LOGIC DESIGN TESTING.

This thesis is presented to the Kingston Polytechnic and the Council for National Academic Awards for the degree of Doctor of Philosophy

by

A.A. Kaposi, Dipl.Ing.

January 1971.

A B S T R A C T

This thesis concerns the testing of the design of logic networks.

It is shown that conventional test methods, such as hardware testing and computer simulation, fail to satisfy the test requirements of modern logic networks.

A new method is devised, consisting of a series of computer based modular tests, which permit the comprehensive verification of designs.

The feasibility of the new method has been demonstrated on a prototype system.

The work is based on a systems engineering approach which permits viewing the problems of logic design as particular cases of the more general problems of designing large interactive engineering systems. The systems approach also permits the extension of the methods described in this thesis to other areas of engineering.

As part of the thesis, a framework of systems engineering concepts is constructed.

ACKNOWLEDGEMENTS

The work described in this thesis was carried out under the supervision of Mr. G. Mayo of the Kingston Polytechnic and of Dr. M.J. Lanigan of the Plessey Company Limited.

The author wishes to acknowledge the contributions of the staff of Plessey Radar Limited, who originated the project and have kept in touch throughout its development; of the staff of the Computer Unit of Kingston Polytechnic for valuable advice and assistance in programming, and to Mr. D.R. Holmes for devising and implementing the prototype system.

Grateful thanks are due to Dr. Lanigan for continued support, guidance and encouragement; to Mr. Mayo for ready help and valuable discussions; to Mr. R.H. Ness and the Directorate of Kingston Polytechnic for providing every facility in the interest of the project.

The course of this work was considerably influenced by the advice of Mr. J.F. Kaposi who scrutinised each stage of its development and by that of Mr. D. Rzevski, who suggested the systems approach.

Finally, and most of all, the author is grateful to members of her family whose unfailing patience made the undertaking of such a work possible.

CONTENTS

CHAPTER 1

Background.

CHAPTER 1 - BACKGROUND.

In the autumn of 1967 Plessey Radar Limited approached the Kingston Polytechnic (then Kingston College of Technology) with the problem of inadequate design reliability of their logic networks. It was thought that the already inadequate design test methods would fail completely when, as a consequence of advancing integrated circuit technology, it will become possible to produce large interactive digital networks as indivisible single components. Thus the management of Plessey Radar Limited requested that the Polytechnic should undertake the development of a method of logic design testing which could satisfy both existing and projected demands.

By the definition of the problem a hardware model of the newly designed network could not always be assumed to be available; thus it was necessary to concentrate upon a computer-based method of solution.

Design testing was seen as a means of generating correct designs. Thus it was thought necessary to construct a design test method which, beyond providing means of comprehensive error detection, also facilitated the correction of these design errors.

Computer simulation was considered as a possible mode of solution. At the time one simulator was generally known and commercially available in this country (36); since then the method has become conventional and has been implemented in some form by numerous industrial and research organisations (for a survey see (1)). These now conventional logic simulators operate upon a computer model of the network by applying a sequence of signal changes to the input terminals and recording the signals appearing at the outputs. This record - a waveform on some suitably designed time scale - contains a part of the required error report in an implicit form. Some of the design errors are not revealed by the test; those detected must be recognised, sorted and classified by the designer upon examination of the output waveforms.

In view of the size and complexity of modern logic networks, the method is considered inadequate. To ensure reliability, a very long sequence of input changes must be originated by the designer; this places extravagant demands on the computer's time, while the very high volume of output data gives an unreasonable task to the designer as a data processor. It is shown in the course of this work that conventional logic simulators appear to raise almost as many problems as they solve. Thus it was thought necessary to search for some alternative method.

The work programme which was proposed for this project in 1968 contained two ideas:

1) that simulators should be purpose-built for a type of network or for an aspect of performance.
2) that care must be taken to use computing facilities efficiently.

This work was divisible into four distinct parts:

a) problem analysis.
b) formulation of the principles of solution.
c) development of techniques of solution.
d) design and implementation of a prototype system, proving the feasibility of the principles and techniques.

The work programme isolated the last item of this list, which was subsequently defined as an individual research project (1) and was carried out under the direct financial sponsorship of Plessey Radar Limited. The implementation project ran concurrently with the work described in this thesis and under the supervision of the author.

Detailed problem analysis revealed severe limitations of simulation, even in its unconventional form, as a means of analysis. It was found necessary to have the freedom of choice between alternative modes of analysis. Thus the changed title of this thesis does not contain the word "simulation".

The development of the project was considerably influenced by a newly emerging interest of a group of people in the Polytechnic in the principles of systems engineering. Common ground was found to exist between projects of seemingly unrelated fields of specialisation and a broadening of outlook permitted useful interchange of ideas between a group of researchers.

In order that logic networks may be placed against a systems engineering background, a framework of consistent concepts and definitions was needed. Since both systems engineering and switching theory are relatively new and rapidly developing subjects, such a framework was not readily available but had to be constructed by adopting existing material, modifying such material, or in many cases, creating new concepts, definitions and terminology. This permitted the viewing of the project against the background of present-day technology and thus it opened the way for the extension of this work to fields outside of logic network analysis. As a consequence, several new research projects have been initiated for members of staff and for post-graduate students (see Chapter 6).

Using the framework of systems engineering, it was possible to propose a model of the design process. Viewing logic network analysis as part of such a procedure, it was found unwise to computerise this part of the process alone. Instead, it seemed desirable to devise methods of automating a larger section of the process, including the assessment of the model performance. The combination of analysis and performance assessment will be termed design testing or verification.

CHAPTER 2

THE FRAMEWORK OF SYSTEMS CONCEPTS.

2.1 - INTRODUCTION.

A system will be described by a model and characterised by its structure and parameters. In this chapter the methods of characterising and analysing systems are discussed and, finally, a model of the systems design process is presented.

2.2 - THE CONCEPT OF A SYSTEM.

2.2.1 - Behaviour, environment, specification.

A system will, in the first instance, be defined as an assembly of objects united by some form of interaction or interdependence. (Note that a more general definition (5) permits a system to contain non-interactive, isolated objects or groups of objects. Systems considered here, which contain no such objects, are defined in (5) as reduced systems). Thus the concept of a system is that of an indivisible entity since all the parts interact and none can be isolated without altering the behaviour of the system; the system concept demands the examination of the overall interaction of a group of objects rather than focussing attention upon the operation of each of the constituent objects in turn.

Behaviour itself is defined as a unique mapping or transformation of inputs (causes) into outputs (effects) and the system will be regarded as the operator performing this transformation. Irrespective of size, appearance, structure or other circumstance of detail, two systems will be regarded as equivalent if their behaviour is identical, that is, if they could be interchanged without altering the relationship between cause and effect.

The system is contained within a boundary. The set of all variables outside the boundary which have an effect upon the system form the system environment. The total set of admissible values of the variables of the system environment forms the admissible domain of the system.

The system environment consists of resources which must be available for the operation of the system (sources of energy, manpower, etc.), the physical conditions in which the system operates (such as altitude, temperature, humidity, etc.), the operating signals available to the system and, finally, the loads to which the system provides a service. Henceforward it will be taken for granted that adequate resources are provided at all times; it will also be assumed that the system boundary is drawn in such a way that the load outside this boundary should be constant.

The rest of the environment will have an effect upon the behaviour of the system; all the environmental variables should be considered as inputs of the system.

Let the system have a total of n input variables forming a set $u_n = \left(U_1, U_2, \ldots U_n \right)$ and consisting of a set u_p of p number of physical signals and a set u_f of f number of operating signals or forcing functions. The admissible domain of input variables encloses an n-dimensional input space which defines the admissible environment u_n of the system. Given a system, there is then a set of u_n of input variables U and a set Z of outputs Z and a transformation F ( ) mapping the inputs into the outputs.

$$\text{Thus } Z = F(u);\quad u_n = \left(U_1, U_2, \ldots U_n \right);\quad Z = \left(Z_1, Z_2, \ldots Z_m \right)$$

Equation 2.2.1 (see footnote)

for a network of n inputs and m outputs.

FOOTNOTE: Equations, tables and figures will be numbered throughout this thesis by assigning to the first digit the chapter number, to the second digit the section number within the chapter and to the third digit a serial number within the section.

This transformation will, in the first instance, define the behaviour of the system.

The formal statement of the system behaviour, together with the definition of the admissible environment, will be a form of system specification. The system specification will thus contain the description of the domain of physical variables, and the domain of operating signals. It will also contain the definition (stochastic or deterministic) of the expected system transformation.

2.2.2 - State.

When defining the system as the transformation $z = F(u)$, it was tacitly assumed that the system was created at $t = -\infty$ and that the history of the system had been retained for all time t up to the time of observation. If this assumption is not valid and the inputs are only known after a given instant t_o, then the history of the inputs previous to t_o will have some effect upon the system. The accumulation of these effects over the interval $-\infty \leq t \leq t_o$ is called the initial state or the state of the system at $t = t_o$. Thus the state is defined as a set of time functions $s(t)$ such that if the input set is known for all $t \lesseqgtr t_o$, then the knowledge of $s(t_o)$ is sufficient to determine uniquely the output set $z(t)$ for all $t \gtrsim t_o$.

With the aid of this definition, equation 2.2.1 can be re-written in an equally general but more useful form:

$$z(t) = G(u(t), s(t_o)); \quad u = \left\{ U_1, U_2, \ldots U_n \right\}; \quad z = \left\{ z_1, z_2, \ldots z_m \right\};$$

$$s = \left\{ S_1, S_2, \ldots S_q \right\} \qquad \text{Equation 2.2.2}$$

for a system of n inputs, m outputs and q states.

Although it is often advantageous to prepare abstract models of systems such that S should be an empty set, physically realisable systems always possess state. Thus, in general, the knowledge of the state of the system is necessary at all time $t \geqslant t_o$. The system equation will consist of the output equation which specifies the behaviour and the state equation which permits the computation of the system state at a time $t \geqslant t_o$.

The general form of the state equation (7) gives the state at time t as

$$s(t_1) = H(u(t_o, t_1), s(t_o)) \text{ .... Equation 2.2.3}$$

Here $u(t_o, t_1)$ expresses the input segment over the time interval (t_o, t_1).

A particular form of this equation will be presented in Chapter 4 as the state equation of logical networks.

The concept of state permits the definition of two useful properties of a system (6):

a) <u>Controllability</u> (see also Connectedness, Chapter 4)

The component $S_i(t)$ of the state $s = \left\{S_j\right\}$, $j = 1, 2, \ldots, i, \ldots k)$ is controllable if there exists an input $u(t)$, $t > t_o$ to bring any prescribed initial value α of $S_i(t_o)$ to any other prescribed final value β of $S_i(T)$ in a finite amount of time $(T - t_o)$.

If all components of the state s of the system are controllable then the system is termed <u>completely controllable</u>.

b) <u>Observability</u> (see also definition of sequential networks Chapter 4).

The component $S_i(t)$ of the state $s = \left\{S_j\right\}$ <u>is observable</u> if there is some finite time T for which a knowledge of the response $z(t)$ over $t_o \leqslant t \leqslant T$ is sufficient to determine the initial value $S_i(t_o)$ when the state equations of the system are known. Thus an observable state can be determined by observations made on the output. If all components of state s are observable then the system is called <u>completely observable</u>.

2.3 - SYSTEM ORGANISATION.

2.3.1 - Resolution.

By definition, a system is a complex interactive assembly of identifiable objects which themselves may be complex interactive assemblies. Thus, by re-defining the boundary, the objects of a given system may be considered as systems themselves, as the given system might become one of several objects of a larger system. The organisation of a system will be considered as a hierarchical structure of sub-systems.

When seeking understanding of a given system, it is frequently necessary to <u>resolve</u> the system to constituent objects. Starting from level 1, when the system P is observed as a whole, the observer may choose to define a number of distinct <u>resolution levels</u> as shown on the <u>resolution graph</u> of a fictitious system (Fig. 2.3.1), suggested by Klir and Valach (5). Each node of this directed graph represents a different system which features as one or more objects of the system at a lower level of resolution. The relationships are simbolized by the arrows; for instance, the system P1 may be constituted of sub-systems P2 and P3; the system P7 is the constituent of systems P4, P5 and P6.

As an example, consider the system P1 of Fig. 2.3.1 once again. Assume that P6 needs no detailed examination, i.e. the highest level of resolution will contain the sub-system P6. P1 may now be constituted in a number of different ways:

of P6, P5 and P2.
of P6, P7 and P4.
of P6, P4 and P5, etc.

In principle all resolution graphs are "cigar-shaped", with arrows starting from a single point of the common constituent of all sub-systems (the highest level) and terminating on a single point of the total system (the lowest level). The peak of the resolution graph is seldom reached, because of practical reasons; the number of sub-systems constituting the total system increases as the level of resolution increases and it becomes impossible for the designer to consider the interaction between the large number of system objects simultaneously.

Resolution graphs are not unique to a given system; since sub-system boundaries may be drawn in a variety of ways, resolution graphs may also be prepared in a corresponding variety.

Consider now the example of Fig. 2.3.2. The selected system is an amplifier, composed of active elements and coupling networks. The equivalent circuit model of active elements and the circuit elements of the coupling networks form further sub-systems. The graph is not continued to higher levels, but it is evidently possible to prepare complex models of each component, accounting for losses, noise, thermal effects, etc., until, ultimately, the graph would reach its peak at a level where sub-systems represent material particles.

It must be noted that this graph offers no insight to the number of objects within the system at each level of resolution, nor to the way in which these objects are interconnected. Information about structure must be presented separately for a given degree of resolution in the form of a graph, table or equivalent.

2.3.2 - Structure.

Let the system be denoted by P and let P, at the selected level of resolution, consist of q number of sub-systems (or system objects). Let the totality of these objects be denoted by D, where the set D is defined as

$$D = \{ d_1, d_2, \ldots, dq \}$$

Now let the environment of the system be represented by a "source object" which emits physical and operating signals. Let the source object be denoted by d_o.

Then the system and its environment will consist of a set of objects $\mathfrak{S}$ where

$$\mathfrak{S} = \{ d_o, d_1, \ldots, dq \}$$

The structure of the system is defined by the manner in which the elements of the set D are interconnected. Let the symbol r_{ij} denote the information about the connection of the inputs of object d_j to the outputs of object d_i. Then the set

$$R = \begin{pmatrix} r_{ij} \end{pmatrix} \quad \text{for all } i, j \text{ between 1 and q}$$

will define the system structure at the selected level of resolution.

The system will now be totally defined by the combination of D and R:

$$P = \begin{pmatrix} D, R \end{pmatrix} .$$

The structure of the system and its environment will be defined by ϱ where

$$\varrho = \begin{pmatrix} r_{i,j} \end{pmatrix} \text{ for all i, j between 0 and q.}$$

Denoting the combination of the system and its environment by π, π will be totally defined by the combination of δ and ϱ:

$$\pi = \begin{pmatrix} \delta , \varrho \end{pmatrix}.$$

Information about the system structure may take such forms as graphs, lists or matrices. For the purpose of this work it was found convenient to adopt the concept of a "structure matrix" and propose its definition as follows:-

Let the system comprise q objects and a source object. Then the structure matrix W will be a square matrix consisting of $(q + 1)^2$ number of elements. Each element W_{ij} for all i, j between 0 and q, will be a matrix representing r_{ij}. If the object d_j has "a" number of outputs and the object d_i has "b" number of inputs then the matrix W_{ij} will have "a" number of rows and "b" number of columns. Elements of W_{ij} will be binary numerals showing the presence or absence of a connection between each of the outputs of d_j and each of the inputs of d_i. The matrix d_{kk} represents the feedback connections between outputs and inputs of the k^{th} object of the system.

Fig. 2.3.3 shows in graphical form the structure of a system and its envinonment. As an example of the above definition the structure matrix of the system will now be prepared.

The system comprises q = 2 objects and a source object of a single output and a dummy input. The structure matrix W has $(q + 1)^2 = 9$ elements as listed below:-

$$W_{0,0} = \begin{pmatrix} 0 \end{pmatrix} \qquad W_{0,1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad W_{0,2} = \begin{pmatrix} 0 \end{pmatrix}$$

$$W_{1,0} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \qquad W_{1,1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad W_{1,2} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$$

$$W_{2,0} = \begin{pmatrix} 0 & 1 \end{pmatrix} \qquad W_{2,1} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad W_{2,2} = \begin{pmatrix} 0 & 0 \end{pmatrix}$$

The structure matrix W serves no purpose of algebraic manipulation having matrix elements of uneven size. Even so, it is of considerable value since it organises the numerical description of system structure in an easily comprehensible form.

At the cost of some loss of information a more concise version of the structure matrix may be obtained. The coarse structure matrix W_c will contain a single numeral in the place of each matrix W_{ij}; this numeral will be 0 if all elements of W_{ij} were 0; otherwise it will be 1. The coarse structure matrix indicates the absence or existence of a connection between objects i and j, without specifying the terminals of inter-connection. The coarse structure matrix of the example of Fig 2.3.3 is

$$W_c = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

The first row of W_c is trivial because the source object, by definition, has no inputs. Consequently the reduced coarse structure matrix W_r will contain the same information. For the example

$$W_r = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Structure matrices will find wide application in the course of this work.

2.4 - MODELLING.

In order to obtain a meaningful definition of the system P it is now necessary to define means for the description of the system objects contained in D. If the behaviour of each object in D could be verbally, numerically, graphically or otherwise described, then the behaviour of the total system could be found by combining this description of object behaviour with information about structure.

Due to the diversity of the ways in which D may be described it is useful to adopt the concept of a model. The model of a system will be defined here as an abstraction of the system, constructed for the purpose of giving insight to the system behaviour.

It is possible, and sometimes necessary, to construct models of objects of the system at several different resolution levels. Thus, a model may seek to represent the total system or one of its constituents.

Models of engineering systems are usually quantitative. Such quantitative models will be characterised by an ordered set of numerals called the parameters of the system.

It will be useful to extend the concept of a parameter to non-numerical information about a modelled system. Thus a parameter may denote the colour, shape or logical behaviour of an object of the system.

Modelling of engineering systems will consist of

a) choosing a resolution level, thus defining the boundary of each system object.
b) characterising the system structure and
c) defining of object parameters.

2.4.1 - Modelling the system.

Modelling will be described as the procedure of obtaining the model of the system for the purpose of observing the system behaviour.

The model usually represents a simplified version of the system, purposely built to facilitate the observation of a limited set of its features. Thus the specification of a model would consist of the specification of

a) the set of characteristics of the system which are to be observed and
b) the accuracy with which these characteristics are to be described.

The quality of the model will be defined here by an objective function, taking account of the extent to which the model meets the specifications, the cost of building the model and the facility of it's use.

There is no improvement in quality associated with exceeding the model specifications and since the simplest model is usually easiest to use, this will represent the model of optimum quality. However, the use of complex models is justified when they are flexible and can be used for more than one purpose. Such multi-purpose models share the cost of development between a number of applications and thus acquire higher quality rating, but, since such models depend for their quality upon the need for all applications, they are very sensitive to modifications of the analysis process.

It is usually easier to maintain the quality of models by constructing a modelling sequence which relies upon a sequence of modifications (usually refinements). In this case a model is purpose-built for each application, thus is optimally easy to use, but it is built upon the foundation of another model, used earlier in the process, thus the cost consists of a relatively small increment. Process modification affects the incremental cost only. Sequential modelling will find application in the course of this work.

2.4.2 - Technique.

The technique of modelling is shown schematically on Fig.2.4.1. The modelling data is the basis upon which the hypothesis is set up. When the model is constructed and tested the modelling data is used again as the basis of assessment.

If serial modelling is used, as recommended in section 2.4.1, the process is repeated several times, taking into account an increasing detail of modelling data. Thus a comprehensive model is built by a series of approximations.

2.4.3 - Classification.

Several criteria have been suggested for the classification of models (5), (9), (10), (12). Without further comment some grounds for classification are listed here, taken mainly from Chestunt (11).

1) Language.

   a) Verbal models

   b) Iconic models (maps, photographs, etc.)

   c) Symbolic models (flow charts, logic diagrams, etc.)

d) Analogue models

e) Physical models.

2) Method of solution

a) Analytic models

b) Numerical deterministic models

c) Numerical stochastic models.

3) Resemblance to reality

a) Isomorphic models

b) Homomorphic models

c) Abstract models.

Such formal classifications have been found helpful in understanding the problems of modelling logical networks.

2.4.4 - Modelling the environment.

It is possible to extend the concept of a model to the environment itself. If a suitable model is found to represent, in some abstract form, the source object d_o, then the objects of δ are all modelled and can be observed. This use of the concept of a model is considered both advantageous and novel.

In this section the environment of a system will first be examined; secondly, the problems of modelling the environment will be discussed.

Let the source object d_o have n number of output terminals. This means that the system P is operating in an n-dimensional input space which is composed of operating signals and physical signals. The boundary of this space is given in the specification of the admissible domain.

A verbal model of the source object is now proposed. Let the signal at the i^{th} output terminal of the source object be given by some quantity U_i (t_o) at the instant t_o. Then the totality of U_i (t_o) for all i between 1 and n defines a point γ in n-dimensional space within the admissible domain. The behaviour of the source object in the interval $\tau = t_1 - t_o$ will now be pictured as the parameters in n dimensions of the moving point γ. In the course of normal operation point Y may travel on some path X where X is usually a continuous line.

Let the purpose of an observer be now to obtain information about the system behaviour. The observer may now follow the variations of the system outputs as functions of the path of point Y. This procedure leads to frustration because the point Y may stop moving for periods or may keep returning to routes already covered, missing interesting areas altogether. It appears advantageous to replace the real-world source object with another which is under the observer's control. If the substitution is sufficiently ingenious then the observer may regulate the route X according to his own purpose.

Let the purpose of the observer be re-defined: information is to be collected about the total range of system behaviour. With the aid of the verbal model of the source object it is easy to see that it is impossible to achieve this purpose: the point Y will never touch all of the points of the space because the number of points is an n-dimensional infinity.

Let the purpose of the observer be re-defined once more: information is to be collected about the system behaviour in such a way that it should become possible to predict the system behaviour by the use of this information at any point within the admissible domain.

This objective is reasonable. It is defined here as the objective of systems analysis.

To demonstrate one way of meeting this objective, let the verbal model of the source object be changed. Instead of the continuous n-dimensional space in which Y may move, consider now the same space in which a finite number of points N are strategically placed. The movement of Y will now be discontinuous, jumping from point to point. The total space may be covered by N number of changes in the position of the point Y. This model of the source object may be translated into numerical form: each of the N number of points can be given by a set of n numerals. The behaviour of the source object may now be given in form of a table which has one column for giving the instances of time when the position of Y changes, and n more columns describing each of the co-ordinates of Y. The observer will achieve his purpose by first covering all of the N points of the space and then, if necessary, interpolating between them to calculate the behaviour at an intermediate point. (see footnote).

The concept of an environmental model has found extensive use in Chapters 3 and 4 of this thesis.

2.4.5 - Modelling the state space.

As seen earlier (section 2.2.2) systems in general possess state. In the course of modelling the objects within the set D, it may have been decided to compile an analytic model of the system which would allow the description of the state in terms of m number of state variables. When the behaviour of the system is observed, the system may be anywhere in this m-dimensional state space and therefore the results can only be interpreted un-ambiguously if the initial state of the system is known.

Let the observer have a new and ambitious purpose of collecting information about the system behaviour in such a way that estimate could be made of the behaviour irrespective of the initial state of the system. This means that tests must be conducted to cover the total state space.

The task is impossible. At the outset, the moving point Y may be anywhere in an m-dimensional infinite number of points. Evidently it is necessary to quantise this area in the same way as in the case of the n-dimensional input space, permitting now a finite M number of points for the purposes of testing. Combining the

FOOTNOTE: This procedure will be shown as valid only for systems without state.

m-dimensional state space with its M points with the n-dimensional input space with its N points, a total space of $q = (n + m)$ dimensions may be perceived, consisting of both state variables and input variables, in which $Q = N \times M$ number of points are defined for the moving point Y.

The ingenious observer may succeed in constructing a source model in such a way that direct control is maintained over each of the n input parameters. The parameters of state space however are not usually directly controllable. Referring to the definition of controllability, it will be seen that systems which are not completely controllable can never be induced to occupy certain areas of the state space. Consequently no physical model of such a system will permit the systematic scanning of the total q-dimensional sample space.

One of the merits of the method of design verification proposed in this work is that it permits the designer the direct control of all variables of the sample space, as will be seen in chapter 4.

2.5 - SYSTEMS ANALYSIS.

Systems analysis will be defined as the process of obtaining information about the behaviour of the system in order to satisfy the objective set up in section 2.4.4. The purpose of analysis is to serve design verification, as defined in section 2.6.1.

