Saturn Computer Design"; IEEE Trans. E.C., 1967, Vol.EC-16, pp.412-429.

(65) McCLURE,R.M.: "A Programming Language for Simulating Digital Systems"; Jour. ACM, 1965, Vol.12, pp.14-22.

(66) McKINNEY,J.G.: "Synchronous Logic Simulation"; Proc. SHARE-ACM Design Automation Workship, 1967, pp.19-22.

(67) ULRICH,E.G.: "Time-Sequenced Logical Simulation Based in Circuit Delay of Active Network Paths"; Proc. ACM 20th Nat. Conf., 1965, pp.437-448.

(68) McKay,A.R.: "Comment on 'Computer-Aided Design: Simulation for Digital Design Logic'."; IEEE Trans. on Computers, 1969, Vol.C-18, p.862.

(69) HAYS,G.G.: "Computer-Aided Design: Simulation of Digital Design Logic"; IEEE Trans. on Computers, 1969, Vol.C-18, pp.1-10.

(70) SHAFER,L., and SCHEFF,B.H.: "Efficient Simulation Within a Comprehensive Design Automation System"; Proc. Joint Conf. on Mathematical and Computer Aids to Design, 1969.

(71) CASE,P.W., et al.: "Solid Logic Design Automation"; IBM Journ. of R. and D., 1964, Vol.8, pp.127-140.

(72) GORMAN,D.F., and ANDERSON,J.P.: "A Logic Design Translator"; Proc. FJCC, 1962, pp.251-261.

(73) TSIANG,S.H., and ULRICH,W.: "Automatic Trouble Diagnosis of Complex Logic Circuits"; BSTJ, 1962, Vol.41, pp.1177-1200.

(74) CHANG,H.Y., and THOMIS,W.: "Methods of Interpreting Diagnostic Data for Locating Faults in Digital Machines"; BSTJ, 1967, Vol.46(2), pp.289-317.

(75) EICHELBERGER,E.B.: "Hazard Detection in Combinational and Sequential Switching Circuits"; IBM Journ. of R. and D., 1965, Vol.9, pp.90-99.

(76) JEPHSON,J.S., McQUARRIE,R.P., and VOGELSBERG,R.E.: "A Three-Value Computer Design Verification System"; IBM Systems Journ., 1969, No.3, pp.178-188.

(77) ELRED,R.D.: "Test Routines Based on Symbolic Logic Statements"; Journ. ACM, 1959, Vol.6(1), pp.33-36.

(78) SCHNEIDER,P.R.: "On the Necessity to Examine D-Chains in Diagnostic Test Generation - An Example"; IBM Journ. of R. and D., 1967, Vol.11(1), p.114.

(79) ROTH,J.P., et al.: "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits"; IEEE Trans. E.C., 1967, Vol.EC-16(5), pp.567-579.

(80) BENNETTS,R.G.: "A Realistic Approach to Detection Test Set Generation for Combinational Logic Circuits"; The Computer Journal, 1972, Vol.15, pp.238-246.

(81) McCLUSKEY,E.J.: "Introduction to the Theory of Switching Circuits"; McGraw-Hill, 1965.

(82) KAUTZ,W.H.: "Fault Diagnosis in Combinational Digital Circuits"; First Ann. IEEE Comp. Conf., 1967, pp.2-5.

(83) SHANNON,C.E.: "A Mathematical Theory of Communication"; BSTJ, 1948, Vol.27, pp.379-423, 623-656.

(84) MANDELBAUM,D.: "A Measure of Efficiency of Diagnostic Tests Upon Sequential Logic"; IEEE Trans. E.C., 1964, Vol.EC-13, p.630.

(85) HENNIE,F.C.: "Fault Detection Experiments for Sequential Circuits"; Circuits"; Proc. 5th Ann. Symp. on Switching Theory and Logical Design, 1964, pp.95-110.

(86) HUFFMAN,D.A.: "The Synthesis of Sequential Switching Circuits"; Journ. of the Franklin Institute, 1954, Vol.257(3,4), pp.161-190, 275-303.

(87) POAGE,J.F., and McCLUSKEY,E.J.: "Derivation of Optimum Test Sequences for Sequential Machines"; 5th Ann. Symp. on Switching Theory and Logical Design, 1964, pp.121-132.

(88) KOHAVI,Z., and LAVALLEE,P.: "Design of Sequential Machines with Fault Detection Capabilities"; IEEE Trans. E.C., 1967, Vol.EC-16(4), pp.473-484.

(89) KUBO,H.: "A Procedure for Generating Test Sequences to Detect Sequential Circuit Failures"; NEC Res. Develop., 1968, No.12.

(90) BOURICIUS,W.G., et al.: "Algorithms for Detection of Faults in Logic Circuits"; Proc. Int. Symp. on Fault Tolerant Computing, 1971, pp.5-8.

(91) BREUER,M.: Generation of Fault Detection Tests for Sequential Circuits"; Proc. Int. Symp. on Fault Tolerant Computing, 1971, pp.18-21.

(92) CARTER,W.C., et al.: "Design of Serviceability Features for the IBM/360"; IBM Journ. of R. and D., 1964, Vol.8(2), pp.115-126.

(93) HSIAO,M.Y., and CHIA,D.K.: "Boolean Difference for Fault-Detection in Asynchronous Sequential Machines"; Proc. Int. Symp. on Fault Tolerant Computing, 1971, pp.9-13.

(94) ROSS,S.M.: "Introduction to Probability Models"; Academic Press, 1972, pp.87-89.

(95) PARADINE,C.G., and RIVETT,B.H.P.: "Statistical Methods for Technologists"; The English Universities Press, 1960.

ACKNOWLEDGEMENTS

I would like to thank members of the staff of the Physics Department and the Computer Operations section for their help and advice throughout this project; also [redacted] of the University of Southern California for suggesting this as a suitable topic in need of investigation; and the advice of Dr. J. I. Missen and [redacted].

This work was supported through the Science Research Council Research Studentship scheme. I am particularly grateful to the Atwell Grants Committee of the Skinners' Company for a grant to permit completion of the writing of this thesis.

The work of [redacted] in typing this thesis is gratefully acknowledged.