The analysis process will be termed comprehensive if it gives deterministic information about the system behaviour at any point within the boundary of the admissible environment.

Fig. 2.5.1 shows a model of the systems analysis process. The model attempts to be general and therefore contains some parts which will be found irrelevant for certain modes of analysis. At the same time the model aims for simplicity and does not show all of the connections between parts of the model which may be required in the course of a given analysis process.

2.5.1 - Modes of analysis.

Depending upon the relationship of the real-world system and its environment to the test system and its environment, the analysis will be said to be conducted in one of several possible modes.

Five different modes of analysis are suggested and defined by Blumstein (2) as reported by Deutsch (3). This classification was found in-complete and the definitions somewhat ambiguous. Therefore, with the aid of Gordon's definition of simulation (4) six modes of analysis will be defined for the purpose of this work:

1) Real-world analysis - observation of the real-world system in its natural environment. In this case the real-world system is identical to the test system and the real-world environment acts as the test environment.
2) Operational exercise - observation of the real-world system in a test environment which has properties similar to the real-world environment.
3) Gaming - the test system is a hybrid, comprising a selection of real-world sub-systems and sub-system models; the test environment is similar to the real-world environment.
4) Simulation - observation over time of a model of a system in a simulated environment.
5) Analytical testing - solution of equations which represent the test system and the test environment in symbolic form.
6) Numerical testing - solution of num~~b~~erical model of the test system without direct reference to the time domain.

Deutsch (3) remarks that the modes of analysis are numbered so that higher-order modes re-present increasing distances from reality. While this is undoubtedly true, choosing one of the higher order modes of analysis does not necessarily imply loss of information about the system behaviour. On the contrary, examples will be found in the course of this work where high-order modes provide information not available by lower order modes of analysis.

It will be observed that the definition of simulation used here contains no reference to the tools of analysis: simulation may be performed on a physical model or on a computer model, so long as direct reference is maintained to the time domain. The time scale may be identical to that of the real system,or trivial mapping of inputs and outputs may be used.

2.5.2 - Input data.

Let the purpose of the designer be to analyse a system comprehensively. The question is now: how should the test data be designed to make such analysis possible.

The task is evidently impossible unless a quantised model of the source object is acceptable. Let it therefore be assumed that such a model has been found and that it contains Q number of points. The analysis will now need to investigate both the steady-state and transient behaviour of the system.

If the system may be assumed completely controllable and unconditionally stable then it is theoretically possible to test the steady-state response in each of the Q points. In addition, the transient response needs to be recorded by changing each of the Q points under the influence of all the input variables.

Designing a test sequence which would allow all these tests and which ~~is/for~~ suitable a system or its physical model is a task of extreme difficulty. Furthermore, the process would usually be found extravagant in terms of testing time and cost. Therefore, comprehensive analysis, even in terms of quantised source models, is seldom attempted. Instead, systems are either randomly tested (i.e. subjected to a random sequence of environmental changes) or, more frequently, tested in terms of a sequence of test data, which is judged to be of particular significance or relevance. The latter practice is dangerous since it is open to the value judgement of the designer whose work the analysis seeks to test (see also Chapter 3).

The test method proposed in the course of this work was designed with particular reference to the problems of test data generation.

2.5.3 - Input - output mapping.

Fig. 2.5.2 shows two ways in which the behaviour of a system may be observed. The direct method on the left corresponds to the lowest mode of analysis. All higher order modes demand that a model of the system under observation be available. In all these cases the real-world signals of input and output require suitable interpretation or mapping.

The mapping of inputs and outputs may be a trivial isomorphic transformation such as scaling, creating analogue signals in new physical dimensions or representing a signal as data in a computer and recovering the output by mapping of the printout. On the other hand, mapping may involve a transformation which changes the mathematical relationship between the systems attributes without affecting the system behaviour. One example is the change from time-domain to frequency domain for the analysis of electrical systems; the inverse transformation re-constitutes the outputs in the time domain. Another example, drawn from the context of logic network analysis, may be the derivation of a truth table or DIRECTORY by a parallel processing procedure (see chapter 4). In this case input variations are mapped into a designation number and the total input domain is covered by a single analysis run through the network. The output is mapped into a table. The result can then be interpreted in terms of time-varying input signals to which the response is available.

2.5.4 - Resolution, modelling and solution.

All but the two lowest-order modes of analysis call for some form of a system model. Before embarking upon a modelling procedure it is necessary to define the level to which the system may be resolved. This choice of resolution level determines the boundary around objects comprising the system.

In the course of analysis of a complex system it may be necessary or opportune to re -define the boundary several times. For instance, in the course of analysing a transistor amplifier, the circuit may, at first, be considered as a d.c. network, stationary at the operating point; then, in subsequent stages of analysis, the model of each component may be resolved to higher levels, increasing the information about the network.

It is also possible to envisage a process of analysis involving the gradual reduction of resolution level. Taking the example of logical networks, after detailed analysis of a non-linear network designed to act as a set of logical gates, it will be found advantageous to move to a lower level and consider the gate as the smallest object in a larger network; then again, after analysis, the large network may be regarded as the smallest object of the over-all system which was the original concern of the analysis process.

The purpose of reducing the resolution level is to reduce the modelling data. The new homomorphic model contains less information than the higher-level model, due to the fact that some simplifying assumptions have been accepted which permit the omission of a certain amount of detail.

Solution may consist of application of test data and observation of outputs, or application of test data and computation of outputs, or again, solution of equations and substitution of test data, In addition, some analysis procedures will be found to solve their models by observation and without any reference to test data. Due to the problems of test data generation discussed earlier in this chapter, these latter methods are particularly attractive and have been given attention in the course of this work.

2.6 - SYSTEMS DESIGN.

A model of the design process will now be proposed and discussed with reference to Fig. 2.6.1.

It will be assumed that some demand exists which must be satisfied by creating a new engineering system. A statement must be available which will specify the required behaviour and the admissible environment of the new system. This statement is called the tentative specification of the system.

The first task of the system designer will be termed problem analysis, which consists of the examination of the feasibility and completeness of the specifications. Specifications will be called non-feasible if they contain contradictary demands and incomplete if they do not describe the environment or behaviour uniquely. In the course of problem analysis the designer will initiate the amendment of non-feasible specifications. He will also investigate whether incompleteness is intentional or not. He will utilise the freedom afforded by intentional incompleteness at a later stage of the design process for the optimisation of some parameters not included in the specifications.

As an outcome of problem analysis the behaviour - and environmental specifications are formalised and the next stage of the design process may commence.

The designer's main function is to generate proposals for the new design. This task is usually so complex that it is carried out in a sequence of stages of refinements which represent successive approximations of the specification. Each stage of approximation operates upon a restricted version of the specifications (termed the partial specifications) which the designer must individually select for each stage.

Assuming now that suitable partial specifications are available, a version of the design must be proposed which will fulfil those specifications. At the present time the designer must rely almost entirely upon intuitive or evolutionary methods of design generation: formal synthesis procedures are only available in a very limited field of engineering and they only operate under severely restricted environmental conditions. Logic design is better served by synthesis techniques than other engineering fields; even so, current practice must rely to a large extent upon the inventiveness of the designer.

The proposal of the new design is a model of the system under development. This model must now be analysed and its behaviour assessed against the partial specifications. The combination of analysis and assessment will be termed verification or design testing.

The purpose of design verification is to provide conclusive answers to questions which are implicit in the partial specifications. A sample list of design verification questions is shown here:

1) Is the performance of the new design correct, given normal environment and nominal component values?
2) How sensitive is the performance to expected changes in environment and component values?
3) If subjected to unusual environmental conditions, would the new design fail catastrophically?

If verification detects design errors, these must be used to stimulate design modification. Alternatively, the error reports may be interpreted as indications of unrealistic tentative specifications. In the latter case the design process fails completely and new tentative specifications must be set up or the project cancelled.

Assuming now an error report which does not lead to design failure, the new design is gradually corrected (correction loop, Fig. 2.6.1) until verification succeeds. Now the next stage of design refinement may be entered and new partial specifications are requested by way of the refinement loop (Fig. 2.6.1).

The iterative design process is complete when all aspects of the formal specifications have been taken into account and verification has been successful.

CHAPTER 3

CONVENTIONAL METHODS OF TESTING LOGIC DESIGN.

3.1 - INTRODUCTION.

This chapter presents a survey of logic design verification methods in current use and aims to show that these methods fail to satisfy current demands of modern integrated circuit devices or complex logical systems.

The shortcomings of conventional methods are carefully noted. The newly proposed method, described in Chapter 4, seeks to eliminate or, at any rate, minimise the problems uncovered in the course of this survey.

With reference to the available repertoire of modes of analysis (section 2.5.1), analysis techniques of current use will be found to fall into the classes of Operational Exercise, Gaming, Simulation, Analytical Testing and Numerical Testing, with overwhelming emphasis on Simulation.

3.1.1 - Objective.

The purpose of logic network analysis is to serve the process of design verification. In turn, the purpose of design verification is to answer specific questions the designer raises about the new design (see 2.6). Consequently, the operational objectives of systems analysis can be formulated as follows:

a) to collect information about the performance of the new design,

b) to present this information in a form convenient for performance assessment,

c) to operate fast, cheaply and reliably.

The analysis techniques discussed in this chapter will be evaluated against these objectives.

3.2 - HARDWARE METHODS OF DESIGN TESTING.

3.2.1 - Mode of analysis.

The traditional way of verifying the design of a logical network is to build the total system of the proposed design in a form resembling the real-world system and subject this to a simulated environment. The data resulting from the analysis is then compared with the specification.

The technique in this form amounts to Operational Exercise and is frequently applied to small logical networks.

In the case of more complex networks it is customary to examine the performance of sub-networks before the system is finally assembled. The phase of sub-system testing is perhaps best classified as Gaming, because the real-world model of the sub-system under test is usually surrounded by models simulating the effect of the rest of the system.

If such systems are mass-produced, then the design verification uses a model which may have no object in common with any system on the production line. The mode of analysis is Simulation - the only mode of analysis appropriate to hardware testing of complex networks.

3.2.2 - Resolution level.

At the time of discrete-component hardware technology, designers had almost unlimited freedom of choice of resolution levels during analysis; it was possible to commence simulation by regarding the system as a whole; at the event of the first failure the design engineer could gradually elevate the resolution until the level of a single electrical component was reached. The procedure gave a sense of satisfaction to an engineer who could correct the design error on the spot; however, it was failing by all the analysis objectives, as would any other procedure which allows the handling of too many individual objects. The search for the location, appearance, connections and signals of several hundred error-prone individual components could not be condoned. Thus, well before the advent of integrated circuit technology, networks of modular and hierarchical structure were constructed, allowing no more than two different resolution levels for each test and no more than a few dozen individual objects and monitoring points.

Design testing of a hierarchical system proceeds in a sequence of stages. First, the design of the smallest module(s) is verified. If these are constructed of discrete components then it is possible to ascend to the level of single component.

When testing the design of a module at the next level of hierarchy then it is necessary to be able to assume all small modules as perfect (verified by the previous test) and indivisible. If this assumption can not be made then, in order to locate and correct the error, the resolution level must be increased, demolishing the boundary of the previously tested module. The difficulty and confusion can obviously be avoided by testing the small module reliably in the first place, as demanded by objective c). This demand for reliability becomes more pressing as the size of the system and the number of hierarchical levels increase.

3.2.3 - Modelling.

The analysis of logical networks by hardware simulation demands that a hardware model be constructed of the real-world system which will be subjected to time-variant signals. The modelling process itself appears to be a trivial exercise of constructing a physical model by use of components like those of the real-life system. In practice however the problems associated with modelling prove far from trivial: due to component tolerances and differences in physical layout, the correlation between the performance of the model and that of the real-world system is in doubt. Thus the information collected during analysis fails to represent reliably the performance of the real-world system, thus failing to satisfy the objectives of analysis.

Designers seek to remedy this situation in two ways:

1) by increasing the sample size, i.e. analysing several models, built of randomly selected components.
While improving reliability, the method increases the volume of data, the cost and time of analysis and raises the new problem of statistical assessment of results.

2) by evaluating the expected extreme in component values and selecting the components of the model so as to represent the "worst case" in system behaviour. The model thus permits the assignment of a boundary to the expected system behaviour.
If the estimate of "worst case" is correct then the method satisfies all the objectives listed; however, the resulting design is unnecessarily expensive.

This second solution is obviously only available in case of use of discrete-component hardware: users of integrated circuit hardware have very little information about the expected behaviour of components and no information at all about the extremes of acceptable performance. Even if this information should be available, the task of selecting worst-case items of multiple-function integrated circuit devices is formidable - worst-case assessment seems completely impractical.

Attempts have been made to build worst-case discrete-component models of integrated circuit devices. These have been proven unsuccessful due to the poor correlation between system and model behaviour. Some designers tried to construct a sophisticated discrete-component model, using time scaling to reconstruct the wavelength of signals in integrated circuit devices. This method is too complicated and the results are far too sensitive to scaling factors to be reliable.

The mounting problems of modelling integrated circuit devices must be viewed against the increasing demands for design verification: the tooling costs for a single component are the same as for the mass-produced article and, due to the magnitude of this cost, design errors can not be tolerated.

Hardware test methods offer no satisfactory solution to the modelling problem of integrated circuit devices. Reluctant designers are forced to seek computer solutions to their problems.

3.2.4 - Inputs.

When a hardware model is constructed, the designer subjects this to simulation, observes the output and verifies the design. If the system has n operating signals, m physical signals and q state variables then the sample space has an (n + m + q) - dimensional infinite number of points (see sections 2.4.4 and 2.4.5).

To attempt comprehensive analysis, it is necessary to devise some suitable discrete model of this sample space. However, unless the system is known to be completely controllable (see Chapter 2) it is not possible to scan the total sample space in a finite amount of time. In the case of completely controllable systems the problem is theoretically accessible but practically unsupportable, since the number of input changes necessary for comprehensive steady-state and transient testing is excessive for all but the most trivial of systems.

In current practice the job of design verification is conducted at a few points of the sample space, under conditions the designer considers representative or critical. If the design fails these tests then it is evidently in need of modification. If it passes the tests then it may still contain numerous errors and may fail in service. Design errors in computers often come to light years after machines are installed, indicating that a compromise has been selected between the conflicting objectives for speedy, low-cost and reliable analysis.

It is suggested here that the problems arising from the vastness of the number of points of the sample space are akin to those arising from the vastness of the number of components comprising a complex system. The solution of the sample-space problem might be sought along the same lines as that of hardware: it could be decided that there is a maximum number of points of sample space one can efficiently handle, therefore models of inputs must be so constructed as to fit within this number. Input signals may thus be "modularly" modelled and as the number of signals increases so the resolution must reduce to keep down the number of points of the sample space.

According to this principle, it will be reasonable to consider the effects of physical signals, rise-and-fall times and signal level variations when testing a single logical gate, but unreasonable to resolve input signals to more than two voltage levels when testing a complex gate assembly. In the latter case it would be necessary to form a model of the objects of the assembly which accounts for the effects of the neglected signals.

Unfortunately for logic designers, the number of points of sample space increases very fast with the number of input variables and the reduction of detail of input signals can not keep pace with this increase. Thus the computer simulation methods discussed later in this chapter merely manage to provide a temporary solution by increasing the speed of simulation. Ultimately the failure of simulation as a means of analysis must be faced.

3.2.5 - Solution and assessment.

Let it now be assumed that a satisfactory set of test conditions has been compiled and the model is subjected to these tests. The result of the analysis is available in the form of a set of output waveforms. These waveforms must now be observed or recorded and assessed against some form of records of the performance specifications.

The traditional instrument of recording, offering the advantage of familiarity, is the oscilloscope. Against this must be set numerous disadvantages: the number of channels is limited, thus, to observe the correlation of several waveforms, tests must be repeated, or more than one oscilloscope used; signals must be repetitively applied to permit observation; the standard signals can not be displayed - the real-world system does not exist - hence the oscillogram must be compared with a waveform or table by eye, or else recorded as a waveform or table.

The instrument of assessment is the design engineer who often represents the weakest link in the chain of the verification process. Human error is mainly due to the repetitiveness of the task of assessment and the inefficiency of men to handle the large volumes of data necessary for reliable analysis.

3.2.6 - Conclusions.

Hardware testing of the design of logical systems is, at best, limited to small systems; at worst it is an unqualified failure. The method fails completely in terms of complex integrated circuit devices which, due to their high initial manufacturing costs, demand reliable design verification. As advances in technology permit the increase of the complexity of these devices and enhanced their prominence among other forms of hardware, the development of alternative forms of design testing become imperative. Thus, almost simultaneously, several computer simulators have been developed by device manufacturers, instrument manufacturers and research institutions, seeking to provide a solution to the problems unsolved by hardware testing.

3.3 - DESIGN TESTING BY COMPUTER SIMULATION.

The preparation of a comparative survey of logic simulators does not fall within the scope of this work. Such a survey, examining the features, facilities, techniques and relative merits of available simulators, will be presented in the thesis concerning the development of a prototype system (1). Instead, this section will attempt to assess the potential of logic simulation as a tool of logic design verification. In the course of discussion references will be made to some of the logic simulators in current use, illustrating some of their features and facilities.

3.3.1 - Resolution.

At the highest level of resolution, used by any of the known logic simulators, stands the single logical gate. It is thus assumed that the system under test consists of objects which represent Boolean operators and the simplest functional object to which the system can be divided operates as a simple logical connective. The standard repertoire of gate functions is AND, OR, NOT, with associated delays (see Modelling). Some simulators extend the range of gate functions to NAND and NOR.

A group of simulators, such as the APACE program 'LOCA', the Plessey DA70 and the Siemens 'DICAP', demand that systems under test be always resolved to such a high level; thus designers must code their networks by use of a fixed library of standard elements.

The advantage of systems using this fixed resolution level lies in the simplicity of the programs. The disadvantages are measurable in terms of coding time and coding error: the designer must repeatedly declare each gate within each standard hardware sub-system, multiplying the coding time and committing a multiplicity of errors. An additional disadvantage is, that the computer must repeatedly analyse standard hardware sub-systems within the system. Thus it is suggested that simulators with fixed resolution level are only suitable for the analysis of small systems.

A second group of programs to which the Norwegian program "LOGIC" and the Elliott "LASS" belong, also fixes the resolution level but permits an expandable library of standard objects. When the computer model of the network is constructed, each object is checked against the current library list. Unrecognised objects are reported as program errors. Standard groups of gate elements can be entered in the library as standard objects by giving them a unique name and describing them by a sub-routine, (21), (22). The use of this method is demonstrated in (21), where flipflops are "modelled" and their functional description is entered in the library as a standard "logic rule".

These facilities, as provided in 'LOGIC', replace the whole of a sub-system by a sub-routine. The sub-routine represents a terminal model of the perfect sub-system; consequently care must be exercised in verifying the design of a sub-system before replacing it by the sub-routine. If reliable sub-system testing may be assumed then this method satisfies the objectives by reducing both the time of coding and the time of computation. The limitations of such sub-routine models of sub-systems will be discussed in the section dealing with Modelling.

The third group of programs to which the Racal "REDAP 22" and the Fairchild "FAIRSIM II" belong, permit a flexible choice of resolution levels. It is possible to construct sub-systems of nested modules and the system under analysis may be composed of combinations of single gate objects or of complex nested sub-systems. It is customary to impose a limit to the depth of resolution.

This method gives great help at the stage of modelling, but it in no way assists the analysis by the computer since, before analysis commences, the modular structure is usually broken down to constituent gate elements and the analysis progresses from gate to gate.

3.3.2 - Modelling.

Simulators differ greatly in the format they use in specifying the objects of the system under test. However, substantially they all refer to four types of data, amounting to the model of an object:

1) object identification
2) structure - the connection of the object to other objects of the system
3) functional description.
4) timing description.

Identification.

Each object is uniquely identified by a code. Information about the object is referenced by this code throughout simulation.

Structure.

Connection between system objects is specified in one of three ways:

a) considering the object as a recipient of signals
b) considering the object as a source of signals.
c) specifying all the connections to and from a given object. This method duplicates the structure information, thus doubling the coding

time. The redundancy is then used for diagnosis of coding errors.

Function.

The functional description assigns to the object one of a set of recognised parameters. Dependent upon the resolution as seen earlier, functions may be simple logical connectives, combinations of simple logical connectives or functions outside of the range of logical functions. The latter may arise by the use of sub-routine models.

It will be appropriate to devote some attention to the principles upon which sub-routine models are based.

Let a sub-system to be modelled consist of a set of logical gates, each representing a simple logical connective. Let this system be subjected to an n-dimensional environment and let it have m number of state variables. Then the terminal model of this system would contain the relationships of

$$\left.\begin{aligned} z(t) &= G\left(u(t),\ S(t_o) \right) \quad \text{and} \\ s(t) &= H\left(u(t),\ S(t_o) \right) \end{aligned}\right\} \quad \text{Equation 3.3.1}$$

where $u = \left(U_1, U_2, \ldots U_n \right)$,

$$s = \left(S_1, S_2, \ldots S_m \right)$$

(see section 2.2.2).

It will now be suggested that sub-system models can be constructed in one of two ways:

a) by retaining the information contained in Eq. 3.3.1; in this case the model may be mapped into a logical network, containing a set of logical gates, each representing a simple logical connective. This new network is equivalent to the original sub-system by the definition of equivalence given in Chapter 2.

b) by reducing the information in Eq. 3.3.1, restricting either the domain of the environment or the admissible time functions of inputs. Using such restrictions Eq. 3.3.1 degenerates to some new form

$$\left.\begin{aligned} z^{*}(t) &= G^{*}\ (u\ (t),\ S\ (t_o)\ ) \\ \$^{*}(t) &= H^{*}\ (u\ (t),\ S\ (t_o)\ ) \end{aligned}\right\}\ \text{Eq. 3.3.2}$$

The sub-routine may now express Eq. 3.3.2 in a form which does not necessarily lend itself to mapping into a logical network. Equation 3.3.2 and its sub-routine version will be termed <u>restricted mode models</u> of the sub-system.

Restricted mode models represent a homomorphic transformation between the systems described by Equations 3.3.1 and 3.3.2. If the sub-system should be subjected to environmental conditions outside of the range of Equation 3.3.2 then the sub-routine model fails. Such in-admissible conditions may easily arise if, for instance, the objects surrounding the sub-system contain some design error.

It will be concluded that notwithstanding the commendation given to sub-routine models in the previous section, such models must be handled with extreme care. Not only must the sub-routine be based upon completely reliable analysis data (as contained in Eq. 3.3.1) but the environment must also be kept under constant observation to ensure the validity of the model.

<u>Timing</u>.

The timing description of a system object seeks to define the mapping of time functions performed by the object. While in practice this mapping is extremely complicated due to the nonlinear characteristics of switching circuits, these characteristics could only be observed by resolving the system to higher levels than a logical gate. Alternatively, accepting the single gate as the object at the highest level, as suggested in section 3.3.1, one must accept relatively crude models of objects, such as, for instance, a perfect (delay-free) logical gate, followed by a lumped constant delay.

Some simulators offer facilities for more refined models. The fundamental weakness of these models lies in the absence of reliable information upon which they are based. Device manufacturers are unable to measure the characteristics of some of their devices, or find it uneconomic to measure the characteristics of others. In any case, these characteristics would vary so greatly with the choice of hardware that a common set of parameters may not be easy to find. Hence simulation programs must rely upon crude guesses of timing information and the accuracy of the analysis will not be improved by the use of sophisticated models whose parameters are based on further crude guesses.

The most commonly used time model is, as mentioned, a propagation delay t_p, associated with each object output. Some simulators permit the use of two time parameters t_{pr} and t_{pf}, representing the time delay associated with rise and fall, respectively.

A more sophisticated time model adds a further threshold delay t_t which demands that the input signal pulse should be ignored unless the duration of the pulse exceeds t_t.

It is reported that an industrial research group is seeking to develop a time model which would incorporate non-constant time parameters. Dependent upon the amount of load supplied by an object output, different values of t_p and t_t would now be assigned to the same system object. This project suffers from the same lack of reliable time data as has been mentioned before.

A far more ambitious project is reported (29) by members of the IBM corporation . The work of this group concerns the development of a statistical delay model for a simple logical network. The model correlates the variation of some physical signals and of load with the time delay of gates. Such is the complexity of this problem that it was necessary to construct a purpose-built computer system to carry out data collection and processing, while calculations were performed on an IBM 7044. In spite of the extravagant resources used by the project, the results have very limited practical application.

The uncertainty of time data emerges as a fundamental shortcoming of simulation as a practical mode of analysis of logical networks. By the definition of simulation, verification of designs is based upon observing the system response in the time domain, but this response is computed on the basis of crude models whose parameters are derived by wild guessing.

3.3.3 - Input data.

Fixing the highest resolution level at gate level implies that operating signals should be modelled by discontinuous jumps between two logical levels. The specification of these operating signals consists of the listing of those instances of time when a given operating signal is scheduled to change. As an aid to coding of this data, some simulators permit the use of oscillators in addition to switches. The oscillators provide periodically changing signals whose ON and OFF time may be specified.

Physical signals (supply potentials, temperature) are considered as constant and, as mentioned in the previous section, their effect upon the time delays associated with object outputs is neglected.

The sample space is thus restricted to dimensions given by the operating signals and state variables. Even so, this space is excessively large for all but trivial networks. The number of necessary steady-state and transient tests will be assessed in chapter 4. It will be sufficient to mention a few problems arising at coding.

The simulation commences by the specification of an initial state. In the absence of such specification simulators assume that all state variables are to be set at, say, logical '0'. If this condition is logically inconsistent (unstable) then a series of calculations commences while the simulator searches for a steady state of the system. This may not always be found, or not found within the time limit set by the designer. In such a case the simulator refuses the network (as in the case of the RACAL program) or over-writes a transient state by permitting the operating signal sequence to begin (such as in case of the LOGIC program). Neither of these solutions is particularly attractive: one refuses to simulate the system altogether while the other may give ambiguous results depending upon a random choice between oscillatory states.

If the initial state selected by the designer or set up by the program is not connected to some of the system states then no finite time sequence of operating signals would result in a comprehensive analysis.

Oscillator-driven inputs imply a restricted mode of analysis. While it is possible to construct oscillator signals which permit the system to be tested for all of the 2^n number of different input signal combinations for n inputs, this number of tests subjects the system to only a small fraction of the total number of different transient conditions and will only ascertain the response in the case of the selected initial state. The solution is sought by selecting sequences of operating signal changes which are regarded by the designer as most probable to occur. For instance, it is often found that networks are simulated under conditions when an output is scheduled to generate a specific signal. If the network passes the test, then the designer assumes that the design is correct. This assumption is fallatious because it is based on the same value judgement as was used by the designer when proposing the design. The network may behave incorrectly by generating outputs at untested conditions.

An alternative method is to select input sequences entirely at random. This way there is a probability of selecting conditions which might have been overlooked at the design stage. Even so, it is very difficult to assign a significance level to a selected test sequence and the reliability of the analysis is in doubt.

The problems discussed here are familiar from experience with hardware testing and there is no evidence that the newly developed techniques of computer simulation offer a satisfactory solution.

3.3.4 - Input - Output Mapping.

The process of computer simulation of logical networks consists of specifying the changes of input signals in the time domain and following the passage of signal changes through the network. The procedure demands the choice of time quanta at the input and throughout the operation of the network.

Synchronous simulation refers to a system in which time is incremented by equal intervals and the model is processed at each time increment. Inputs and outputs are shown on a linear time scale, with a line of print assigned to each increment.

Asynchronous simulation is frequently termed "next event" or "event-by-event" simulation, indicating that time is incremented by periods measuring the distance between subsequent events, either at the input or within the network.

The printout of event-by-event simulators show input and output waveforms on a non-linear time scale, with a line of print assigned to each event, against a parameter of time.

Asynchronous simulators offer flexibility; they lend themselves to the analysis of synchronous or asynchronous networks. Most of the known simulators are asynchronous.

Synchronous simulators are obviously suited to synchronous networks. When used for the analysis of asynchronous networks the time increment must be adjusted to the smallest common unit of all delay elements and of all input events. The method is considered to be of limited use.

Notable examples of synchronous simulators are the GEC and Plessey systems.

3.3.5 - Solution and Assessment.

Solution consists of two phases: calculation of the logical state of each monitoring point (system output or output of selected objects) at each of a sequence of instances, and display of the logical state of inputs and outputs at those instances.

Assessment presents problems familiar from the discussions of hardware test methods. Standard signals need to be displayed in a form comparable to the simulator output. Such standards may not be easily available. The instrument of assessment is, once again, the designer who is now faced with volumes of data unheard-of at the time of hardware testing, as a direct consequence of the speed and facility of producing data by computer simulation.

The columns of logical '1's and '0's of the printout contain inconclusive answers to the questions of verification in implicit form. The increased volume of data makes the designer's task of assessment and data processing more formidable than was the case with hardware testing. This problem of assessment may be attributed to the fact that communication between man and machine needs to be established at the point where the volume of information is the greatest.

In terms of the design verification questionsof section 2.6, the method will be seen to have limited use, allowing no answers to any but the first question. This is a direct consequence of the method of modelling system objects and of the constraints applied to the environment.

It may be concluded that conventional methods of computer simulation fail to provide the answer to the problems of design testing of modern logic networks.

3.4 - DESIGN TESTING BY HIGHER-ORDER MODES OF ANALYSIS.

The literature contains a great variety of well-established and some recently-proposed methods of analysing logical systems by use of Boolean equations (analytical testing) or tables (perhaps best classified as numerical testing). Such methods range from tests of steady-state response to diagnosis of ambiguous transient behaviour. The basis of analysis is almost invariably a Boolean model and signals are assumed as discontinuous jumps between one logical level and the other, as in the case of computer simulation.

The review of these methods led to the following conclusions:

1) it is possible to correlate the results of some existing methods with some of the design verification questions designers may wish to ask.
2) it is humanly impossible to apply these methods to all but the most trivial networks without the use of computers
3) it is necessary to extend or modify some of the accepted methods as well as to propose some completely new methods if answer is sought to a reasonably comprehensive range of design verification questions.

Subsequent chapters of this thesis will describe the way in which these higher-order modes of analysis may be used in the interest of logic design verification.

CHAPTER 4

A MODULAR METHOD OF TESTING LOGIC DESIGN

4.1 - INTRODUCTION.

Chapter 3 shows the inadequacy of conventional design verification techniques as applied to logical systems of modern size, construction and complexity. This inadequacy can be attributed to a number of causes:-

1) The difficulty of man-machine communication; this problem arises at the specification of the system, of the test data and, especially, at handling the data produced by the analysis.

2) The excessive size of the sample space of systems with modern dimensions.

3) The increasing demands of reliability of designs.

4) The reduction of detail and reliability of information about individual logical elements of the system.

5) The conflict between the complexity, reliability and accuracy of delay models.

4.1.1. - The modular system of design verification.

In view of these considerations it was decided to break with conventional test methods of simulation. These methods relied upon a prolonged general-purpose analysis process whose output data was the basis of design verification. By contrast, the new method consists of a sequence of special-purpose verification procedures, each answering a specific question about the newly proposed design. For this purpose a compatible set of design verification modules is constructed. If the designer now produces a list of questions about his newly proposed system, he can select the appropriate sub-set of off-the-shelf modules which will provide the answers.

The modules must be so designed as to ease the man-machine communication problem. Thus they should operate on the minimum of information and, instead of producing analysis data, must produce direct answers to the design verification questions. Such answers may condone the proposed design or list its errors in a format which facilitates design correction.

Test data for the modules must, wherever possible, be automatically generated. This overcomes the communication problem and improves reliability because the automatic test data generators scan the sample space systematically.

The quality of system model must be scrupulously preserved. Using the minimum of data for the description of the system eases communication, minimises the demands of computer storage and computing time. In addition, detailed models demand the specification of delay parameters whose reliability is doubtful. Thus the design verification procedure must restrict the use of delay models as far as possible, in the interest of reliability.

The modular system appears to have the potential to overcome most of the inadequacies of conventional testing. It offers the additional features of flexibility and ease of development. The system may continually be improved by addition of new modules, re-design of existing ones or deletion of obsolete ones, without putting it out of service. With the aid of a carefully specified interface, several programmers may be engaged on the development of the system without the need to refer to each other's decisions on details.

The subject of this chapter is to describe the design of the modules of such a design verification system.

4.1.2 - The choice of verification questions.

The idea of such a design verification system is thought to be novel. This means that no previous experience was available upon which to build such a system.

The first difficulty was encountered when trying to anticipate the questions designers may wish to ask. At the present time designers do not formalise their questions but deduce some answers from analysis. Formalised questions demand disciplined thinking and a measure of formal education in logic design; one or both appears frequently to be lacking.

The questions proposed in this chapter become gradually more sophisticated, matching the sophistication of the facilities they provide (not necessarily the techniques they use; some of the most complex questions are answered in exceedingly simple ways). This is claimed as an advantage; simple modules will give designers useful information as well as a gentle introduction to computer-aided design.

It is further considered that the use by designers of some sophisticated concepts (such as a VERIFIED DIRECTORY) are inevitable. Logic systems have become so complex that they are not approachable by simple test methods (see chapter 3); the concepts used in this work are not always simple, but are presented to the designer in a form which is thought to be easily accessible. Experience in this field is available through the Prototype System (chapter 5) which has been found easy to use by several teams of logic designers.

It is anticipated that the proposed design verification questions may need modification in the light of experience with the system in service. Such modification is facilitated by the modular structure as mentioned before.

4.1.3 - Input - Output.

It was possible to use some of the experience gained from logic simulation in the choice of input format. However, since the service provided by the method is new, the output format was designed without background. The Prototype System gave some help here, but, again, extensive use by designers on large and complex systems is needed before the output format may be considered finalised.

As mentioned earlier, the output data is, in almost all cases, an error list. The input data consists of the following items:-

a) Specification of new design.
b) Specification of operating signals.
c) Specification of behaviour.
d) Test instructions.

These will now be discussed briefly.

a) Specification of new design.

It was decided to describe a logical network by four items of data:

i) structure

ii) function; parameter of objects

iii) load parameter of objects

iv) delay parameter of objects.

The choice of these items will be discussed and justified in the course of this chapter.

b) Specification of operating signals.

In the interest of reliability it was decided to cover automatically the total sample space of a given test whenever possible. If this space proved infinitely large, or too large to be completely covered, then instructions were required about the manner in which the space could be restricted. In only a few circumstances in the course of this chapter was it found necessary to compel the designer to specify the wave-forms of input signals.

c) Specification of behaviour.

It is considered that such specifications initiate the whole of the design process, hence they must always exist in some form. It is understood that usually the specifications are not formalised and are often unrealistic or unreasonable. Attention was given to allow the presentation of specifications in a variety of formats. In some instances it was also suggested that the design verification methods could be usefully employed in the analysis of the specifications themselves.

d) The test instructions consist of nominating the verification modules for a given test.

4.1.4 - Analysis and Assessment.

Use was made of accepted and generally used concepts of logic network analysis. Such concepts include truth tables, state and output tables, the definition of essential hazards, etc. These concepts, in almost all cases, needed extension or modification before they could be used as bases for test techniques.

Frequently it was found necessary to design entirely new concepts. Thus, inevitably, a jargon was created, which was based on general systems usage. New terms were only introduced if they helped to make discussions or definitions more concise or clear.

4.1.5 - Constraints.

While care was taken to compose questions concerning a variety of aspects of the design, it can not be claimed that the list of questions is comprehensive. Nor are the answers completely satisfactory in every case; such is the complexity of the problems that the validity of answers had to be made conditional upon some simplifying assumptions. However, it is trusted that the techniques used for the generation of answers may be perfected along such lines as indicated in the chapter on FURTHER DEVELOPMENTS, allowing the removal of some of the restrictions. Further, these techniques, or the experience gained by their development and use, would prove valuable in finding answers to verification questions not listed in this chapter.

In some cases it was found that a given question may be answered in alternative ways and each alternative contained some interest; in such cases the alternatives are described here.

Some of the questions were found convenient to be answered in terms of answers to others; in such cases a test sequence is obviously implied. Otherwise the sequence of conducting the tests is not pre-determined.

4.1.6 - Organisation.

This chapter will be organised by devoting a section to each design verification question. Within a section relevant concepts and techniques will be discussed and at least one test module suggested.

4.2 - IS THE STATE OF THE SYSTEM FULLY DETERMINED BY THE STATE VARIABLES NOMINATED?

The question tests the designer's understanding of the system. The assignment of state variables is, in some cases, a trivial exercise, such as associating a state variable with each S - R flipflop. In other instances the structure of the network is complicated by feedback loops linking a large number of logical gates and in the course of evolutionary design the designer may be unaware of the presence and consequences of a group of system states. In such cases designers should check if they understand the system correctly by nominating a comprehensive set of state variables which, in their view, should be sufficient to describe the states of the system. A test module should then check if the designer's estimate was correct.

When developing the necessary techniques for answering the question it was found sufficient to use structural information about the system without reference either to test data or to models of objects of the system.

4.2.1 - Concepts and techniques.

A combinational network will, for the moment, be defined as a structure containing no feedback loops at the selected level of resolution. Conversely, a sequential network will contain feedback and the number of state variables in terms of which the network is described depends upon the number of feedback loops (see footnote). In order to describe a sequential network in terms of inputs and state variables, each of the feedback loops must be broken once and a state variable inserted. The place at which the loop is broken is arbitrary, hence there is no unique way of describing a sequential

---

<u>FOOTNOTE</u>: These definitions are consistent with the concept of observability (chapter 2); if a network containing feedback does not display a terminal behaviour customary for sequential networks (such as the strange network of ref (30) ..) it will be classed as sequential on the basis of its structure, although the sequential behaviour is not observable. A sequential network with no observable states may, under these definitions, be equivalent to a combinational

(continued overleaf)

network in terms of state variables. All comprehensive descriptions (i.e. ones which leave no feedback loop intact) are acceptable to the test system but, as will be discussed later, those which describe the network in terms of the minimum number of state variables are to be preferred.

The technique of analysis suggested here accepts the designer's choice of a set of state variables but rejects the design if this set is not comprehensive. The error message generated by the test warns the designer that he has failed to break all of the feedback loops. Should the designer be unwise and break any of the loops in more places than one, the design is accepted by this verification step as correct. If such a system is subjected to verification by other test modules then the analysis would be performed correctly but inefficiently, by describing the state space of the system in more than the minimum number of dimensions. This inefficiency implies an imperfection of the module to be described in this section. An alternative module, free of this imperfection, will be presented in a subsequent section.

The test techniques are presented in terms of a simple example shown on Fig.4.2.1. A unique identifier has been assigned to each object of that network. Environmental signals are restricted to operating signals which are generated by source objects 10, 20 and 30. The structure matrix, as defined in Chapter 2, consists of 10 x 10 = 100 elements:

FOOTNOTE (Cont'd):

network. Following this argument to its conclusion the concept of a combinational network may be eliminated altogether by defining all networks as sequential but recognising that a network may not have observable states at a given level of resolution. Increasing the level of resolution may then reveal the existence of feedback loops.

For the purposes of this work the concept of a combinational network will be retained under the above definition.

$$W = \begin{pmatrix} W_{10,10}, & W_{10,20}, & \ldots & W_{10,6}, & W_{10,7} \\ \vdots & & & & \\ W_{7,10}, & W_{7,20}, & \ldots & W_{7,6}, & W_{7,7} \end{pmatrix}$$

Assume now that the resolution level has been chosen at the level of a single logical gate, representing a simple logical connective. Selecting one of the gates of the network at random, the matrix elements concerning the selected gate can be computed noting that all gates have a single output and gate inputs are numbered as shown on Fig. 4.2.1. Thus, collecting all the matrix elements referring to gate 6, say:

$$\left.\begin{array}{l} W_{10,6} = (0) \; ; \quad W_{20,6} = W_{30,6} = W_{4,6} = (0) \; ; \\ W_{1,6} = (0,0); \quad W_{2,6} = W_{3,6} = W_{5,6} = (0,0) \; ; \\ W_{6,6} = (0,0,0) \; ; \qquad W_{7,6} = (0,1) \end{array}\right\} \text{From '6'}$$

$$\text{To '6'} \left(\begin{array}{l} W_{6,10} = (1,0,0) \; ; \; W_{6,4} = (0,1,0) \; ; \; W_{6,30} = (0,0,1) \; ; \\ W_{6,20} = W_{6,1} = W_{6,2} = W_{6,3} = W_{6,5} = W_{6,6} = W_{6,7} = (0,0,0) \end{array}\right.$$

The matrix is unwieldy and contains a great deal of redundancy. Since '6' is known to have only three inputs, the group of matrix elements marked "To '6'" can only contain three positive indications of connection. Thus, of the 30 bits of information describing connections "to '6' " only three can be relevant. Collecting these in a single matrix of W_6, and indicating the identifiers of circuit objects connecting to the input of 6 as elements of W_6:

$$W_6 = (10, \quad 4, \quad 30).$$

If the network is constructed of simple gate elements representing an elementary Boolean connective then the functional significance of all inputs is the same and the gate output is not altered if inputs are interchanged. This circumstance allows the specification of W_6 be given in terms of 10, 4 and 30 listed in any order, or, more conveniently, in terms of a connection matrix of a single row and with a single-bit column assigned to each of the objects of the network:

$$W_6 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \multicolumn{1}{c}{10} & \multicolumn{1}{c}{20} & \multicolumn{1}{c}{30} & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} & \multicolumn{1}{c}{5} & \multicolumn{1}{c}{6} & \multicolumn{1}{c}{7} \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline \end{array}$$

In this simplified form the total structure matrix will be a square matrix of 100 binary elements as shown on Fig. 4.2.2. The columns represent the signal sources and the rows the loads.

The structure matrix has certain regular features: the rows of source objects, by definition, contain only zero elements and in accordance with logic design practice forbidding the feedback from the output of a gate element to its own input, the diagonal of the matrix is zero. (see footnote)

The structure matrix may be reduced, without loss of information, to the form shown on Fig. 4.2.3.

The reduction of the structure matrix of chapter 2 to the form of Fig. 4.2.2 is possible for two reasons:

1) all the inputs of a network object have been assumed to be interchangeable, thus the identification of inputs became unnecessary.

2) all network objects have been tacitly assumed to have a single output, thus the identification of outputs became superfluous.

FOOTNOTE: If this latter condition is not satisfied, the test reports an error of an unspecified state variable. If such a feedback loop is intentional then the programmer must introduce a dummy element, as described later, in the loop. Thus no element feeds back upon itself and the diagonal is zero.

These assumptions are valid only as long as networks are constructed of standard objects (AND, OR, NAND, NOR, STATE VARIABLE, INPUT, OUTPUT). As soon as a more sophisticated sub-network becomes an object of a larger network, neither of the two assumptions can be taken for granted. For instance, the simple sub-network of Fig. 4.2.4 has two outputs which must be uniquely identified. Should any of the inputs be interchanged, the resulting functions would be altered unacceptably.

The problem is eliminated if a sub-network is resolved at a level high enough to permit the identification of each standard object. In this case the network is termed to be a <u>homogeneous structure</u> which can be analysed in terms of a matrix such as that of Fig. 4.2.2.

In the forthcoming discussions of section 4.2.1 it will be assumed that the designer presents the network for verification at the resolution level of standard objects (single gates), or that the system is presented at some lower level to a translator which generates a homogeneous structure for the purpose of this analysis.

It will be noted from Fig. 4.2.1 that the network is combinational, containing no feedback loops. Object outputs may be computed without reference to state variables, and a sequence of computation may be found which allows the determination of all element outputs.

The matrix of Fig. 4.2.3 is divided to a "known" and an "unknown" area by a vertical dotted line. The columns of the known area contain purely source variables.

Those rows of the matrix which contain '1's in the known area only are "computable" and can be entered in a "processing list". Their column can then be transferred to the "known" area and the next computable variable sought. The process is continued until the "unknown" area is depleted and the processing list is complete. It is frequently possible to locate more than one computable variable at any one time, hence more than one valid processing list exists for a given network. One of the valid processing lists for the network of Fig. 4.2.1 will be 1, 2, 4, 6, 3, 5, 7.

The search fails to result in a complete processing list if the network contains an unspecified state variable. Demonstrating this with respect to a modified version of the network of Fig. 4.2.1, let a feedback connection be added from the output of gate 7 to the input of gate 4. The structure matrix of the new version of the network is shown on Fig. 4.2.5.

The processing list will contain objects 1, 2, 3 and 5, leaving objects 4, 6 and 7 locked in a loop, each depending upon one of the others (Fig. 4.2.6).

If the designer is aware of the existence of the loop then he must insert a dummy object at an arbitrarily selected point within the loop when preparing the data for the computer analysis of the network. The dummy object has the function of a state variable. For the purpose of the processing list a state variable has the nature of a source object and falls in the "known" area of the structure matrix.

The sequential network and its dummy object generating the state variable, are shown on Fig. 4.2.7. The structure matrix is increased by a column (Fig. 4.2.8) and the processing list may be, for instance: 1, 2, 4, 6, 3, 5, 7, 71.

4.2.2. - The test module A.

The model of design verification, as presented in chapter 2, demands that the proposed design be analysed with reference to environmental conditions, and assessed against the behaviour specifications.

In this case the proposed design is offered to the test module in the form of a homogeneous structure of multiple-input, single-output objects. The format of this data may be a list of declarations. Each declaration is headed by an object identifier and contains the list of those object identifiers to which the inputs of the declared object are connected. Each declaration will therefore specify a row of the connection matrix and there will be as many declarations as there are objects in the network. To permit the identification of source objects, these must also be declared; their identifiers will head an empty list since their inputs are connected nowhere.

The proposed design of the network of Fig. 4.2.1 would be offered for verification in the form shown on Table 4.2.1. Both the order of the declarations and the order in which objects are listed within the declarations are arbitrary.

The test may commence without reference to environmental conditions; the logical functions generated by the source objects have no relevance to a test based entirely upon structure. The analysis consists of manipulations as described in section 4.2.1.

The expected behaviour of the network under this test is that a complete processing list is produced. Therefore, the behaviour of a tested design is checked against the specification of a processing list containing as many elements as objects of the set D (chapter 2). It is not necessary to specify the desired behaviour separately; the objects of the set D may be counted at the time of the declaration of the new design. The outcome of the design verification test is a binary decision. In case of the example the processing list contains 7 elements as does the set D. The designer was right in declaring no state variable.

For further reference this test module is given the alphabetic character A. The inputs and outputs of the module are shown in Table 4.2.2. The processing list is an optional output in which the designer will have no direct interest. It is however valuable for subsequent analysis modules.

4.3 - WHICH IS THE MINIMAL SET OF STATE VARIABLES OF THE SYSTEM?

This question is an alternative to that of 4.2. It may arise simply because the designer finds the part of data preparation which concerns the nomination of state variables tedious and inconvenient; the design of a network will often be spread over a number of logic diagrams, and feedback loops linking large areas of network will be difficult to find. Module A does locate the designer's failure to nominate a comprehensive set of state variables, but offers no help in locating the unbroken feedback loop. Nor does Module A give any indication if the designer errs the other way by overestimating the number of state variables. As indicated before, the sample space increases very rapidly with the number of state variables and the subsequent analysis becomes inefficient; of all the tests conducted only a fraction will be independent and the rest will contain no information.

The answer to question 4.3 facilitates the coding of networks and, at the same time, prepares the way for subsequent analysis.

4.3.1 - Techniques.

It will now be assumed that the designer does not specify state variables but describes the connection to the inputs of each object from which a homogeneous data structure is prepared. Linking the output of an object to its own input is now permissible. It is further assumed, as in section 4.2, that the designer presents the network at the resolution level of single gates or that a translator prepares a homogeneous structure previous to this analysis.

The reduced structure matrix is prepared in the usual way and elements of the processing list identified until no further computable variable can be located. The rows of un-computed variables and the columns of the "unknown" part of the reduced structure matrix form a "residue matrix" which shows the gates locked in, together with the interconnections between these gates. This residue matrix is always square.

As an example, the residue matrix of the network of Fig 4.2.1, with the output of gate 7 linked to the input of gate 4, is shown in Fig. 4.3.1.

A method is now sought by which the feedback links of a network may be identified automatically and in such a way that the number of state variables assigned to the network should be the minimum.

With reference to the residual structure matrix, two questions arise:-

1) Which is the smallest set of column variables to be assumed, to allow the completion of the processing list?

2) How should the network be modified in such a way that a state variable be associated with each column variable?

The second answer is easily found; the state variable should be generated by a dummy element which is fed from the assumed column variable and which feeds all the other gates associated with that column variable.

The first question is more difficult to answer. The method proposed here leads to the definition of a set of parameters which influence the choice of members of a minimal set of state variables.

The subject will be discussed in terms of specific examples which have been designed or selected to demonstrate the problems in question. All the examples will refer to networks which form reduced systems (see chapter 2).

The first observation is negative; the minimal set can not be found by looking for the feedback loops of the structure matrix because, as the simple example of Fig. 4.3.1 shows, the matrix does not permit distinction between feedback and feed-forward variables.

Nor is it generally possible to find the feedback loops by identifying their origin with the outputs of the complete network. Unless the network is known to be a Moore machine (see for instance (20)) state variables are not necessarily associated with the outputs of the network.

It is proposed that the residue matrix should form the basis of the computation of three relevant parameters which determine the minimal set. The parameters, associated with column variables, will be denoted by D, I and M, respectively, and will be defined and computed as follows:

<u>D</u> - This parameter counts the number of directly inter-dependent pairs of variables of which the variable in question is one. Such pairs always contain a state variable and the parameter D helps to assume the ones with the largest number of direct inter-dependences.

The directly inter-dependent pairs are characterised by symmetrically placed '1' entries about the diagonal of the residue matrix. One way of computing the D parameter of each variable is to form the transpose of the residue matrix and to count the '1' entries in each column which are common to the residue and the transpose.

<u>I</u> - This parameter counts how many variables depend upon a given variable, not counting those with which is is directly inter-dependent. Thus I is the sum of the numerals in the column of the residue matrix associated with a given variable, less its D parameter.

<u>M</u> - This parameter measures how many variables would become immediately computable by assuming only the variable in question. To determine M, the sum of the numerals in each row of the residue matrix is computed and the row(s) with the smallest sum selected. The parameter M is the sum of the column of numerals in the matrix constructed only of such minimally dependent rows.

The parameters of the variables of the residue matrix in Fig. 4.3.2 will now be computed as an example. The rows and columns are marked by alphabetic characters for convenience.

<u>Compute D:</u>

The transpose matrix, with the inter-dependent variables ringed:

$$\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & \textcircled{1} & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 \\
0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}$$

The D - parameter of each column variable therefore:

$$D_P = D_R = D_T = D_X = 0$$

$$D_Q = D_S = D_U = D_W = 1$$

Compute I:

$$I_P = 3$$
$$I_Q = I_U = 0$$
$$I_R = I_T = I_W = I_X = 1$$
$$I_S = 2$$

Compute M:

The sum of numerals in each row:

$$\Sigma_X = \Sigma_P = \Sigma_R = \Sigma_U = 1$$

$$\Sigma_Q = \Sigma_S = \Sigma_T = 2$$

$$\Sigma_W = 3$$

The minimally dependent rows:

$$\begin{array}{c|cccccccc}
 & P & Q & R & S & T & U & W & X \\
\hline
P & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
R & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
U & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
X & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

Hence: $M_Q = M_R = M_T = M_U = M_X = 0,$

$$M_P = M_S = 1$$
$$M_W = 2$$

Defining the sum of the three parameters as a figure of merit, the minimal set of state variables may be sought by assuming the variable with the highest figure of merit, reducing the residue matrix if possible. If no variables are processable, the variable with the next highest figure of merit is assumed, etc.

An alternative method assigns priorities to the three parameters in the order of D, I and M. Considering D first, obviously one variable of a directly interdependent pair must always be assumed before the processing list can be completed, and it is most advantageous to assume the variable with the highest number of interdependences. Based upon the residue matrix and its transpose, pairs of directly interdependent variables are located and the variable in each pair with the highest D parameter assumed. After this, the residue matrix is searched for computable variables as before.

In case of equal D - parameters the choice of the assumed variable depends upon the highest I parameter, or, in case of equal I, the highest M parameter of the two variables within a pair. If all three parameters of the pair are equal, the selection is random.

Following this procedure in the case of the example of Fig. 4.3.2, the directly interdependent pairs are Q - S and U - W. Since the D - parameter of variables within pairs are the same, based upon parameter I, of the first pair S is selected and of the second W. The procedure is then as follows:-

1) remove the column of assumed variables from the residue matrix.
2) search for empty rows. Remove the columns of variables to which empty rows correspond. Add these variables to the processing list.
3) repeat this procedure until no further reduction is possible.
4) prepare a new residue matrix by deleting the rows to which no column corresponds. Then prepare the parameters of the new residue matrix and repeat the procedure of determining the variables to be assumed.

5) Repeat the procedure of preparation of new residue matrices until all variables become computable.

6) The processing list can now be completed by examining the partial list which was compiled before the residue matrix was found, against the list of assumed variables, and adding those to the partial processing list which are not yet included. These variables will describe the next state of the network upon the assumption of its present state. To distinguish between the two, the assumed variables are given a suffix "s" marking them as state variables.

The procedure is now demonstrated here, in terms of the previous example:-

The assigned (state) variables, as seen, are Ss and Ws. Removing columns W and S from the residue matrix, the empty rows are P, U and X. Now removing columns P, U and X, rows Q, R and T become empty. Removing their columns, W and S are empty and the process terminates. The Processing list is therefore; P, U, X, Q, R, T, W, S. The network has thus been processed with the aid of two state variables. Since there are two separate and directly dependent variable pairs, the set is minimal.

The value of the procedure will now be demonstrated by showing that alternative choices of assumed variables result in the same or higher number of state variables.

Selecting the other two members of directly inter-dependent pairs, Q and U will be assumed. Removing their columns from the residue matrix no rows become empty and the new residue matrix is shown on Fig. 4.3.3.

It is now possible to make unfortunate assumptions, such as variables X and T, which still do not result in a complete processing list, rendering only W computable. The complete set contains no less than five state variables instead of the minimum of two.

An alternative selection, based entirely on figure of merit favours variables P, S and W equally. The unfortunate random choice of P results in a set of three state variables, because the two directly interdependent pairs must subsequently yield two state variables.

The method of selecting the minimal set upon the basis of priorities is not generally proven, but it has been found to succeed in all cases so far.

4.3.2 - The test module B.

As in case of Module A, the proposed design takes the form of a set of declarations where each member of the set concerns a different object and lists all those object identifiers to which the inputs of the declared object are connected.

No reference is required to environmental conditions. The module provides information about the system which can only be used for design verification if the designer had a notion of the expected number of state variables. The module output may contain:

a) A list of assumed variables
b) The number of assumed variables
c) The declaration of the network, containing the state variables located by Module B.

While the designer may use a) or b), it is considered that c) is only necessary as the input to a subsequent analysis module. The inputs and outputs of the module are shown on Table 4.3.1.

The processing list is shown as an output option.

4.4 - CAN THE PROPOSED DESIGN BE CONVERTED INTO A HOMOGENEOUS STRUCTURE?

The question implies that the designer used a low resolution level when declaring his system. An automatic test is now required to ascertain if sufficient information is available to describe the system gate by single gate. The test module is to perform two functions:-

a) notify the designer if the homogeneous structure can not be compiled.

b) compile the structure if possible and make the data available to other modules.

<u>4.4.1 - Techniques.</u>

To answer question 4.4 it is now necessary to describe the system in more detail than that required by modules A and B; in addition to information about the connections between system objects, the function performed by these objects must also be specified. The new information is termed the <u>function parameter.</u>

At the highest level of resolution used in this work system objects may have a function parameter selected from this list:

AND	high-level function parameter list.
OR	
NAND	
NOR	
INPUT	
OUTPUT (optional)	
STATE VARIABLE	

If any part of the proposed design is declared by the designer at a lower level of resolution, then it is the designer's responsibility to assign to each different type of system object an individual function parameter. Such parameters are subject to the syntactical rules defined by the mode of programming implementation.

The designer may wish to construct sub-systems for the purpose of a single design verification. Alternatively, he may wish to retain a sub-system, under its own function parameter, for further reference. For instance, hardware modules used repeatedly would conveniently be declared under a function parameter. Thus the designer would gradually accumulate a function parameter list containing sub-system parameters as well as the parameters of the low-level list.

The sub-networks at a lower level of resolution are generally multiple-input, multiple-output devices. The example of Fig. 4.4.1 shows such a simple sub-network. Without its source and output objects the network consists of three gates. In addition to the customary structural information the gate declarations will now contain function parameters selected from the high-level list:

```
( 42  AND  10  20 )
( 41  NOR  20  )
( 43  OR   10  41  30  )
```

Let these three gates be referenced by the function parameter TRIO and let TRIO be an object of another system called SUPER. When attempting to compile the reduced structure matrix of SUPER, TRIO will be found a misfit for two reasons; it has more than one output and its inputs are not interchangeable. Consequently TRIO must be declared in sufficient detail to permit its use in the non-reduced structure matrix of SUPER; INPUT and OUTPUT objects associated with TRIO must be identified with serial numbers. The full declaration of TRIO will therefore be as follows:-

```
(  10   INPUT    1  )
(  20   INPUT    2  )
(  30   INPUT    3  )
(  41   NOR     20  )
(  42   AND     10  20  )
(  43   OR      10  41  30  )
(  51   OUTPUT   1  42  )
(  52   OUTPUT   2  43  )
```

The INPUT and OUTPUT objects are auxiliary and should be discarded after TRIO is placed in SUPER, that is, when the homogeneous structure of SUPER is compiled and a reduced structure matrix is available. The first reference of an OUTPUT object is a serial number; the second is the indication of the object identifier connected to the OUTPUT.

Let the construction of the homogeneous structure of a network be demonstrated on another example, taken from the internal research report "The Kingston Logic Simulator" compiled by D.R. Holmes in 1969. The design of the network 'CIR 3' is to be verified. The network is shown on Fig 4.4.2 and its declaration is given in Table 4.4.1.

When the declaration of 'CIR3' is received then the function parameter of each object is checked against the list. If 'CIR2' is not found on the list then an error message is produced, indicating that the answer to question 4.4 is NO.

If 'CIR2' is found on the list then its declaration is checked against that of 'CIR3' to see if these are compatible.

Let 'CIR2' be the network declared in Table 4.4.2.

The reference to all objects named 'CIR2' on Table 4.4.1 indicates that 'CIR2' has three inputs. The declaration of 'CIR2' on Table 4.4.2 also shows three inputs, indicating compatibility of inputs.

Table 4.4.1 demands two outputs of each of the four objects whose function parameter is 'CIR2'. Table 4.4.2 shows that two outputs are in fact available, thus the outputs are compatible.

Thus 'CIR3' is an acceptable network in terms of 'CIR2'. Now 'CIR2' itself must be examined.

The function parameters show the non-standard object 'CIR1'. If this is found on the list then again a compatibility check is necessary.

Let the declaration of 'CIR1' be given on Table 4.4.3. Comparison with Table 4.4.2 shows input and output compatibility. Furthermore, 'CIR1' is composed entirely of standard objects. Thus a homogeneous structure of 'CIR3' may be prepared - the answer to question 4.4 is YES. This structure contains 52 gate objects:

'CIR1' contains 3,

'CIR2' contains 4 plus 3 x 'CIR3'; a total of 13,

'CIR3' contains 4 x 'CIR2'; a total of 4 x 13 = 52.

The homogeneous structure of 'CIR3' can now be made available by a sequence of substitutions. The data may be output if the designer wishes; more usefully this data may be available for further analysis.

For the sake of completeness the networks of 'CIR2' and 'CIR1' are shown on figures 4.4.3 and 4.4.4, respectively. The complete network of 'CIR3' is a four-bit adder.

4.4.2 - Linked outputs.

The introduction of function parameters permits designers to handle a type of network which has thus far, by implication, been inaccessible for verification modules. Networks whose object outputs are directly connected can not be handled by structure matrices and can therefore not be given meaningful object declarations.

As an example, consider the simple network of Fig. 4.4.5.

The declaration of objects 61 and 62 are straightforward. The declaration of 63 is problematic. The version (63 OR 61 62) implies an OR gate of two inputs and the logical connection

$$63 = 61 + 62.$$

However, the gate 63 has only one input and its output depends upon some detail about the hardware of gates 61 and 62 which is not available to the designer at this level of resolution.

The problem may be approached in two ways:

1) by increasing the resolution level beyond the level of a single gate. This would reveal the mechanism used by the hardware to eliminate the ambiguity between the conflicting signals of gates 61 and 62.
2) by introducing some artificial device at the selected level of resolution which would have the same terminal performance as the junction of the outputs.

The first of these approaches is unattractive; increasing the resolution level beyond that of a single gate renders the network non-logical. The operation of such a network would have to be examined with reference to non-linear sub-system models.

The second method is simple but can not be automated; it demands that the designer should insert a dummy gate at the junction of the outputs and assigns to it one of two logical functions. If, at the junction, logical '1' has priority then the output of the dummy gate is '1' due to any of the joined outputs being at '1' and the dummy performs an OR function. If, at the junction, priority is given to logical '0' then the output of the junction can only be '1' if all of the joined outputs are at '1', therefore the dummy should be an AND gate. The designer must add the dummy at coding and assign to it the appropriate function by examining the priority.

This is the method recommended here.

To illustrate the method it will now be assumed that the joined outputs of Fig. 4.4.5 act as a wired OR gate. Assigning the identifier '64' to the dummy gate, the declaration of the network is as follows:-

```
( 61  AND  10   20 )
( 62  AND  30   40)
( 64  OR   61   62)
( 63  OR   64   )
```

4.4.3 - The test module C.

The proposed design is presented once more as a series of declarations containing structural and functional descriptions.

No reference is needed to environmental data. The behaviour specifications are implied; acceptable designs consist of compatible and recognised objects which lend themselves to translation into homogeneous structures.

The output is a YES - NO answer to the question 4.4. In addition, the homogeneous structure is to be made available.

With reference to the input requirements of modules A and B it will be seen that the output of module C contains all the necessary data in the required form. Module C also generates data concerning function parameters; this data is redundant so far as modules A and B are concerned. Thus, if module C is to act as a translator for A and B, function parameters are first to be omitted or just ignored.

The module inputs/outputs are shown on Table 4.4.4.

4.5 - ARE ANY OF THE OBJECTS OF THE SYSTEM MISUSED OR OVERLOADED?

The question seeks to locate design errors due to two reasons:-

1) the fan - in restrictions are violated.
2) the fan - out restrictions are violated.

It will be assumed that errors are systematic and not due to faulty coding of otherwise correct networks. For the sake of clarity of these discussions it will also be assumed that the proposed design is presented in homogeneous form. This latter assumption may be removed without altering the validity of the discussion or of the test.

4.5.1 - The test module D.

The objects of the proposed system must now be described in detail beyond that demanded by earlier tests. Each object must carry three additional parameters, collectively termed "load parameters":

a) Fan - in. This is an integral numeral indicating the number of input terminals the object has. The number of input references in the object declaration will, in case of correct design, be smaller or equal to Fan - in.

b) Load. This is a real numeral indicating, in some selected unit, the maximum load demanded by any of the inputs of the declared object.

c) Capacity. This is a real numeral indicating the ability of the declared object to supply load to other objects, connected to its output. Capacity is expressed as a num~~b~~eral referring to the same unit as Load.

The techniques applied by the test module are trivial, consisting of arithmetic operations and comparisons.

The output is diagnostic, specifying the erroneous object and classifying the error as "FAN-IN" or "OVERLOAD".

The inputs and outputs of the module are shown in Table 4.5.1.

4.5.2 - Comments.

In case of wired logic (linked outputs, section 4.4.2) due care must be exercised when specifying the load parameters of the dummy element. Current hardware practice implies a limit upon the fan-in and upon the capacity of the dummy gate. These two parameters are usually inter-related and specified in terms of each other by the manufacturer. The load represented by the dummy gate may be considered zero since the wired junction commands all of the output current of the linked gates.

Another remark is appropriate. The fan-out capacity of several types of hardware in present use is a function of the distance between connected objects. Module D has no facility for accommodating such and other non-constant load parameters. The extension of the module would assign to each object a sub-routine which calculates the load parameters on the basis of a selected set of variables. In the absence of reliable information about hardware, such a facility could not be put to efficient use; thus, at the present time, module D is thought best served by a set of constant load parameters.

4.6 - IS THE STEADY-STATE RESPONSE OF THE COMBINATIONAL NETWORK CORRECT?

The question implies that the design under verification has been proven by an earlier module (A or B) to be combinational. It also implies that the correct behaviour is described in some specifications to which reference can be made at the time of assessment. Since the behaviour specifications are normally given with reference to environmental signals, analysis should also be conducted with reference to these signals.

Supposing now that structural and functional information is available about the proposed system and its object, the question is now whether this information is adequate as a basis for answering question 4.6, or is some other detail such as time parameter of objects, necessary.

4.6.1 - Techniques and concepts.

In the literature combinational networks are customarily described as realisations of propositional functions. This description is, strictly speaking, only valid if the delay of all network objects is zero. Obviously this is never true in practice. On the other hand, when the steady-state response is required then it is assumed that the inputs have been held constant since $t = -\infty$ and the outputs are propositionally related to these at any finite value of t. Consequently the object delays are irrelevant to the calculation of the steady-state response.

Inputs.

It was discussed in chapter 2 that environmental signals fall into two classes; physical and operating signals. If it can be assumed that the admissible range of physical signals is controlled in such a way that no gate fails to perform its prescribed Boolean function then the physical signals can only influence the length of the time delay of system objects which in any case have no relevance to the steady-state response. Thus the analysis consists of finding the mapping of operating signals u into outputs Z by a system which has no states.

Let the design under verification be a network with n operating signals. The highest resolution level - that of a single gate - implies an n - dimensional sample space with a binary choice in each dimension.

The sample space in three dimensions (i.e. the sample space of a three-input network) is shown in Fig. 4.6.1. Points connected by an edge of the cube are unit distance apart and are one-bit different in their code (see, for instance, (18)).

The "designation numbers" devised by Ledley (~~25~~) map this space into a binary matrix with a row assigned to each operating (input) signal and a column to each point in the space. The designation number for each of the three variables is shown in Eq. 4.6.1. where dU_i is to be read "the designation number of U_i".

It will be noted that the adjacency between points representing unit-distance codes is lost; nor is it considered worthwhile to transcribe the designation number equation to Gray-code form since such a code preserves the adjacency in two dimensions only.

The concept of designation numbers has led to the automatic scanning of the sample space. A designation number can conveniently be handled by digital computers in one of two ways: in the form of an array of words, each word representing a binary digit of the array, or in the form of an array of bits within a computer word, each bit representing a bit of the array. The latter solution is obviously advantageous and holds the potential of increase in the efficiency of computation by a factor equalling the number of bits in the computer word. This bit-handling method is adopted here.

The length of the designation number array is an exponential function of the number of operating signals and array lengths exceed the word length for all but trivial networks. To accommodate arrays which contain more bits than the word-length W, "multi-words" are formed of length of P.W bits, where P is the smallest positive integer for which $P.W \geq 2^n$.

Designation number arrays of operating signals have regular patterns; the input variable U_i has 2^{i-1} number of '0'-s followed by the same number of '1'-s, with this sequence repeating to a length of 2^n bits, for a total of n inputs. If the network is subsequently extended to have an additional input, all the designation number arrays must be increased to a compatible size of 2^{n+1} bits.

The automatic generation of inputs by the use of designation number arrays is implemented by a program package ISOPACK (27) which also provides manipulation and storage routines for the handling of such arrays. ISOPACK assigns the lowest order array to the source object carrying the lowest serial number; thus the source object 10 of Fig. 4.6.2 would be declared as

(10 INPUT 1)

and its designation number would be a 2^{1-1} bit sequence of '0's followed by the same number of '1's (see Table 4.6.1). The next input will be declared as

(20 INPUT 2)

and its array would consist of a 2^{2-1} bit sequence of '0's and '1's, etc.

Since many networks operate in an environment which is restricted to specified areas of the sample space, it was found advantageous to permit the automatic generation of all-zero and all-one designation numbers of inputs as well as the use of input patterns designated by the designer. The input declarations

(10 INPUT 1 FIXED Ø)
(20 INPUT 2 0001)
(30 INPUT 3)

will assign the patterns

d 10 = 0000
d 20 = 0001
d 30 = 0101

to source objects 10, 20 and 30 respectively. The length of the patterns is fixed by the highest demand; here the specified pattern of source object 20 sets the length to 4 bits; the other two demand shorter lengths but conform to that of the highest demand.

Analysis.

Having found a solution to the generation of input signals, analysis may now commence. This would merely consist of the computation of object outputs as Boolean functions of object inputs. The computation must be made in an order in which object inputs become computable. It is opportune to utilise the Processing List which was generated as optional output of module A or B.

A simple example is shown on Fig. 4.6.2. The processing sequence is not unique but may be, say, (1, 2, 3, 4). The designation number array of object outputs is shown, together with inputs, on Table 4.6.1. The manipulation routines of ISOPACK permit the simultaneous (PARALLEL) calculation of up to W number of bits of each designation number array for a machine of W number of bits per word.

The analysis is thus very simple; a problem is presented by the size of arrays of networks with large number of inputs. Now multi-words consist of several words (P is large) and while one word of the multi-word is actively manipulated, (P-1) words of each array need storage. To overcome this problem a procedure is added which estimates the storage demand of a given analysis. If this demand proves excessive for the available storage capacity then the length of multi-words is halved; the procedure is repeated until the estimated demand is met. This mode of analysis is termed SERIES-PARALLEL, by contrast to the fully PARALLEL mode when all P words are simultaneously accepted. In series-parallel mode some of the high-order input arrays are not accommodated, hence only a fraction of the sample space is covered by the analysis; thus several runs of analysis are needed, one after the other (in series) to permit comprehensive analysis. The method is described in detail in ref.(32).

The output of the analysis is the designation number array of the system output or outputs. The printout of these arrays, together with the input arrays serving as reference, is an option designers may demand. This truth table needs assessment against behaviour specifications in order to give an answer to question 4.6.

<u>Specifications and Assessment.</u>

The behaviour specifications may be presented in several forms of which three is considered here:-

a) a standard network is specified whose output is known to be correct.

b) the desired behaviour is given in the form of Boolean equations.

c) Same is given in a truth table.

Since the analysis of the proposed design results in a truth table, it is necessary to bring the specifications in the same form. In the case of a), the standard network is analysed in the same way as the network under test.

In the case of b), symbolic manipulation routines are necessary to interpret the equations which may be given in a nested form. When a homogeneous structure of equations is found then inputs are made to assume the array patterns as before and the outputs are computed by the stack manipulation routines of ISOPACK.

Case c) is trivial.

The manipulation of the specifications, bringing them to truth table form, is the main part of Assessment; the process is concluded by a comparison between appropriate arrays, facilitated once again by ISOPACK.

4.6.2 - The test module E.

The proposed design is presented as a series of declarations containing information about structure and function parameters.

Auxiliary input is required in the form of the Processing List (module A or B). This means that module E is not self-sufficient and can only be operated subsequently to A or B.

Environmental data may be optionally given, describing the restrictions of the sample space. In the absence of such data the space is fully and automatically covered.

Behaviour data is to be presented in one of three standard forms.

The output is either an implied YES or a sequence of truth table entries indicating those input conditions which have led to error.

The module inputs/outputs are shown on Table 4.6.2.

4.7 - ARE OUTPUTS OF THE COMBINATIONAL NETWORK SUBJECT TO TRANSIENT SPIKES?

Combinational networks of n inputs are, by definition, operating in n-dimensional space, having no state variables at the selected level of resolution. If they should now be resolved to a level higher than that of a single gate, the model of the network would contain numerous reactive objects, indicating the presence of several state variables. The effect of these state variables is observable under transient conditions at the level of single gates. The transient behaviour of combinational networks forms the subject of this section.

4.7.1 - Concepts and definitions.

The combinational network will be defined here to exist in a stable state if the inputs and outputs of each of its objects are propositionally related. Such a state is therefore characterised by valid Boolean relationships between inputs and outputs of all objects of the network.

Let a network be composed of a finite number of objects. Let each object have a finite but un-specified time delay associated with it. Let the system now rest in a stable state Y_o up to and at a time t_o, to which a set of inputs u_o and outputs Z_o correspond. If the inputs now change at t_o to u_1 then the system will be in some logically inconsistent state for a finite but unspecified period τ until a time $t_1 = (t_o + \tau)$ when a new stable state Y_1 is reached to which the logically consistent inputs u_1 and outputs Z_1 belong. In the period of τ the network is said to be in an unstable state $Y\tau$. In fact, $Y\tau$ is not a single state but an infinite state sequence as dictated by the reactive objects of an inaccessible high resolution level.

Let D_k be the k^{th} object of the system at the resolution level of single gates. Let the state of the outputs of D_k be given at the stable states Y_o and Y_1 as D_{ko} and D_{kl} respectively. It will be observed that, since D_k is a gate, it may only have a single output, hence D_{ko} and D_{kl} are characterised by a single bit each. If $D_{ko} \equiv D_{kl}$ then during the change of u_o to u_1 D_k is said to be static; otherwise D_k is dynamic.

If it were possible to manufacture hardware with delay-free gates then $Y\tau$ would not exist. In this case if D_k would be static, its output would not change at all; if D_k would be dynamic, its output would undergo a single change.

Due to finite-delay hardware, D_k transfers from D_{ko} to D_{k1} in the time interval of τ, passing through a sequence of transient states. Since the object is certain to reach the state of D_{k1} at the time t_1, in the course of τ it may either undergo <u>no change</u> of output and will be said to be <u>free of hazard</u> or else undergo an <u>even number of changes</u> of output and will be said to be <u>hazardous</u>.

The above discussion uses accepted terminology but gives, it is thought, more meaningful definitions. It will be useful to mention here that it is customary in the literature to distinguish between static and dynamic hazards, '1' and '0' hazards and single-and multiple hazards as shown on Fig. 4.7.1. These distinctions have been found unnecessary in the course of this work; the cause of single static hazards has been found to be always the same and they can be detected without reference to polarity.

Dynamic or multiple static hazards on the other hand are the consequence of single static hazard spikes existing in the system; detecting and correcting the latter will eliminate the former, which are therefore of no concern to the designer.

<u>4.7.2 - Techniques, further concepts and definitions.</u>

It will be assumed throughout this section, as indeed throughout this work, that systems under test operate in fundamental mode (see, for instance, (18)). This means that a change of operating signals is only permitted if the system has reached a steady state in response to the previous change. The need for such an assumption arises in the context of question 4.7 when examining the sample space of operating signals. If fundamental mode can not be assumed then it becomes necessary to include the state space of transient response in the dimensions of the sample space whose size is already embarrassing. The state space could not easily be described in the chosen resolution level and the problematic sample space would obscure rather than illuminate the system behaviour.

The next assumption is that the behaviour should be observed with reference to operating signals. This assumption may, in fact, be abandoned by developing analysis techniques based upon the observation of topology and function parameters (see FURTHER DEVELOPMENTS).

The sample space of an n-input combinational network contains 2^n number of points. Adding the dimension of "transient space", it must now be considered that, for comprehensive testing, the system must be analysed under transit from each point to each of the others. The total number of transient tests is thus $2^n.(2^n - 1)$ - too large in all cases of interest.

One way of restricting the sample space without prejudicing reliability to a great extent, is to eliminate those tests which refer to events of low probability. To find such events, attention was given to the way in which the system is expected to operate in real life.

The logical decisions made by combinational networks are used to control some other system. If the time constant of the controlled system is so small that it may react to output transients then the designer may choose to use a clocking device which inhibits the output until the transient is over - if such a clock is available. Thus clocked (synchronous) systems usually cause no anxiety on account of their transient behaviour.

If the system is unclocked (asynchronous) then the time of the occurrence of events is stochastically determined, depending upon the time delays of earlier parts of the system. In this case the change of operating signals is also stochastic and the probability of two signals changing simultaneously is zero. Thus the sample space may, without much loss of reliability, be restricted to unit-distance (single-input) changes. Due to a change in each of n input variables, a transition may occur from each of 2^n points of the sample space; the total number of tests needed for comprehensive testing under single-input-change conditions is $n \times 2^n$. Throughout this section tests will be conducted under such conditions.

Let the Boolean function relating the inputs V_k of the system object D_k to the output, also denoted by D_k, be given as

$$D_k = B_k(V_k) \; . \; . \; . \; . \; . \; . \; . \; . \; \text{Eq. 4.7.1.}$$

By the definition of the high-level function parameter list of section 4.4.1. the function B_k is selected from the list OR, AND, NOR, NAND. Since inputs and outputs of all the gates of the system are connected by such equations as Eq. 4.7.1, D_k will be connected to the system inputs u by a complex combination of Boolean connectives C_k :

$$D_k = C_k(u) \; . \; . \; . \; . \; . \; . \; . \; . \; . \; \text{Eq. 4.7.2.}$$

where $u = \left(U_1, U_2, \ldots U_n \right)$ for n inputs.

<u>The partial difference</u> of D_k under U_j will be denoted as $\frac{\Delta D_k}{\Delta U_j}$ and will be defined as the function

$$\frac{\Delta D_k}{\Delta U_j} = C_k(U_j) \; . \; . \; . \; . \; \text{Eq. 4.7.3.}$$

which is obtained by setting all the input variables of u to Boolean constants of either '1' or '0', with the exception of U_j. Since the remaining (n-1) input variables have 2^{n-1} different combinations of values, the partial difference of D_k under U_j is not unique, i.e. Equation 4.7.3. may have several forms. Should any of these forms be reducible, by normal Boolean algebraic manipulations, to one of the forms of Eq. 4.7.4 or 4.7.5, then the object D_k will be said to be <u>hazardous under U_j</u>.

$$\frac{\Delta D_k}{\Delta U_j} = U_j + \overline{U}_j \; . \; . \; . \; . \; \text{Eq. 4.7.4.}$$

$$\frac{\Delta D_k}{\Delta U_j} = U_j \cdot \overline{U}_j \; . \; . \; . \; . \; . \; . \; \text{Eq. 4.7.5}$$

If B_k was an OR or NAND function then the hazard will show in the form of Equation 4.7.4. If it was AND or NOR then the hazard takes the form of Equation 4.7.5.

These equations (without the use of the concept of partial difference) are the basis of hazard detection as used by Zissos and others (see for instance, (19)).

An example is shown in Fig. 4.7.2.

For this network - $D_3 = U_1 . U_2 + \overline{U2 + U3}$ and the Boolean connective of D3 is OR.

The partial difference of D3 under U_2 will now be sought. This will have four potentially different versions as shown on Table 4.7.1.

The third row of the table indicates the hazard under U2 in the form of Equation 4.7.4.

<u>4.7.3 - The test module F.</u>

It is now possible to devise a test module which produces the answer to question 4.7.

<u>The inputs</u> to the test module consist of structural and functional information about the system objects in the same format as that of module E. In addition, as in the case of E, the processing list is required; thus this module relies upon module A or B for some of its inputs.

<u>Analysis</u> consists of converting the gate declarations, which give equation 4.7.1 of each gate, into the form of equation 4.7.2 by the use of symbolic manipulation routines. Next, each of the 2^{n-1} partial differences for each of the n variables (a total of n x 2^{n-1} calculations) are formed for each of p number of objects of the network, and manipulated by the same routines as above until no further simplification is possible.

<u>Assessment</u> is conducted with reference to the error equations

of 4.7.4 and 4.7.5. Each of the p x n x 2^{n-1} equations must now be compared with the appropriate error equation. Since the error equations are of standard format, no behaviour specifications need be given.

The output is diagnostic, consisting of a list of hazardous gates. Two options are possible: nominating the operating signal(s) under which the faulty gates are hazardous, or else nominating the particular combination(s) of constant operating signals which lead to hazard under the j^{th} input.

Table 4.7.2. shows the inputs/outputs of module F.

4.7.4. Alternative techniques.

The mode of analysis used by Test F was, basically, analytic testing because it was built on symbolic manipulation. As an alternative, numerical testing may be used to detect hazards. The method was suggested by Zissos (19) and is the foundation of the test module G.

The error equations of 4.7.4 and 4.7.5 imply that under hazard conditions two of the inputs undergo opposite polarity changes while all other inputs are ineffective in eliminating the hazard pulse. Thus the hazardous gate may be found by scanning all of the p number of objects of the system under each of n single-variable transitions and starting from each of 2^n stable states, observing if any pair of object inputs are liable to opposite-polarity change. Should such a pair be found then the object is potentially hazardous. If there are no input signals which would be effective in blanking out the spike then the object is hazardous and should be reported to the designer.

Examination of this procedure reveals that comprehensive testing is accomplished in n x 2^{n-1} carefully selected tests because it is sufficient to examine one of two possible polarities of change for each input variable.

The procedure will now be demonstrated on the example of Fig. 4.7.3.

10, 20 and 30 are source objects; gates 42 and 43 represent AND functions; 44 is an OR gate and 41 is an inverter (NOR).

The steady-state response of the network is found by Module E. It is tabluated on Table 4.7.3., where rows are numbered for reference.

Starting from each of the $2^n = 8$ possible steady-state conditions, $n = 3$ single-variable changes are possible, as shown on Table 4.7.4.

It is possible to halve this number of tests by permitting one of the two possible polarities of transitions. The choice of polarity is arbitrary. Electing here to transfer from a lower to a higher reference state, the list of necessary tests is shown on Table 4.7.5.

Table 4.7.6 shows the output of each gate in response to these input changes. Each column gives the logical state of the gate output before and after the input transition. The head of each column indicates the transition from one input condition to another. Arrows are reversible, indicating that the polarity of the change is immaterial.

The gates of interest are those of more than one input, since these are the only ones open to hazard. In this network there are only three: 42, 43 and 44. The inputs to these gates are now examined, with the aid of Table 4.7.6, for opposite-polarity changes. The only static hazard of the network may arise at the input of gate 44, at the transition of $6 \rightarrow 7$, as ringed on Table 4.7.6. Gate 44 is therefore entered in a provisional error list. A further test is now required to detect the presence of blanking signals which would inhibit the hazard spike. These signals maintain a steady level throughout the hazardous transition such that the hazard spike does not propagate. Such blanking signals may be applied at the hazardous gate or at some other gate to which the hazard spike propagates. Since scanning the whole of the network for possible blanking signals is judged to raise excessive demands of computing time and space, an error is reported in terms of the signals of a given object, even if the hazard is subsequently eliminated by blanking signal.

The technique of detecting blanking signals at the input of a potentially hazardous gate is based upon the observation that 'OR' and 'NOR' gates demand a constant signal at logical '1' which would keep the gate output steady during the hazardous transition; 'AND' and 'NAND' gates need a constant signal at logical '0' for the same reason. Since the output of the hazardous gate is, under steady-state conditions, static, the blanking signals are logically redundant. The transient-error-free redundant version of the network of Fig. 4.7.3 is shown on Fig. 4.7.4. The blanking signal is generated on gate 45.

4.7.5 - The test module G.

The test is based upon numerical methods.

Inputs consist of the truth table generated by module E, plus the processing list.

Analysis compiles the n x 2^{n-1} single-variable transitions for each of the p objects of the system.

Assessment consists of two parts. First opposite-polarity input pairs are located, resulting in the diagnosis of potentially hazardous gates. Next, the remaining inputs of such gates are searched for effective blanking signals. Assessment is thus based on generalisations, requiring no behaviour specifications.

Outputs list hazardous gates with the same options as module F. Table 4.7.7 shows inputs/outputs.

4.8 - IS THE STEADY-STATE RESPONSE OF THE SEQUENTIAL NETWORK CORRECT?

It will be assumed that the proposed design is a sequential network under the definition given in 4.2:- at the resolution level of single gates the network will contain feedback. It will be further assumed that the network is declared in terms of an adequate set of state variables (by the use of module A, module B or otherwise). In order to make the question 4.8 meaningful, it must also be assumed that the desired behaviour of the system is known to the designer in terms of the same set of state variables.

A technique is now sought to permit the assessment of the response of the new design without reference to externally specified operating signals or to time delays of system objects.

4.8.1 - Technique.

Let the network have n inputs and m state variables. Then, under steady-state conditions, the sample space has (n + m)dimensions and 2^{n+m} number of points.

The mapping of inputs and state variables into outputs and the next value of state variables is given by the system equations of Chapter 2, if the operators G and H are interpreted as Boolean operators and the input can be considered constant over the time segment (t_o, t_1).

$$\begin{aligned} Z\ (t_1) &= G\ (u\ (t_1),\ s\ (t_o)\ ) \\ S\ (t_1) &= H\ (u\ (t_o,\ t_1),\ s\ (t_o)\ ) \qquad \text{Eq. 4.8.1.} \end{aligned}$$

It will now be decided to search for a numerical test method. Analysis could then consist of compiling the system equations in a table. A convenient format is proposed to be a DIRECTORY which combines the information of state tables and output tables (for instance, (18)).

The DIRECTORY consists of 2^{n+m} number of rows and (n+2m+q) number of columns for a network with n inputs, m state variables and q number of outputs. Each row corresponds to a different point of the sample space.

The operating signals and state variables at the time t_o are the independent variables determining the parameters of a point of the sample space. They are assigned a column each in the DIRECTORY. The input is regarded as constant over the time segment (t_o, t_1).

The outputs and the state variables at the next instance t_1 are the dependent variables which are to be computed by the analysis and assessed against the behaviour specifications. They too are assigned a column each in the DIRECTORY.

The DIRECTORY appears in the format of a truth table relating independent and dependent variables. In this interpretation the sequential network of n inputs, m state variables and q outputs is resolved into a combinational network of (n + m) inputs and (m + q) outputs. Since the techniques of module E cope adequately with such networks, these techniques can be adopted for the sequential test module H.

The transformation of a sequential network to an equivalent combinational form is shown diagramatically on Fig. 4.8.1. The use of this transformation in analysis is demonstrated on the simple network of Fig. 4.8.2.

The state and output equations for the model of Fig. 4.8.1 can now be written, with reference to Eq. 4.8.1, as

$$Z\ (t_1),\ s\ (t_1)\ =\ J\ (u\ (t_o),\ s\ (t_o)\ )$$

or else,

$$D\ (t_1)\ =\ J\ (I\ (t_o)\ )\ .\ .\ .\ .\ .\ .\quad \text{Eq. } 4.8.2.$$

where D and I denote dependent and independent variables, respectively and J is a Boolean operator.

The DIRECTORY is the tabular model of Eq. 4.8.2.

Here 40 is the state variable inserted by the designer or module B. The output of 30 represents the "next state" of the state variable. The output is taken from 10. Thus there are

$(n + m) = (2 + 1) = 3$ independent and

$(m + q) = (1 + 1) = 2$ dependent variables.

The processing list may be, say, (10, 20, 30). The allocation of order to designation number arrays is, in principle, arbitrary; in practice it has been found convenient to assign low order input arrays to state variables (see (24)). Table 4.8.1 shows the arrays of the network of Fig. 4.8.2.

The truth table of Table 4.8.1 is re-arranged in the format of Table 4.8.2. which will hence be referred to as the DIRECTORY. SP and SN abbreviate "present" and "next" states. Rows are given a serial number for reference.

The behaviour specifications can thus be presented in one of three forms: as a standard network, a set of state and output equations or as a DIRECTORY.

4.8.2 - The test module H.

The proposed design is once more presented as a series of object declarations containing information about structure and function parameters. In addition, the processing list compiled by module A or B is necessary.

Environmental data is optional and is used merely to describe restriction of sample space.

Behaviour data takes one of three standard forms.

Output is an implied YES or a sequence of DIRECTORY entries indicating those parameters of the sample space which have lead to error. The total DIRECTORY is available as an option.

Inputs/outputs are shown on Table 4.8.3.

4.8.3 - Comment.

The model of the system as shown on Fig. 4.8.1. assigns the nature of independent variables to all inputs and state variables of the system. In fact state variables depend upon inputs and their logical value is not directly controllable. Interactions between state variables are frequently functions of the delay of system objects and, dependent upon the relative value of these delays, the system may reach one of a number of possible stable states. Thus, unlike in combinational networks, transient conditions in sequential networks may become staticised, causing permanent mal-function as a consequence of adverse path delays. In terms of the analysis technique of module H this means that the DIRECTORY is a valid model of the network only if one of two conditions are satisfied:

a) the network is not sensitive to changes of relative delays in signal paths

b) the network processes the signal in the order given in the Processing List.

It is the purpose of a number of test modules to ascertain if conditions a) or b) are satisfied; meanwhile, the output of module H must be regarded as tentative; the resulting DIRECTORY is termed PRIMITIVE. A definite answer to question 4.8 will only become available after a VERIFIED DIRECTORY is compiled, with the aid of further test modules and as the output of module P.

4.9 - WHICH ARE THE STABLE STATES OF THE SYSTEM?

The importance of the question is matched only by the simplicity of obtaining the answer, using the DIRECTORY of module H. By contrast it will be recalled from Chapter 3 that simulation programs have no effective means of locating satisfactory stable conditions from which to start the analysis and neither simulation nor lower-order modes of analysis have any means of reaching some of the stable states of incompletely connected systems.

4.9.1 - The test module J.

Let the proposed design be described by its DIRECTORY after the use of module H. This data represents the input of module J.

Analysis consists of comparing the present and next state vectors within each of the rows of the DIRECTORY. Where these agree the system is evidently static and will stay so until the inputs are changed. The parameters (input and present state variables) of these points of the sample space will define the stable states of the system, hence these represent the answer to question 4.9.

Inputs and outputs are shown in Table 4.9.1.

4.9.2 - Comment.

The stability analysis may be conducted on the basis of the PRIMITIVE DIRECTORY because the stable conditions of a network are not dependent upon the relative delays of signal paths. Indeed, the stable states of a sequential network may be considered as the equivalents of the stable states of a combinational network. Consequently the STABLE entries of the PRIMITIVE DIRECTORY require no further verification.

4.10 - ARE ANY OF THE GATES OF THE SEQUENTIAL NETWORK LIABLE TO GENERATE STATIC HAZARD SPIKES?

Having reduced sequential networks to equivalent combinational form by breaking feedback loops and inserting state variables, it becomes possible to locate static hazard errors by the use of the techniques of module F or G. However, the test conditions require some thought.

4.10.1 - Technique.

It will be recalled (section 4.8) that, for the purpose of compiling the DIRECTORY, state variables were regarded as independent variables and the sequential network was effectively reduced to combinational form. The model of this procedure is shown on Fig. 4.8.1.

Then again, it will be remembered that section 4.7 led to the conclusion that n x 2^{n-1} tests were necessary for comprehensive analysis of an n-input network. Thus, if the "independent variables" of section 4.8 are interpreted to be the "inputs" of section 4.7, (n + m) x 2^{n+m} number of tests are required to give a complete answer to question 4.10.

Fortunately this very large number grossly over-estimates the sample space due to the wrong interpretation of the nature of "independent variables". The transient tests of section 4.7 were based on the concept that the network could always be transferred from one stable state (Y_o) to another (Y_1) under the influence of a single input change, while other inputs are kept constant. This condition is in no way valid for sequential networks: if such networks are stable in some state Y_o, they can be moved from this state by changing any one of the n input variables but, since the m present-state variables are not connected to any directly accessible terminal (except in the conceptual model), they are not liable to change independently of the n inputs. Consequently, if the network has P number of stable states, it is only liable to n x P number of transients.

Unfortunately this attractively modest number of tests does not cover the sample space completely. If a network starts from the stable state Y_o and undergoes a change of state due to some changing input variable U_j, the new state Y_1 may be stable or unstable. If Y_1 is stable (under the criterion of 4.9) then the transient test is complete. If Y_1 is unstable, the DIRECTORY will indicate a number of state variables which are scheduled to change. Hazards may arise due to the change in these, which need diagnosis and subsequent correction. Still applying the principle of single-variable change, static hazard tests should now be conducted by starting from each of the $(2^{n+m} - P)$ number of unstable states and subjecting the network to change in each one of the state variables indicated by the DIRECTORY. The number of such changes in a well-designed network will be seen to be one or not much more than one. However, in a general case the number may be anything up to m.

The total number of tests will be computed as

$$n \times P + \sum_{i=1}^{2^{n+m}-P} \Delta m_i$$

where Δm_i indicates the number of state variables which are scheduled to change in the i th row of transient states.

An example will now be appropriate. Let a simple sequential network be characterised by the DIRECTORY of Table 4.10.1. There are two inputs and two state variables, giving $2^{n+m} = 2^{2+2} = 16$ rows of DIRECTORY entries. A column has been added to the DIRECTORY for the purpose of this discussion, indicating the STABLE entries (i.e. those for which Present and Next state variables are identical) and Δm for each row, showing the number of state variables scheduled to change. The sample space of this example has X number of points, i.e. X number of tests are necessary as a basis of answering question 4.10, where

$$X = n \times P + \sum_{i=1}^{2^{n+m}-P} \Delta m_i =$$

$$= 2 \times 6 + (1 + 1 + 2 + 1 + 1 + 1 + 1 + 1 + 2 + 1) = 24$$

Careful observation of Table 4.10.1 will reveal some redundancy in this mode of testing, indicating that some points of the sample space are covered more than once. Take for instance the stable states of rows 9 and 13 which differ only by the value of the input variable U_1. One of the changes scheduled under X will transfer row 9 to row 13; another will transfer row 13 to row 9. It was possible to reduce the number of tests of section 4.7 from $n \times 2^n$ to $n \times 2^{n-1}$ due to just this type of redundancy. It would also be possible to eliminate the redundancy here, but this would involve searching for stable states which differ only by one input variable. Since stable states form only a fraction of the total number of system states, the extra computation of the search may not be offset by the saving in the number of tests; thus the method is not proposed here.

Another comment is called for when considering the implications of tests starting from a transient state. When such a state is reached, as a result of an input change, it will be considered as virtually stable for the purposes of module K. The test will then move the system from a virtually stable state under the influence of each state variable in turn. The circumstances in practice are different: the system is not subjected to partial changes as above but to the simultaneous change in input and each scheduled state variable. Such multiple changes cause erroneous operation which will be detected under some other test. Meanwhile module K concentrates on the detection of errors on the basis of an abstract concept: the virtually stable state.

4.10.2 - The test module K.

The test matches the numerical techniques of module G.

Input consists of the DIRECTORY of module H plus Processing List.

Analysis compiles $n \times P + \sum_{i=1}^{2^{n+m} - P} \Delta m_i$ number of single-variable transitions for each object of the network, registering hazardous gates.

Assessment needs no behaviour specifications.

Outputs list the hazardous gates with a single option of the definition of the independent variable leading to hazard.

Inputs/outputs are shown in Table 4.10.2.

The module is not self-sufficient but relies for its inputs on module A or B and module H.

4.11 - IS THE STEADY-STATE RESPONSE OF THE SEQUENTIAL NETWORK SENSITIVE TO VARIATIONS OF DELAY PARAMETERS OF SYSTEM OBJECTS?

In other words: is the behaviour liable to deviate from the specification as given in a DIRECTORY due to normal variations of delay parameters, such as caused by manufacturing tolerances or changes in physical signals of the environment? Is the PRIMITIVE DIRECTORY ambiguous?

The question is of vital importance to the designer and, as shown in Chapter 3, answers are not found by simulation or lower-order modes of analysis.

The techniques used here are based on standard techniques of switching theory with some suitable modifications which permit the design of test modules. It was also necessary to introduce some new concepts and definitions.

4.11.1 - Concepts and techniques.

The examination of question 4.11 leads to two further questions:

1) to what extent would the behaviour specifications, as given in a DIRECTORY, reveal sensitivity to delay parameters?
2) to what extent can question 4.11 be answered without direct reference to the delay parameters of objects of a given network?

The second question will only be answered after the techniques are established.

The first of these questions implies that it may be possible to predict delay sensitivity before a new design is initiated. If this is true, then the new design may be created with special attention to object delays, thus avoiding or minimising design errors due to this cause.

Let this question be considered with reference to a specific example. The DIRECTORY of Table 4.10.1 may be the outcome of the analysis of a network by module H. It may also be the tabular specification of a new design to be created. What can such a terminal model reveal about sensitivity to delays of objects which, at this the lowest level of resolution, are not distinguishable?

The rows of the DIRECTORY will now be examined in turn. If the system is stable then it can not be affected by the delay or any other parameters of its objects. Thus rows 0, 6, 9, 10, 13 and 15 are exempt from further scrutiny.

Paying no attention for the moment to the way in which the system reaches one of the unstable states, let these now be examined in turn. Rows 1, 2, 4, 5, 7, 8, 11 and 14 schedule a single state variable change which will undoubtedly occur sooner or later, so long as the input signals (U_1 and U_2)are constant. Let this condition be assumed just for the moment; then the only rows indicating multiple changes in state variables are 3 and 12. Still assuming that the input variables are constant, one of three events may occur to the network which is at one of these two states:

a) both state variables change together.

b) S1 changes before S2.

c) S2 changes before S1.

The probability of a) in an asynchronous network is zero. If the network is synchronous and is operating within the limits of its response time, then the answer to question 4.11 is NO and none of this discussion is relevant.

Without further information about the system and its object delays, b) and c) appear equally probable and a condition arises which is referred to in the literature as a SECONDARY RACE, indicating that two or more secondary (state) variables are involved simultaneously and the outcome is in doubt.

With reference to the example of Table 4.10.1, if conditions b) and c) lead to different stable states then the answer to question 4.11 is YES.

Contemplating row 3 first, the case of b) indicates that the state variables change from 11 to 01. For constant inputs this condition is that of row 2 which finally leads to the stable state of row 0, where the state variables become 00, as was originally scheduled in row 3. This condition will be termed NON-CRITICAL because of the agreement of the originally scheduled and finally reached stable state.

Row 3 and the case of c) indicates a change of state variables from 11 to 10. For constant inputs this condition is met in row 1 which leads to the stable state of row 0 - another non-critical transition. Thus row 3 shows no sensitivity to object delay variations.

Repeating the procedure for the other potentially sensitive row of 12, the originally scheduled change from state variables 00 to 11 prescribes the final state of row 15. Case b) however leads to the stable state of 13 - this condition is CRITICAL. Case c) does lead, via row 14, to the desired stable state of row 15.

The summary of these investigations is that if the network ever reaches state 12 then it will be sensitive to delay variations and will in fact mal-function if the delay associated with the state variable S1 is greater than that of S2, (condition c)).

To permit the development of the requisite test techniques the following definitions are proposed:

A <u>secondary race</u> exists in a network between two state variables S_i and S_j if there is a stable state Y_k and an input variable U_c such that a single-variable change in U_c causes a change in both S_i and S_j.

Let the DIRECTORY schedule that, in the above circumstance, the network should reach the stable state Y_v. The secondary race in the network leads to a <u>critical</u> condition in terms of object delays if, by initiating a change in S_i subsequent to the change in S_j, the network reaches a stable state $Y_w \neq Y_v$, for any i or j.

The secondary race leads to a <u>non-critical</u> condition in terms of object delays if, under the above circumstances, $Y_w = Y_v$.

The secondary race leads to an <u>oscillatory condition</u> if, under any circumstances of relative delay between the state variables involved in the change, the network does not reach a stable state.

It will be noted that a secondary race set up between two state variables S_i and S_j will often involve other state variables of the network which respond to a change in S_i, S_j or both. Such variables may race between themselves or with either of the original racing pair. Taking into consideration all possible distributions of relative delay (i.e allowing no assumptions regarding the value of delays), starting from the initial stable state of V_k the network may traverse upon a critical path CP, non-critical path NCP or oscillatory path OSC, under the above definitions.

It will be worthwhile to discuss a single test in terms of a much larger DIRECTORY than that of Table 4.10.1. An extract from such a DIRECTORY for a network of two inputs and five state variables, is shown on Table 4.11.1. The network is stable in the state 100 and undergoes a single-variable change in U_2, which transfers the system to state 36. A secondary race is set up between S3 and S5, leading to a maze of paths as shown on Fig. 4.11.1.

The diagnosis is simple, in spite of the complexity of the diagram. If S3 changes before S5 does then the transition leads to one of two NC paths. If S5 changes first then the network is liable to mal-function. The output must warn the designer about the latter condition.

<u>4.11.2 - The test module L.</u>

The module concerns the diagnosis of secondary race conditions (CP and OSC) which render a sequential network sensitive to delay parameter changes.

<u>Inputs</u> for the module are provided by a DIRECTORY.

<u>Analysis</u> commences by finding all of the P number of stable

states and initiating n different single-variable changes, starting from each stable state. With the aid of the DIRECTORY multiple state-variable changes are located and followed through, considering each possible combination of relative delay among the state variables concerned. The paths so traced are then assessed against the next state variables of the multiple-change schedule. Output then will be an implied NO or a description of the initial stable state, the changing input variables and the state variables of the multiple change which lead to error, together with the critical relative delay. For instance the example of Fig. 4.11.1 gives rise to the printout

u = 11, S = 00100; u: = 10, s: = 00001

CP if TS3 > TS5

(here TS_i: the time delay associated with S_i)

Inputs and outputs are tabulated on Table 4.11.2.

4.11.3 - Comments.

Module L operates upon information contained in the DIRECTORY and requires no reference to object delay parameters. It detects all race conditions between state variables and is therefore a valuable and comprehensive test method. However, it uses a test technique which implies that the network responds to the change in an input variable before any state variable change is effective. This condition does not always apply; thus an additional test module is developed to deal with a type of delay sensitivity not diagnosed by module L.

When used as a tool of design verification, module L indicates to the designer how to arrange additional delays to favour non-critical paths. However, the error reports of module L may raise conflicting demands (S_i to be delayed more than S_j to eliminate one error but S_j more than S_i to eliminate another). In such cases the designer must re-consider the state assignment and modify the DIRECTORY itself. To avoid such waste of effort, the module L may be used to formulate the design by assessing the DIRECTORY containing the specifications or the proposed state assignment.

If a network has been subjected to module L and is proven to have no secondary races, that network will be said to have a DIRECTORY verified under module L.

4.11.4 - Further techniques and concepts.

The following definitions are now proposed:

An essential race exists in a network between an input variable U_i and a state variable S_j if there exists a stable state Y_k such that a single-variable change in U_i causes a change in S_j.

An essential hazard exists in a network which contains an essential race between U_i and S_j if a change in U_i followed by a change in S_j results in a stable state Y_v whereas a change in S_j followed by a change in U_i results in a stable state Y_w where $Y_v \neq Y_w$.

Although it is not possible to move a network from a stable state by changing a state variable, it is possible that the effect of an input variable change is so delayed that it reaches a part of the network later than the effect of the secondary change. If the terminal performance reveals sensitivity to such a condition, i.e. if the network is liable to reach a different steady state dependent upon the relative length of the delay of the input and secondary path, the network is said to have an essential hazard.

The alternative definition of essential hazards is that of Ungar (25) which does not permit easy appreciation of the cause of the hazard but provides a basis on which such hazards may be diagnosed. Ungar's definition is as follows:

> an essential hazard exists in a network if there is a total state Y_k and an input variable U_i such that, starting from the state of Y_k, three consecutive changes in U_i bring the network to a state other than the first input change.

Let essential hazards be demonstrated on the network whose DIRECTORY was shown in Table 4.10.1. As an example, consider the case when the initial stable state Y_k is that marked by reference number 10. A single-variable change in U_1 implies a change in S_1.

Using the definition proposed by Ungar, the first change in U_1 leads to stable state 15; the second change leads to stable state 9. The third change leads to state 13 $\neq$ state 15; thus, under conditions of $U_1 = 0$, $U_2 = 1$, $S1 = 0$, $S2 = 1$ and U_1 as the changing variable, the network contains an essential hazard.

Using the alternative definition proposed here and starting from state 10, the change in U_1 leads to 15 as before. Starting from the same state and changing S1 first, the stable state of 9 is reached; changing U_1 now leads to the stable state of $13 \neq 15$: the network contains an essential hazard.

Either of the two definitions may be accepted as a basis for the design of test module M.

Module M, as Module L, may be used to verify a PRIMITIVE DIRECTORY; it can also be used to analyse the behaviour specifications of a new design, allowing the detection of delay sensitivity.

<u>4.11.5 - The test module M.</u>

<u>Inputs</u> consist of a DIRECTORY.

<u>Analysis</u>, based on Ungar's definition, is as follows. Let the sequential network have P number of stable states and a number of inputs. Selecting one of the stable states (Y_1) as the initial condition, the network is subjected to a single-variable change in U_1 which leads to some stable state Y_2. Changing U_1 again leads to Y_3; a third change in U_1 leads to Y_4.

<u>Assessment</u> consists of comparing Y_1 and Y_4. If $Y_4 \neq Y_1$ then an error message is generated, specifying the initial condition Y_1 and the variable U_1 which leads to the essential hazard. The test is repeated for each of the P stable states and, within them, for each of the n inputs, amounting to a total of Pxn tests.

Outputs therefore consist of a list of error messages or else a straight NO, indicating that the network is insensitive to delay variations.

Inputs/outputs are shown on Table 4.11.3.

4.11.6 - Trivial cases of essential hazards.

Some of the essential hazards reported by module M can in fact never cause mal-functioning of the network. Such cases will be termed trivial essential hazards. An example will be shown here with reference to an extract of the DIRECTORY of a network with 5 inputs and 3 state variables. The network itself is shown on Fig. 4.11.2.

Let the network rest in the stable state 221; let the single-variable change be in U_3. Then, after transitions through states 253 and 252 the stable state of 254 is reached. The second change in U_3 transfers the network through 222 to state 223. The third and final change in U_3 leads through 255 and 251 to 249. Since 254 and 249 mark different states, an essential hazard is diagnosed. The hazard involves U_3 and the state variable S_1 (see table 4.11.4)

Applying the alternative algorithm and permitting the change in S_1 to precede the change in U_3, the network transfers from state 221 through state 220 to stable state 223. Here again, since 254 and 223 are different, an essential hazard is diagnosed.

The question is now: how likely is it that mal-functioning will occur in the circuit whose DIRECTORY is that of Table 4.11.4. In other words, what is the probability that in case of the circuit which gave rise to this DIRECTORY, the change in S_1 will reach a critical gate before the change in the input variable U_3 does?

Close examination of the network of Fig. 4.11.2 will reveal a race at gate 5 between a signal from gate 1 and another element 4000 (i.e. U_3 and S_1). Since the change in S_1 originates from gate 1 and encounters the delay $\tau 4$ of gate 4 before is is applied to gate 5, the signal change of gate 1 is bound to reach gate 5 ahead of this change by the time interval $\tau 4$. Without requiring the delay model of any of the gates of the network it can be stated for certain that the outcome of the race between U_3 and S_1 is not in doubt - the essential hazard is <u>trivial</u>.

Fig. 4.11.3 shows the critical gate 5 and the race paths. The delay model used for the interpretation of this diagram and later in this chapter associates the delay with the gate elements; no delay is assigned to paths themselves. Thus the delay of path U_3 is zero; the delay of path S_1 is that associated with gate element 4.

A general definition of trivial essential hazards is now proposed as follows. Let U_i and S_j be engaged in an essential hazard terminating at some system object D_k. Let T_i denote the set of delays associated with objects along the path of U_i and T_j the set of object delays along the path of S_j. Then the essential hazard between U_i and S_j is trivial if $T_i \subset T_j$. (In this case all the delays in the path of U_i are also present in the path of S_j which encounters some additional delays; consequently U_i always wins the race).

The definition is based upon the assumption that the network is resolved to a homogeneous structure of single gates; only then can the object D_k and the elements of sets T_i and T_j be identified. Module M operates at the lowest level of resolution where the network is given by its terminal model of the DIRECTORY; thus neither of the two definitions proposed in section 4.11.4 would permit distinction between trivial and non-trivial essential hazards. The test methods described in FURTHER DEVELOPMENTS allow this problem to be tackled. Meanwhile, designers must exercise some judgement in their response to the error messages of module M by ignoring trivial essential hazards.

4.11.7 - Transient performance of sequential networks and the VERIFIED DIRECTORY.

Modules K, L and M all concern transient testing of sequential networks. At this stage it will be useful to review these tests paying attention to concepts and not to techniques.

Combinational networks needed a single transient test: under the assumption of single-variable change and fundamental mode of working they needed to be tested only for static hazards, i.e. for signals originating from a single input.

Sequential networks need to be tested under the same conditions; however, the model of Fig. 4.8.1 implies that a single input change may originate the change in two or more independent variables - these variables are not in fact independent of the inputs except for the purposes of the model of Fig. 4.8.1 The problem is complex and, to help to see it clearly and solve it systematically, it is divided to three parts. The static hazard module K considers each independent variable as an input and seeks to find single-input hazards. These, when diagnosed, can be eliminated by redundant logic; hence such errors are of no further consequence. Hazards involving more than one independent variable are then further divided to secondary races (module L) and essential hazards (module M) depending upon the number of state variables involved. The total sequence of tests K, L and M covers all transient conditions. This means that a network passing these three tests with a clean bill of health is free of all transient hazards, its operation is not dependent upon object delay variations and its DIRECTORY is <u>VERIFIED</u>. Networks with static hazard (module K) errors will also be said to have a VERIFIED DIRECTORY so long as they have been verified under both module L and module M because static hazard errors can easily be corrected in every case. However, networks with module L or module M error reports need further testing: their DIRECTORY has failed verification under the circumstances specified by the error reports and, therefore, their DIRECTORY still contains some ambiguity. To disperse this, such networks must be resolved to a higher level, the delay model of network objects must be consulted and an ultimate decision made about their operation under the conditions specified by the error reports. Such detailed testing would then disperse the uncertainty of the performance and would allow the generation of a complete VERIFIED DIRECTORY.

4.12 - WHAT IS THE SIMULATED RESPONSE OF A NETWORK, SELECTED AT RANDOM FROM A GIVEN BATCH, TO A SPECIFIED INPUT CHANGE?

The question may be raised under several circumstances:

1) the designer wants to see the response of a network to some specific input sequence.
2) the designer seeks an estimate of the speed of operation of the network.
3) the designer wishes to gain a definite answer to question 4.8 and hence seeks to verify the DIRECTORY.

4.12.1 - Techniques.

For the first time in the course of this chapter a question can only be answered by the use of delay parameters. A simple delay model is proposed: the system object D_i, at the resolution level of a single gate, will have a delay parameter τ_i such that an excitation at a time t which demands a change of output of D_i will cause such a change to occur at a time $t + \tau_i$. The value of τ_i is to be computed by the use of a pseudo-random number generator which assigns a value to τ_i from a specified distribution. The distribution is assumed as normal, truncated at zero, for all objects; they can, however, carry individual parameters of mean delay μ and standard deviation σ. These two parameters must individually be declared for each object of the system as real integer multiples of a selected and common unit of time. The parameters may be declared when the network is first presented for verification. Alternatively, since verification procedures may not always require the use of delay parameters, these parameters may be declared in conjunction with the call for the appropriate test module.

The information regarding the network structure and function parameters is obtained for the simulation module from the output of module C. The structure data is re-shuffled to accommodate the "look ahead" method of simulation; the object D_i carries a list of other object identifiers indicating those objects whose inputs are connected to D_i. (Note that this is the opposite method to that used in module C - this time the list represents a column rather than a row of the structure matrix.) Thus only those object outputs are computed which are under the influence of a given signal change.

The simulator is asynchronous. A column is assigned to each variable whose output is monitored (this can include any object of the homogeneous structure). A row is assigned to each instant of time when any of the monitored outputs change. Thus time progresses in uneven intervals down the page. A column of printout specified the timing of each row to a number of digits of accuracy appropriate to real numbers. Each monitored variable is assigned a unique alphabetic character which is printed in each row for which that variable has a value of logical '1'. Otherwise the column is blank.

Simulation commences at a time $t < o$ when the network rests in one of its stable states as detected by module J. At $t = o$ an input change is originated and the numeral in the time column advances until either

a) the network reaches another stable state
b) the time parameter exceeds some pre-set limit of T_{max}.

In either case the simulation terminates. If another run of simulation is required then the procedure is repeated: the initial stable state is specified, an input change is initiated at $t = o$ and the response computed as before.

A sample of printout is shown on table 4.12.1 which was obtained when the network of Fig. 4.11.2 was simulated. The alphabetic identifier and the delay τ of each network object is listed on Table 4.12.2. The simulation tests the response to a change in U_3 and the network rests initially in the total state of

U1 = 1, U2 = 1, U3 = 0, U4 = 1, U5 = 1,
S1 = 1, S2 = 0 S3 = 1.

This transition has been reported as containing an essential hazard (see the discussion in section 4.11).

On the printout the changing variable(s) appear ringed. This is to allow the tracing of signal flow through the network. There are two rows with more that one ringed entry; this is due to the fact that the auxiliary object associated with a state variable has no time delay, hence a simultaneous change occurs at the output of objects 4 and 4000 and also at objects 6 and 6000.

4.12.2 - The test module N.

Inputs to the module are assembled from outputs of module C, module J (for sequential networks) and delay parameters of single-gate objects.

Analysis is by asynchronous simulation.

Assessment does not take place: the output is directly generated by analysis and takes the form of a waveform-like printout on a trivially transformed time-scale.

The summary of module inputs/outputs is shown on table 4.12.3.

4.13 - WHAT IS THE VERIFIED DIRECTORY OF THE SEQUENTIAL NETWORK?

It was discussed in section 4.11 that the DIRECTORY of the network is verified by definition if there are no error reports originating from module L and module M. In the presence of such error reports the VERIFIED DIRECTORY is defined by the response of a network of nominal delays. Since the response of such a network is only in doubt for those conditions specified by the above error reports, these are the conditions to be investigated by the use of the techniques of module N and for a parameter of $\sigma = 0$ for all gate objects. Monitoring points should include all dependent variables of the model of Fig.4.8.1. The simulation run must be terminated when any of the state variables changes. The output of module N must then be translated by the module P to a format of DIRECTORY entry. These entries, added to the entries of the PRIMITIVE DIRECTORY which were in no need of verification form the output of module P and also the VERIFIED DIRECTORY.

4.13.1 - The test module P.

Inputs and outputs of the module are summarised on Table 4.13.1.

Analysis is by simulation, starting in turn from each condition specified by the error report and terminating at the first secondary change. The rest of the analysis then consists of compiling the output.

Assessment is not applied: the output of the analysis is the output of the module.

4.14 - DOES THE VERIFIED DIRECTORY REPRESENT THE DESIRED PERFORMANCE.

The question is worded advisedly: the answer will, in fact, give the information which was sought in section 4.8, but could not be satisfactorily provided by module H.

4.14.1 - Concepts.

This perhaps the most important verification procedure uses no new techniques but relies upon a new definition of equivalence between the DIRECTORY of the behaviour specifications (denoted φ) and the VERIFIED DIRECTORY representing the actual performance (denoted ϕ).

Obviously $\varphi \equiv \phi$ if the two tables are, bit for bit within row for row, identical. However, this condition is not necessary: the transient performance might easily have been broken down to single state-variable changes by module P, thus obscuring the fact that the network performs as specified. As an example consider a two-row extract of the behaviour specifications φ of a fictitious network (Table 4.14.1). The state of ref. 0 is open to a secondary race; after analysis the VERIFIED DIRECTORY ϕ is found to contain the rows shown on Table 4.14.2.

Direct comparison, row for row, shows a discrepancy at the 0th row; however, the network is seen to reach the desired stable state of 3 after touching on state 2 and resolving the sudden, double change of state variables to two distinguishable steps. The equivalence of φ and ϕ would be easier to appreciate if the DIRECTORIES would not show the unnecessary detail of transient behaviour (made certain by verification) but would indicate, in a REDUCED DIRECTORY, the stable state to which a given state connects (assuming, as always, fundamental mode of working, i.e. no change of inputs until the stable state is reached). With the aid of the new concept of a REDUCED DIRECTORY equivalence between φ and ϕ is simply defined as the equivalence between φ_r and ϕ_r, row for row and bit for bit. Here φ_r and ϕ_r stand for REDUCED DIRECTORY of behaviour specification and VERIFIED DIRECTORY, respectively.

4.14.2 - Oscillations.

The REDUCED DIRECTORY of a simple network is shown on Table 4.14.3. Two of the states have no stable-state assignments: they are locked in an oscillatory cycle. Such cycles are detectable by noting that an n-input, m-state-variable network must reach its stable state in no more than 2^m number of transitions. If this did not happen then the appropriate entries of the DIRECTORY are marked OSCILLATORY.

4.14.3 - The test module Q.

Answer to 4.14 is now obtained on the basis of input data derived from module P and behaviour data as given in the form of a DIRECTORY φ.

Analysis now consists of tracing the entries of both DIRECTORIES until stable states are found. Thus a REDUCED DIRECTORY is compiled of both φ and ϕ.

Assessment is a comparison of the two.

Outputs now show the conditions in terms of inputs and state variables which lead to disagreement. The module does not report on oscillatory entry of the REDUCED DIRECTORY ϕ_r as an error if the behaviour specifications also prescribe oscillations.

Optionally and as input to some subsequent modules the REDUCED DIRECTORY of the sequential network is available.

The test is summarised in Table 4.14.4.

4.15 - IS THE SEQUENTIAL NETWORK UNCONDITIONALLY STABLE?

4.15.1 - Concepts.

The question probes an important feature. The answer, as provided by module R, is based on the following definition:

A sequential network will be termed unconditionally stable if its reduced DIRECTORY contains no OSCILLATORY entries.

It will be recalled that the VERIFIED DIRECTORY was compiled on the basis of nominal object delays (the parameter σ was set to zero); the REDUCED DIRECTORY was obtained by processing the VERIFIED DIRECTORY; thus unconditional stability refers to the network whose objects have nominal delay parameters. Since these delay parameters are, in practice, subject to statistical variations, a given network within a statistically related batch may prove oscillatory although the nominal-delay model representing the batch was found unconditionally stable.

This state of affairs is obviously very disturbing if the designer needs reassurance that no oscillations are likely to occur in any of the networks within the batch. Such reassurance can only be found by statistical assessment of the batch behaviour. Such assessment is outside of the scope of module R but will be undertaken by a subsequent module (module T).

4.15.2. - The test module R.

The module needs, as its only input, the reduced DIRECTORY which is produced as an optional output of module Q.

Analysis consists of sorting out the oscillatory entries.

Assessment is not required and output consists of listing the oscillatory entries, together with the conditions which gave rise to them (see Table 4.15.1.).

4.16 - IS THE SYSTEM LIABLE TO GET LOCKED IN AN UNDESIRABLE STATE OR SET OF STATES?

The usual concern is that, upon turn-on, the system may get into some state from which it can not be easily moved. Frequently the only way in which correct operation is induced is to switch the system on and off repeatedly until the desired conditions appear. Such circumstances are usually undesirable and often dangerous; thus designers should wish to be reassured that their design is not liable to this sort of behaviour. It will be recalled from chapter 3 that conventional test methods can not give such reassurance. While a decisive answer to question 4.16 is not available, module S is considered to produce a partial answer.

4.16.1 - Concepts.

The problem will now be discussed with reference to the $(n + m)$ dimensional sample space of a sequential network of n environmental signals and m state variables. The operating conditions will once more be modelled by a moving point Q whose co-ordinates in $n + m$ dimensions undergo discontinuous changes under the influence of binary signals.

Now let the n environmental signals be divided into n' physical signals and n" inputs. Physical signals contain such variables as supply potentials which are taken for granted under conditions of normal operation. Thus, under such conditions, the designer has n" degrees of freedom in choosing to move the point Q from one of the stable states of the system. After this, the point is liable to movement in m more dimensions of the state space, coming to rest finally in some new stable state.

This discussion wishes to lead to the realisation that, while the sample space has $(n' + n'' + m)$ dimensions, the designer is in direct control of only n" of these. Thus point Q is liable to be stuck within some undesirable area of the space and the n" degrees of freedom are not enough to move it out of there.

Let the problem be over-simplified by a model of Fig. 4.16.1. Here $n' = 2$ and consists of two power supplies E_1 and E_2. The Venn diagram of these two variables has four possible areas: $\overline{E}_1 . \overline{E}_2$ marks the condition when the system is switched OFF, $\overline{E}_1.E_2$ means that E_2 turns on before E_1, $E_1.\overline{E}_2$ means that E_1 turns on before E_2 and $E_1.E_2$ means the unlikely condition that E_1 and E_2 turn on at precisely the same time. Thus the point Q will rest within one of these areas at any instant in time.

If the system is switched OFF then point Q rests within $\overline{E}_1.\overline{E}_2$. Expanding Fig. 4.16.1 in (n" + m) dimensions, the position of Q within $\overline{E}_1.\overline{E}_2$ is given by (n" + m) parameters.

Now let the system be turned ON. There are three distinct paths possible for the point Q which will ultimately come to rest within $\overline{E}_1.E_2$, $E_1.E_2$ or $E_1.\overline{E}_2$. The position of Q can now no longer be shifted out of the chosen area except by switching the system OFF and then ON again in some other sequence. If it is now found that the system performance (the response to changes in n" dimensions) is not the same in the three areas then the designer is in difficulty.

Consider the same problem in (n" + m) dimensions only. Upon turn-on the designer can control n" number of independent variables but must be able to tolerate the possible drift of point Q in m dimensions. If there exists a group of stable states y_i such that no pattern of changes in n" input variables can move point Q outside of the states within y_i then the designer may be in the type of trouble described above.

Zadek (7) defines a connected and a strongly connected finite state system on the basis of system states. This definition is presented here in somewhat modified form. Let n" inputs and m state variables define a set of stable system states:

$$y = (Y_1, Y_2, \dots Y_k) \quad \text{where } k \leq 2^{n'' + m}.$$

Then the FSS is said to be connected with respect to Y_i if every state Y_j in y is reachable from Y_i with an input sequence of finite length. The FSS is said to be strongly or completely connected if it is connected with respect to every one of its states (in this case every state is reachable from every other state).

Connected and strongly connected FSS's will be seen as particular cases of Controllable and Completely controllable systems, respectively (see Chapter 2).

The problem is seen in this way: if the system is not completely connected then it is possible to envisage that, under the influence of the n' physical signals, it reaches some set of states which is not connected to the rest. Presumably such a system may contain some redundant state variables and minimising these would reduce the probability of the existence of un-connected states. However, systems with non-binary number of required stable states will always contain redundancy and it is often essential that the redundant states do not form a set un-connected to the others.

Question 4.16 will be re-phrased: is the system completely connected?

It is possible to design a test of connectedness on the basis of the REDUCED DIRECTORY. One possible test is outlined here. Since operating signal sequences may be designed in an infinite number of different ways, it is not possible to gain a deterministic answer in all cases of testing the connectedness; it is therefore necessary to qualify the findings by specifying the number of operating signal changes which were allowed. It is further necessary to utilise the allowed quota of changes to best advantage. Thus, operating signal changes must be chosen with care. Consideration will now be given to the most advantageous set of operating signal changes.

It will be remembered that, throughout the analysis, networks are considered to operate under conditions of single-variable change and in fundamental mode. Thus, unless the network is otherwise specified, there are n" number of equiprobable single-variable changes available to our network of n" operating signals, starting from each one of k stable states. This set of changes in operating signals will be called <u>first-order changes</u>; due to the symmetry of the circumstances it would be unreasonable to select a second-order change before all of the first order changes have been exhausted. Thus, if the designer specifies the maximum number of input changes in the test sequence as X and $X \geq n''$ then each of the possible single-variable changes will be applied. All the states of y reached by this sequence of tests are listed as connected to the starting state Y_i.

If, at any point during these tests, it is found that all the rows of the directory appear on the LIST OF Y_i CONNECTEDNESS then the test is discontinued and an output is printed to say that the NETWORK IS CONNECTED WITH RESPECT TO y_i. If, at the end of the test, a group of rows of the REDUCED DIRECTORY are still not found to be connected then a set of second-order changes may commence. The number of jth order changes is $(n'')^j$ and the total number of changes necessary to exhaust the jth and all lower order changes is X_j, where

$$X_j = \sum_{\ell = 1}^{j} (n'')^{\ell} \quad \text{for } 1 \leq \ell \leq j$$

Evidently the quota of X changes will be exhausted for relatively low order of changes and an error message will now be output to indicate the number of rows of the DIRECTORY which have been found un-connected to Y_i after X number of changes of operating signals.

<u>4.16.2 - The test module S.</u>

This simple test requires three <u>inputs</u>: the REDUCED DIRECTORY, the specification of the parameter X, giving the maximum number of changes allowed, and the description of the starting state Y_i.

<u>Analysis</u> is by manipulation of the DIRECTORY.

<u>Assessment</u> uses the DIRECTORY again, checking the total number of stable states k against the number of different states on the list of connectedness.

The <u>output</u> will give an answer of NO to question 4.16 if all stable states and all oscillatory states are found to be connected. Otherwise a list of states un-connected to Y_i, together with the description of Y_i is given.

The inputs/outputs are shown on Table 4.16.1.

4.17 - WHAT IS THE PROBABILITY OF DEVIATION OF THE PERFORMANCE FROM THE VERIFIED DIRECTORY?

The question demands the statistical assessment of the behaviour of a batch of networks whose structure and function parameters are the same but whose delay parameters are statistically related.

It will be recalled that certain entries of the VERIFIED DIRECTORY are not in doubt: only those conditions listed by module L and M are open to change due to delay variations, hence these will be the only ones scrutinised by module T.

It will be assumed that any change in the circumstances of the network under test (such as manufacturing tolerances, environmental changes, etc.) results in random variation of the time delay associated with the objects of the network. It will also be assumed that the delay variations are adequately described by the delay parameters μ_i and σ_i of each object D_i at the resolution level of a single gate.

4.17.1 - The test module T.

The module uses the same analysis technique as module N: a network is constructed whose delay parameters are chosen by a pseudo-random procedure and this network is then analysed under each condition specified by module L and M. The procedure is repeated a number of times given by a parameter X which, in this case, determines the batch size. The result of the analysis is assessed against the VERIFIED DIRECTORY and a failure record produced. The simplest form of the failure record is to give the number of failed tests relative to the number of tests conducted.

The test thus uses inputs consisting of the error reports of module L and M. The test conditions are specified by the parameter X and the behaviour data is given by the VERIFIED DIRECTORY of module P. The output is the numerical description of the failure rate.

The test is shown in the usual form on Table 4.17.1.

4.17.2 - Comment.

The crude methods of test T could do with refinement: the failure record may be complemented by assessing the significance level of the test; alternatively, the test may be replaced by a direct method of calculating the expected percentage failure.

These areas of investigation have low priority on the list of plans for further development. The reason for the lack of enthusiasm for these interesting projects is the lack of practical application. Designers at the present time have no reliable information about the expected delay of their network objects, and manufacturers of hardware have no facility, nor indeed interest in developing facilities, for the measurement of delay times of individual gates. Thus neither the types of distribution nor their parameters are likely to be available, thus no use could be made of the newly developed techniques. It is felt that the crudeness of the techniques of module T are well suited to the crudeness of the data they use.

4.18 - IS THE SEQUENTIAL NETWORK, SPECIFIED BY THE VERIFIED DIRECTORY, LIABLE TO GENERATE TRANSIENT OUTPUT SPIKES?

It will be assumed that the sequential network has been tested by module K and if static hazards have been detected then these were subsequently corrected before subjecting the network to module U. Thus if output spikes are present then these are systematically generated by the network. Such spikes may be due to a sequence of secondary changes following a single input change while the network settles into the final stable state (as given in the REDUCED DIRECTORY). Consequently the outputs must be monitored throughout each such sequence and if there is möre than one change in any output then a dynamically generated output spike is present. This is a design error which needs logical correction if the load supplied by the system is sensitive to such spikes. New techniques of analysis are not required since the test consists of observation of the VERIFIED DIRECTORY and the counting of the number of changes of each output in response to a single change of input.

4.18.1 - The test module U.

Inputs consist of the VERIFIED DIRECTORY.

Analysis requires n number of single-variable changes for an n-input network, and this number of tests must be conducted starting from each of the P stable states.

Assessment is by counting the output changes in response to a single test.

Outputs will list the test conditions which led to multiple-output change, together with the description of the faulty output concerned.

Inputs/outputs are shown on Table 4.18.1.

4.19 - SUMMARY.

The nineteen modules described in this chapter are connected by the same purpose: they were designed to answer questions of design verification in a concise way. Their operational objectives are also common: they use what is thought to be the minimum of information consistent with generating the answer, they aim at comprehensive testing and utilise techniques which are efficient in terms of computer use. The modules differ in their mode of analysis: they use predominantly numerical methods but some of the techniques may be best described as analytic; others, such as those towards the end of the chapter, are based on simulation. Table 4.19 has been constructed to allow the assessment of the service provided by each of the modules.

Some of the modules succeed in giving what is thought to be complete and satisfactory answers to the design verification question they serve. Others produce incomplete, deficient or over-pedantic answers. Care has been taken to point out such deficiencies, explain the reason for their existence and indicate some way in which the module may be re-designed at some later time to minimise these deficiences.

The modules are compatible and frequently rely upon each other for test data, inputs or technique. It is evidently possible (even desirable) to design a control program which automates the use of a set of these modules. The design of such automatic test systems is briefly discussed in chapter 5. The design and implementation of such a system is the subject of a research project (1) which has been set out in conjunction with this work.

CHAPTER 5.

ORGANISATION OF A MODULAR LOGIC DESIGN TEST SYSTEM.

5.1 - INTRODUCTION.

The purpose of this chapter is to outline the manner in which the modules of Chapter 4, or their variants, may be combined to form a system of logic design verification. The design of such a system would require careful consideration of the need and facilities of the industrial organisation which undertakes the implementation. While it is possible to give general outlines of systems design, a particular system must suit the computer installation, the design methods, the staff training facilities, etc., of the user concerned. The work involved is outside the scope of this thesis.

In order to substantiate, indeed in many cases to develop, the techniques described in Chapter 4, it was necessary to undertake the development of a Prototype System. This development, as mentioned earlier, is the subject of an allied research project (1). This chapter will contain the outline of the aims of the Prototype System. Reference (24) is a published report describing its organisation and methods. The material of the report will form the basis of Mr. Holmes' thesis.

Fig. 5.1 shows the block diagram of the design verification process. The designer may be required to present some or all of the types of data shown. The Proposed System consists of a selection of the test modules of Chapter 4, complemented by some control mechanism which allows them to be operated upon the receipt of relevant data.

The next section outlines two alternative types of design verification systems. The Prototype System is akin to one of these and will be compared with the Proposed System in the course of this chapter.

5.2 - SYSTEM DESIGN ALTERNATIVES.

5.2.1 - CONVERSATIONAL SYSTEM.

The type of system which would afford the greatest flexibility and the shortest turn-around time may be conversational. The time-shared processor would provide simultaneous service to several logic designers who would have a terminal in their own office, and would thus enjoy immediate access to design verification facilities.

The time-shared system appears to be simple to organise. The designer would be presented with a set of seemingly autonomous modules of which to choose. Module manuals would give instructions about the type and format of data required by the module. The service would need a common backing file from which the designer may select standard hardware sub-systems. Furthermore, individual files would have to be provided for each designer.

At the level of the processor the modules are not autonomous; some are dependent on others for their data; several may use similar analysis techniques. Thus the system would need organisation where a call for a given module would activate not only the nominated module but all necessary auxiliary modules as well.

The system must provide diagnostic facilities. These may include syntactical checks of the network data, instructions and behaviour specifications, as well as more sophisticated checks, such as compatibility between network and specifications and demands of storage capacity.

An important part of the verification system design is the selection of standard verification modules. The 19 modules of Chapter 4 make a formidable list; designers would prefer a smaller selection perhaps. The restriction of the list by the omission of duplicated or un-important facilities is obviously desirable. However, restriction by combining several facilities would amount to reduction of the resolution level of the system and may deprive the designer of some important detail. It is considered that detail must be preserved in a conversational system, otherwise less than full advantage will be taken of expensive facilities. Thus, for such a system, a comprehensive list of modules is recommended, omitting of the list of Table 4.19.1 only module F (or module G), module H (to be included as auxiliary of several others) and module P$'$ (this too is an auxiliary).

5.2.2 - Batch processing system.

This alternative pre-supposes that the designer has no immediate access to the computer, hence the verification process must be conducted in a smaller number of larger steps than in the case of conversational systems. Thus it is prudent to reduce the number of test modules and anticipate that the designer may call several modules of the list at a given run in an attempt to reduce the over-all turn-around time.

For such a system a list of modules is proposed as shown on Table 5.2.1.

When initiating a verification run, the designer must prepare a universal data tape which includes the list of nominated modules and all relevant data. This data therefore contains more detail than any module may require. The system must provide facilities for sorting the data for each nominated module.

If the list of nominated modules contains inter-dependent modules then the sequence of operations must be automatically ordered to ensure that auxiliary modules are operated before dependent ones. Apart from this, the designer must be given the freedom to nominate dependent modules without calling their auxiliaries, as in conversational systems.

5.3 - THE PROTOTYPE SYSTEM.

The Prototype System may be regarded as an example of the type of system described in section 5.2.2. Since however, the system was used as a test bed of ideas on verification and programming techniques, its facilities are less comprehensive and frequently less sophisticated than those of the modules of Chapter 4.

The purpose of the design of the Prototype System was four-fold:

1) to help the development of design verification techniques
2) to prove the feasibility of a modular design verification system

3) to give experience in the design of such a system
4) to provide an operative system for the use of logic designers.

It was considered important to select a modest computer installation as the basis for such a system: if the newly developed techniques were to gain widespread acceptance, expensive installations could not be taken for granted. Thus the Prototype System was designed for a machine with a small core store (16k) and three tape handlers as the only backing storage.

For the purpose of generality the programs of the Prototype System have been written substantially in ALGOL. However, to permit efficient use of the computing facilities, certain sections, amounting to some 15% of the total, are written in machine code. An example of such a machine code section is the use of the bit-handling routines of ISOPACK.

To allow the evaluation of the facilities of the Prototype System, let all of the modules of Chapter 4 be combined to form a "Proposed System". The Prototype System will now be evaluated against this standard.

In the forthcoming discussions references to the report of (24) will carry the prefix "A".

The terminology and notation of (24) will be seen to differ somewhat from that of the rest of this work. Thus for instance, "module" in this work has so far denoted a design verification procedure, whereas the report uses this word to describe a sub-system of the total system of programs. Furthermore, the word "Analysis" in the title of the report is used to mean "design verification".

A reference to figure A 2.1 will show the Prototype System to consist of 10 program modules. Of these, INPUT and LIBRAN perform ancillary duties while the rest are test modules in the usual sense of the word.

5.3.1 - The test modules of the Prototype System.

Table 5.3.1 shows a comparison of facilities. It will be seen that, while the Prototype System lacks certain facilities, it provides others not found in the range of the Proposed System.

The facilities provided for combinational networks are identical, with the only exception that module D has no equivalent in the Prototype System.

Important differences occur in terms of sequential networks. There are no facilities in the Prototype System to compile a VERIFIED and a RESTRICTED DIRECTORY, although ANALYS (section A3.9) and TIMED (section A.3.9) contain most of the ingredients. Several important decisions are made upon information contained in the PRIMITIVE DIRECTORY but, since this is an ambiguous record of the network performance, these decisions must be handled with caution. Thus, for instance, the list of UNSTABLE entries of Fig. A5.3.3 do not necessarily mean that the network oscillates since, due to conditions detected by modules L and M, the VERIFIED DIRECTORY may significantly differ from the PRIMITIVE. Similarly, the count of transitionary cycles as reported by ANALYS (same section) carries information of doubtful value.

Another facility within ANALYS is CIRCUIT OPERATIONS (section A3.6.2.5). This is an interesting feature, providing waveform-like outputs on an un-specified time scale. This too is based on the PRIMITIVE DIRECTORY and is subject to the same comment as above.

SERIAL is not so much a separate test module but a mode of using SEQUEN and ANALYS for networks which would otherwise cause embarrassment due to the size of their logical (designation number) arrays. The installation for which the Prototype System was designed imposes a limitation that, if the number of independent variables exceeds some parameter (15 in this case), then comprehensive testing is not attempted, but networks will be analysed by the nominated test modules only under the input conditions defined by the designer.

The facility is not required in the Proposed System, since such large networks can be automatically analysed under a restricted set of conditions. Instead of specifying the test data, designers need merely to define the admissible environment. The method is considered preferable to that of the Prototype System.

It will be observed from Table 5.3.1 that the module ANALYS performs several functions. This does not result in reduced resolution because, in fact, ANALYS is divided to several small modules (see section A3.6). Similarly, STATIC is entered on Table 5.3.1 twice; this is because the same module is used in the Prototype System for the analysis of combinational and sequential networks.

The report contains a User's Guide (section A5.1) as well as several worked examples within section A5. From there the output format may be observed to be at variance with the Proposed System in several details.

5.3.2 - Conclusions.

The Prototype System succeeded in providing the basis for a more sophisticated system of design verification; it has also established undergraduate work on logic network analysis at Kingston Polytechnic. The system has been used by logic designers in Industry whose response, after an initial period of aversion from anything new, has been favourable.

5.4 - FUTURE PLANS.

It is considered that the research work contained in this thesis and in the allied project (1) is a basis upon which development of a design verification system could be founded. Support is now sought, and has been promised, for such work to be carried out under a sponsored development contract. The new system is to be designed to allow the inclusion of the new facilities and the improvement of techniques outlined in Chapter 6.

CHAPTER 6

FURTHER DEVELOPMENTS

6 - FURTHER DEVELOPMENTS.

The implementation of the test methods described in this thesis is now under consideration by a Government Department and by industrial organisations. In addition, the experience gained by this work is now applied in several new fields of research, some of them not connected to logic design.

It is the purpose of this chapter to describe some of the new fields of investigation to which this project has led.

6.1 - AUTOMATION OF THE ITERATIVE DESIGN OF LOGIC NETWORKS.

Reference is made to the model of the design process, shown on Fig. 2.6.1.

The correction loop of the figure will now be examined in more detail.

When the partial specifications are presented, the designer generates a proposal of a new design which represents the best solution he is able to offer to the problem. When the verification process results in an error report, the designer initiates corrections of the design and offers a new version for verification. The procedure is shown on Fig. 6.1.1.

The methods described in chapters 4 and 5 of this thesis proposes to automate the verification process, but demands that the designer performs the tasks of generation of new design and error correction.

It is possible to envisage a fully automatic design system where each of the tasks is performed by the computer. As a first step towards such a system, the error correction process may be automated, leaving the designer with the job of generating the initial design. Research is carried out at the present time, under a

Government contract, to automate the error correction process. A modular error correction procedure is under development, with a correction module matching each verification module within a design test system. The correction module interprets the error report produced by verification and modifies the network in a systematic way, thus eliminating the error.

The final stage of automation would bring the generation of the design proposal under the computer's control. This step has now been planned.

6.2 - MODELLING OF LARGE LOGIC NETWORKS.

The homogeneous structure of single-gate resolution (Chapter 4) becomes untenable if the network under test is very large. The designer has no coding problems when handling such large networks, because the facility of nested structures permits him to operate at much lower levels of resolution. The same facility must be extended to the computer.

The solution lies in automating the process of restricted-mode modelling. A restricted mode model is defined as a model of a network whose behaviour is the same as that of the network itself if the environment of the model is constrained to a part of the admissible environment of the network. As a consequence of such constraints it is generally true that the restricted mode model contains less information than the comprehensive model of the homogeneous structure and thus is more efficient to store. Since it also operates at a lower level of resolution, it offers additional efficiency by reducing the time of computer manipulation.

It is possible to design several different methods of restricted mode modelling. One of these methods is now pursued under a Government contract.

6.3 - THE ANALYSIS OF HYBRID SYSTEMS.

For the purpose of this discussion a hybrid system is defined as one containing both analogue and digital sub-systems; more precisely, a hybrid system may be resolved to objects whose environment is modelled as a continuous space and other objects whose environment is modelled as a number of discrete points.

The analysis of hybrid systems presents particular problems due to the difficulty to interface objects of dissimilar type. A method of analysis is therefore sought which will simultaneously accommodate all of the objects of a hybrid system.

A Government-sponsored post-graduate student is at the present time investigating this problem under the supervision of the author.

6.4 - QUALITATIVE ANALYSIS OF THE TRANSIENT BEHAVIOUR OF LOGIC NETWORK.

The term qualitative analysis is defined here as the investigation of the behaviour of a system without reference to input signals.

It is thought possible to find the causes of faulty or ambiguous transient behaviour of logic networks on the basis of such qualitative analysis. It is further considered that it is possible to initiate the correction of some types of transient errors, entirely on the basis of qualitative observations.

The method eliminates the shortcomings of the verification modules of Chapter 4: it permits the elimination of a statically hazardous gate from the error list of module F or G if an effective blanking signal is found to be available at a subsequent part of the network; it permits the sorting of trivial and non-trivial essential hazards and allows the identification of the gates causing sensitivity to object delay variations.

Although the improvement of design verification services is significant, the main advantage offered by the qualitative analysis method lies in its efficiency. The number of tests necessary for comprehensive analysis is a linear (instead of the usual exponential) function of the number of inputs. Thus, by replacing some of the modules of Chapter 4 by others, based on qualitative analysis, it becomes possible to analyse very large networks comprehensively.

Methods of qualitative analysis are now partially developed and a paper is under preparation describing the use of these methods for the analysis of combinational networks.

6.5 - AUTOMATION OF PRODUCTION TESTING OF LOGIC SYSTEMS.

The subject is of considerable topical interest. A selection of recent publications can be found in ref. (28).

The problem could briefly be outlined as follows: product testing has become the critical step in the production process of both discrete-component and integrated circuit logic systems. It is thus imperative that product testing should be efficiently conducted.

A product test method is efficient if it detects and classifies faults in the hardware by applying a minimal test sequence. Since the minimal sequence will be different for each design, a generalised method is sought for the automatic generation of such test sequences, based on the description of the verified design.

It is thought that the design test methods described in this thesis could form a suitable foundation for a production test method. Arrangements are being made at the Kingston Polytechnic for the appointment of a post-graduate research assistant who will work on this subject.

PRINCIPLE REFERENCES.

(1)	D.R. Holmes:	M.Phil. Thesis. (Under development)
(2)	A. Blumstein:	"The Choice of Analytical Techniques in Cost-effectiveness Analysis. Research Paper P-206, Institute of Defense Analyses, Washington, D.C., October 1965.
(3)	R. Deutsch:	System Analysis Techniques. Prentice Hall, 1969.
(4)	G. Gordon:	System Simulation. Prentice Hall, 1969.
(5)	J. Klir & M. Valach:	Cybernetic Modelling. Iliffe, 1965.
(6)	R.W. Newcomb:	Concepts of Linear Systems and Controls. Brooks/Cole Publishing Co., 1968.
(7)	L.A. Zadeh & E. Polak:	Systems Theory. McGraw Hill, 1969.
(8)	A. Gill:	Finite State Machines. McGraw Hill, 1962.
(9)	P.H. Rigby:	Models in Business Analysis. Merrill, 1969.
(10)	R.D. Buzzell:	Mathematical Models and Marketing Management. Harvard University, 1964.
(11)	H. Chestnut:	Systems Engineering Tools. J. Wiley, 1965.
(12)	T.H. Lee, G.E. Adams, W.M. Gaines:	Computer Process Control. J. Wiley, 1968.
(13)	I.G. Wilson & M.E. Wilson:	Information, Computers and System Design. J. Wiley, 1965.
(14)	R.E. Kalman:	Canonical Structure of Linear Dynamical Systems. Proc. Natn. Acad. Sci., U.S.A., 1962.

(15)	R.E. Kalman, P.L. Falb, M.A. Arbib:	Topics in Mathematical System Theory. McGraw Hill, 1969.
(16)	B. Porter:	Synthesis of Dynamical Systems. Nelson, 1969.
(17)		Information Processing Machines. Research Inst. of Math. Machines, Prague, 1968.
(18)	E.J. McCluskey:	Introduction to the Theory of Switching Circuits. McGraw Hill, 1965.
(19)	D. Zissos, G.W. Copperwhite:	Logical Design Manual. Pitman, 1968.
(20)	D. Lewin:	Logical Design of Switching Circuits. Nelson, 1968.
(21)	P.C. Gaston, S.P. O'Byrne:	Logic Simulation. Plessey Research Report No. T.M.54, 1968.
(22)	D. Leevers:	Users Manual for the LASS Logic Simulator. Research Report 70/55/A. D560/00104, Marconi-Elliott Computer Systems Ltd., May 1970.
(23)	J.S. Reynolds:	A Conversational Logic Simulator. CAD Conference, I.E.E., April 1969.
(24)	D.R. Holmes:	PROLOG - Logic Network Analysis System. Kingston Polytechnic Research Report, June 1970.
(25)	Ungar:	Hazards and Delays in Asynchronous Sequential Switching Circuits. IRE Trans. Circuit Theory, CTG, 1959.
(26)	Z. Kohavi:	Switching and Finite Automator Theory. McGraw Hill, 1970.
(27)	K.S.H. Halstead:	ISOPACK. KCT internal Research Report, March 1968.
(28)		I.E.E. Conf. on Automatic Testing. University of Birmingham, April 1970.

(29)	D.D. Schurmann, K. Maling:	A Statistical Approach to the Computation of Delays in Logic Circuits. I.E.E. Trans-Computers, April 1969.
(30)	W.H. Kautz:	The Necessity for Closed-Circuit Loops in Minimal Combinational Circuits. I.E.E.E. Trans. Computers, February 1970.
(31)	A.A. Kaposi:	Logic Systems Analysis, Symposium on CAD, Northern Poly-technic, 1969.
(32)	A.A. Kaposi:	Logic Testing by Simulation. CAD Conference, I.E.E., April 1969.
(33)	A.A. Kaposi:	Notes on Computer-aided Design of Logical Networks. CAD., Vol. 2, No. 1, Autumn 1969.
(34)	A.A. Kaposi, D.R. Holmes:	Logic Network Analysis, CAD., Autumn 1970.
(35)	A.A. Kaposi, D.R. Holmes:	Transient Analysis of Logic Networks. C.A.D., (Under publication)
(36)	F. Stevenson:	An Introduction to LOGIC, Extract from LOGIC Users' Manual, May 1968.
(37)	F. Stevenson:	Design and Simulation of Digital Networks. C.A.D. Conference, Sheffield Uni-versity, April 1968.
(38)	R.F. Wells:	Users' Notes DA 70. (Plessey Company), 1969.
(39)	R.G. Hicks:	Survey of Logic Simulation. B.T.R. Research Report No. 97/69/42/TR, July 1969.
(40)	R. George:	Programs for Logic Simulation, STL Report No. R.660/RGG/met, April 1970.

BOOK OF TABLES AND FIGURES

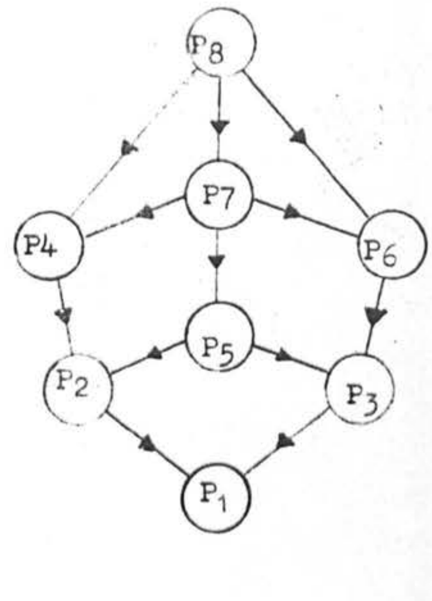


Fig.2.3.1.

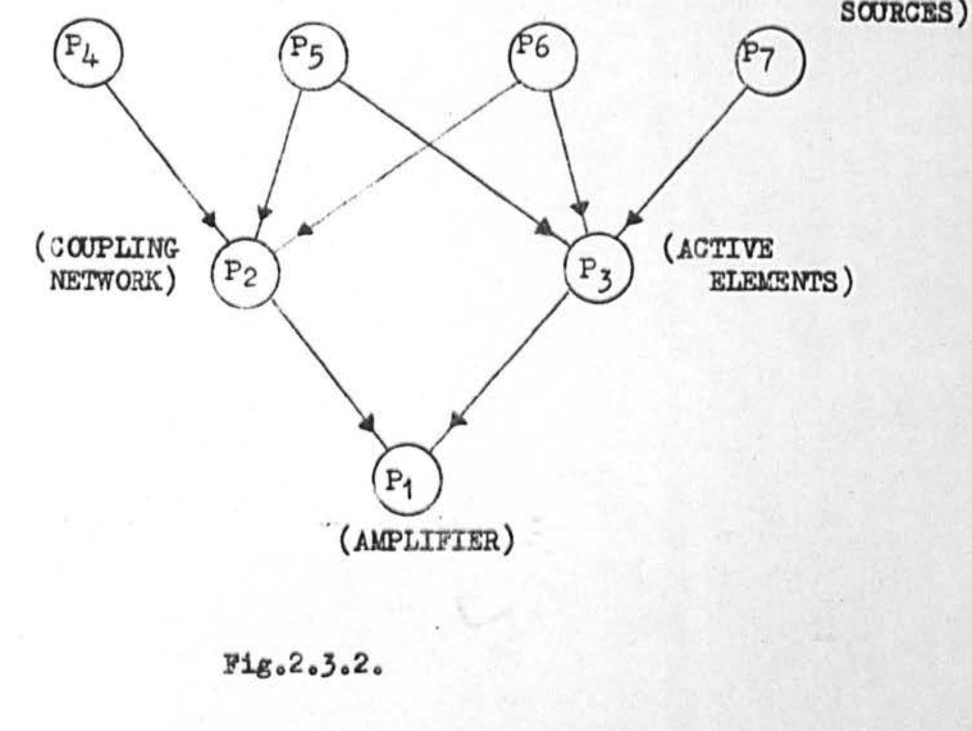


Fig.2.3.2.

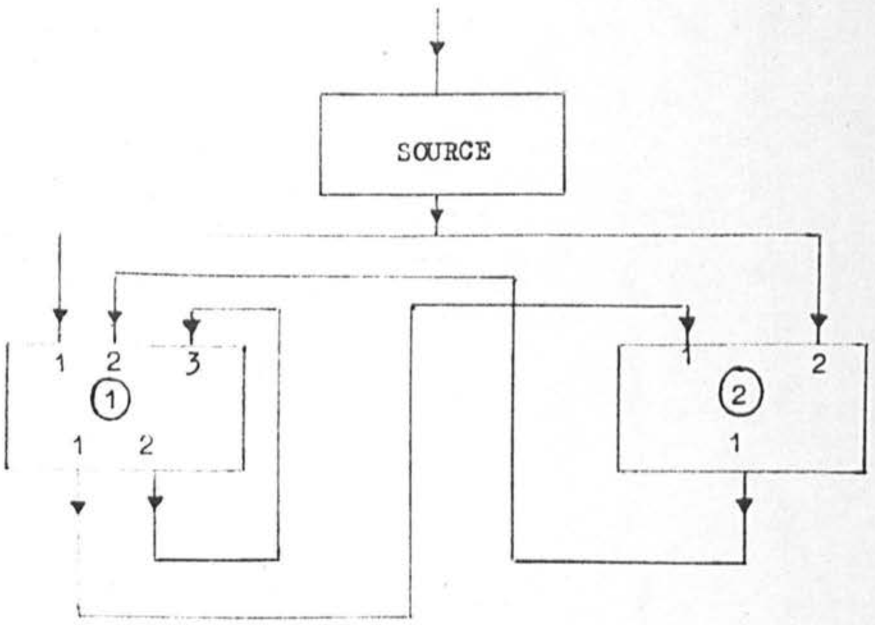


Fig.2.3.3.

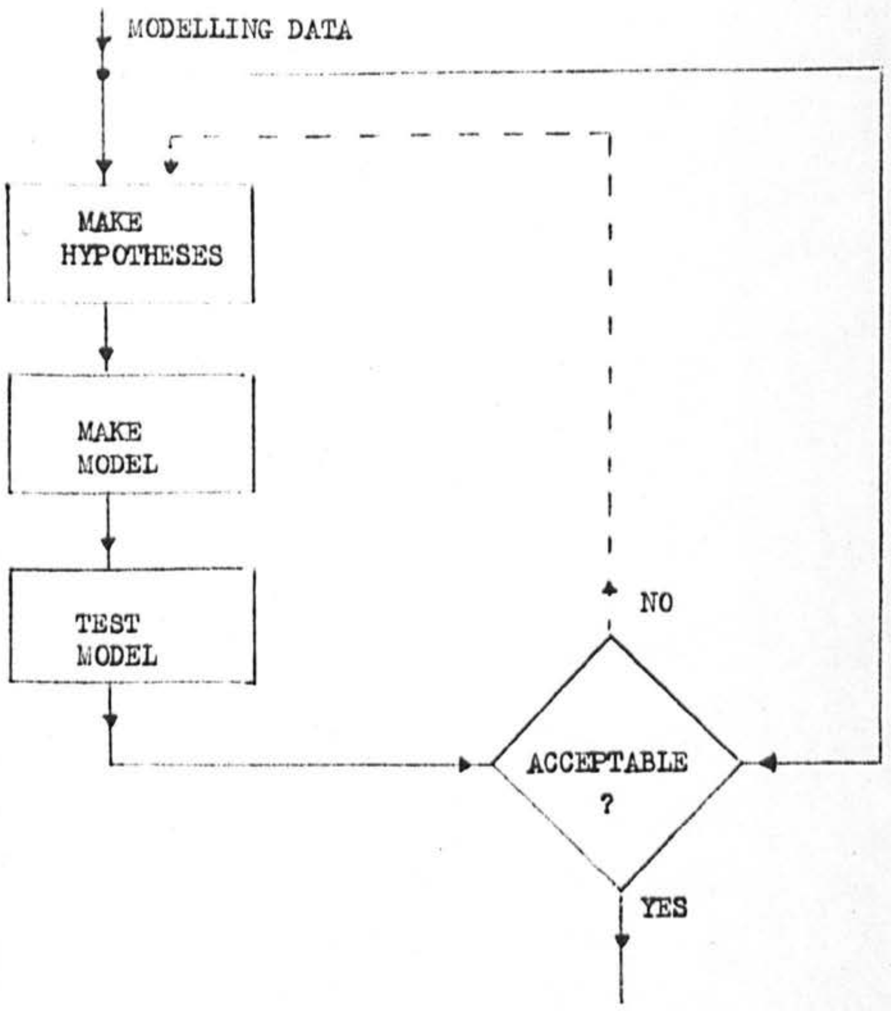

MODELLING DATA
MAKE HYPOTHESES
MAKE MODEL
TEST MODEL
NO
ACCEPTABLE ?
YES

Fig.2.4.1.

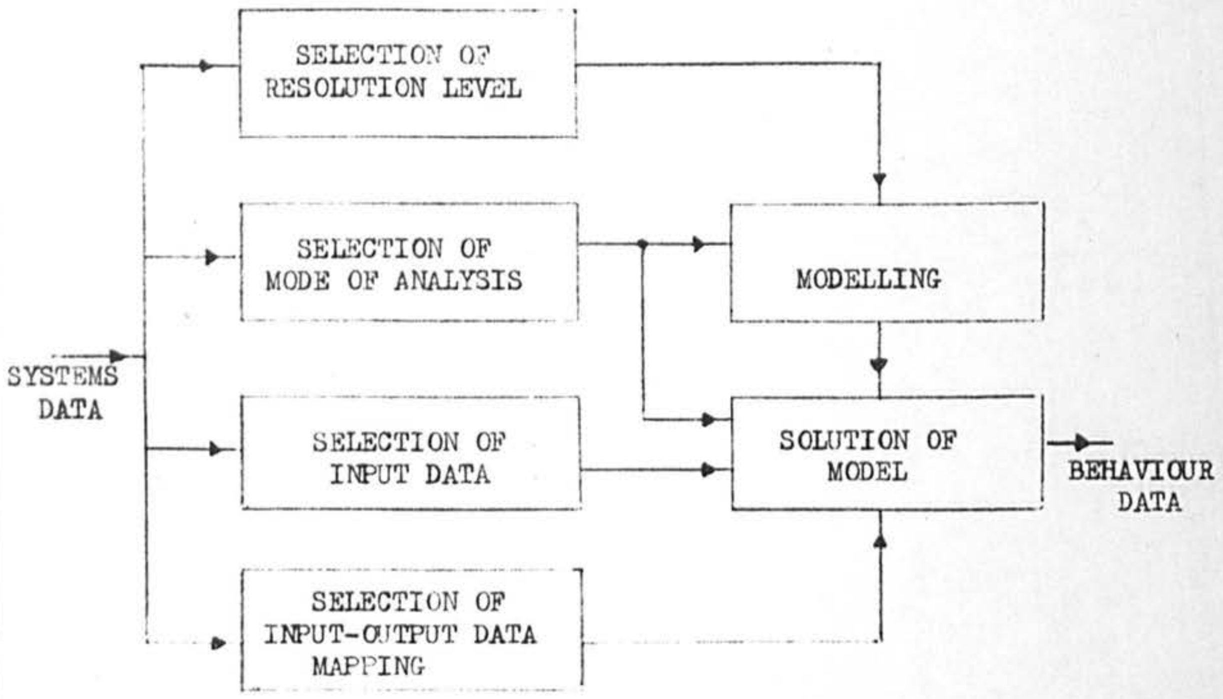

SYSTEMS DATA
SELECTION OF RESOLUTION LEVEL
SELECTION OF MODE OF ANALYSIS
SELECTION OF INPUT DATA
SELECTION OF INPUT-OUTPUT DATA MAPPING
MODELLING
SOLUTION OF MODEL
BEHAVIOUR DATA

Fig.2.5.1.

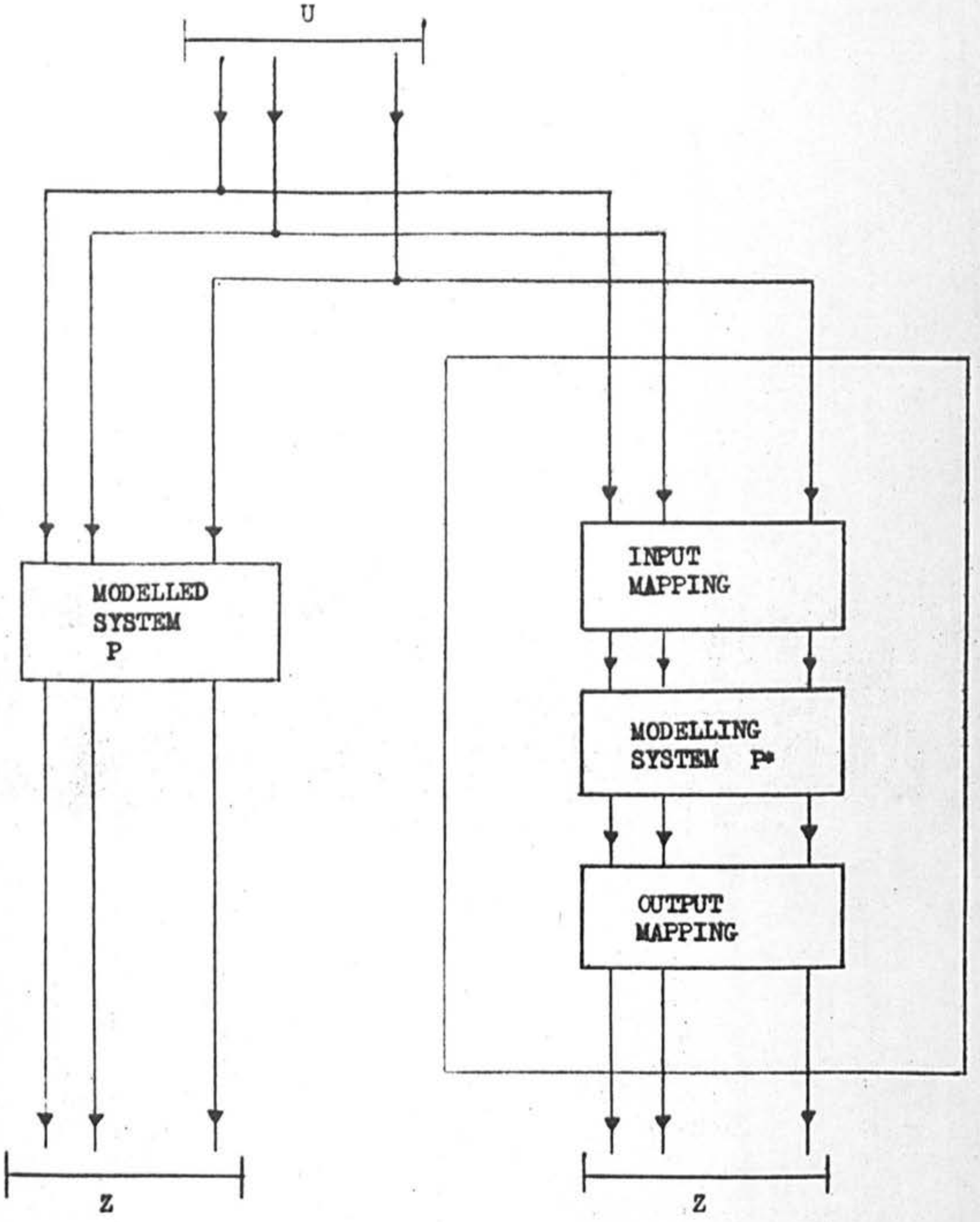

U
MODELLED SYSTEM P
INPUT MAPPING
MODELLING SYSTEM P*
OUTPUT MAPPING
Z
Z

Fig.2.5.2.

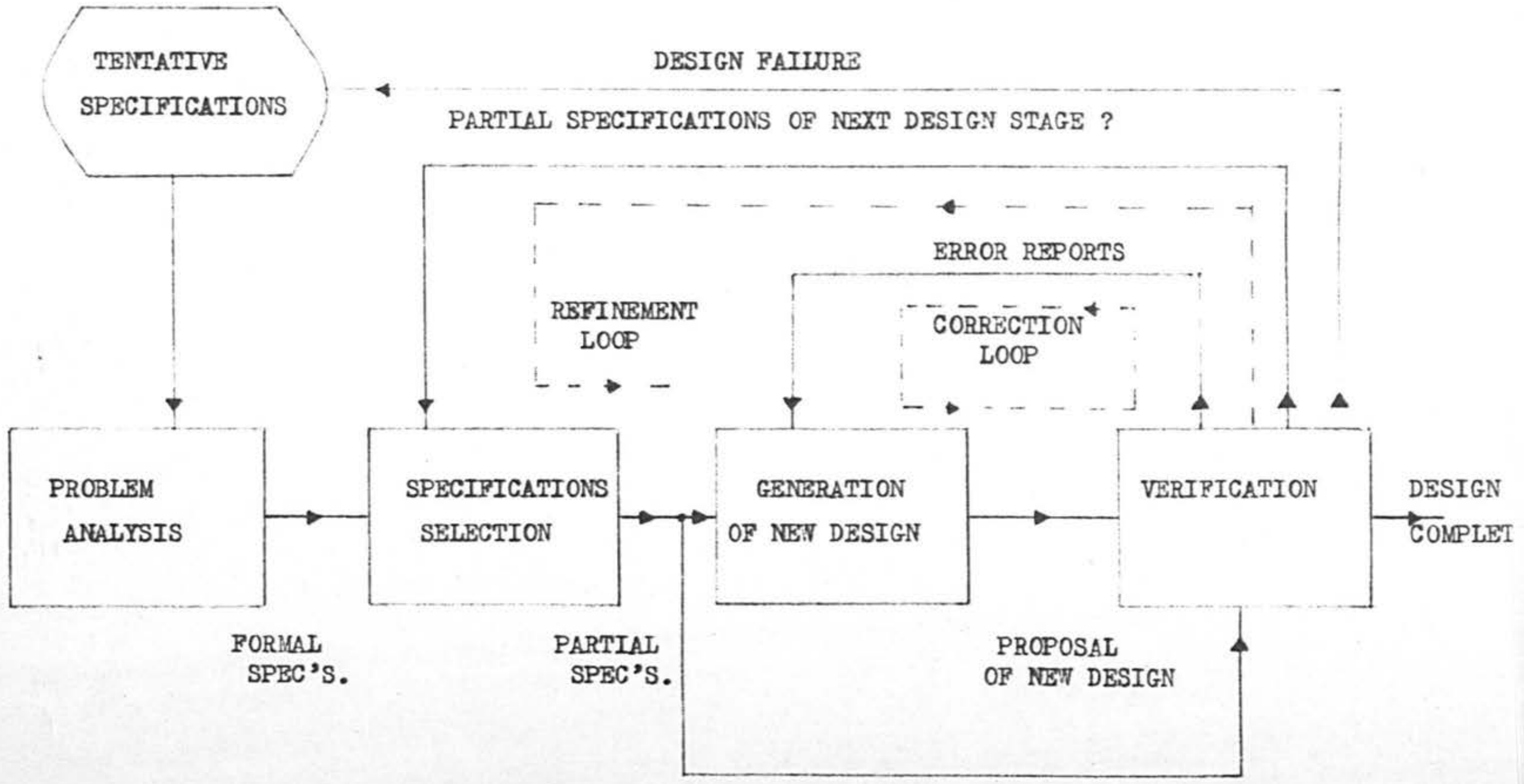

TENTATIVE SPECIFICATIONS
DESIGN FAILURE
PARTIAL SPECIFICATIONS OF NEXT DESIGN STAGE ?
ERROR REPORTS
REFINEMENT LOOP
CORRECTION LOOP
PROBLEM ANALYSIS
SPECIFICATIONS SELECTION
GENERATION OF NEW DESIGN
VERIFICATION
DESIGN COMPLET
FORMAL SPEC'S.
PARTIAL SPEC'S.
PROPOSAL OF NEW DESIGN

Fig. 2.6.1.

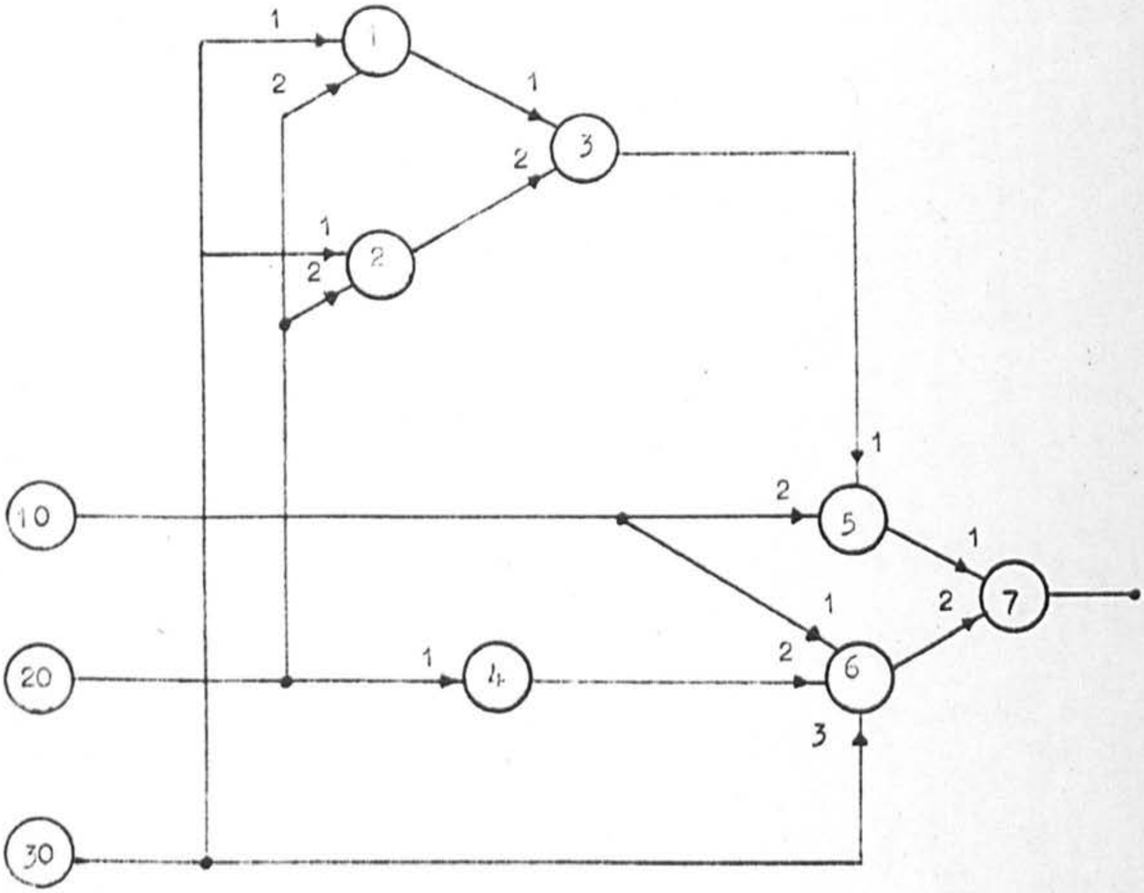


Fig.4.2.1.

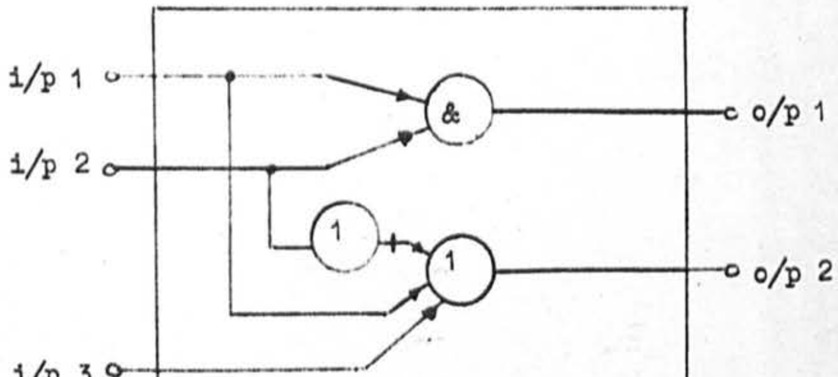


Fig.4.2.4.

$$\begin{array}{rcccccccccccc}
 & & 10 & 20 & 30 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \\
10 & (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
20 & (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
30 & (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
1 & (& 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
2 & (& 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
3 & (& 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 &) \\
4 & (& 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
5 & (& 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 &) \\
6 & (& 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 &) \\
7 & (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 &)
\end{array}$$

Fig. 4.2.2.

$$\begin{array}{rcccc|ccccccccc}
 & & 10 & 20 & 30 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \\
1 & (& 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
2 & (& 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
3 & (& 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 &) \\
4 & (& 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
5 & (& 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 &) \\
6 & (& 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 &) \\
7 & (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 &)
\end{array}$$

Fig. 4.2.3.

$$\begin{array}{c|ccc|ccccccc}
 & 10 & 20 & 30 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
5 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
6 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}$$

Fig. 4.2.5.

$$\begin{array}{c|ccccccc|ccc}
 & 10 & 20 & 30 & 1 & 2 & 3 & 5 & 4 & 6 & 7 \\
\hline
1 & & & & & & & & 0 & 0 & 0 \\
2 & & & & & & & & 0 & 0 & 0 \\
3 & & & & & & & & 0 & 0 & 0 \\
4 & & & & & & & & 0 & 0 & 1 \\
5 & & & & & & & & 0 & 0 & 0 \\
6 & & & & & & & & 1 & 0 & 0 \\
7 & & & & & & & & 0 & 1 & 0
\end{array}$$

Fig. 4.2.6.

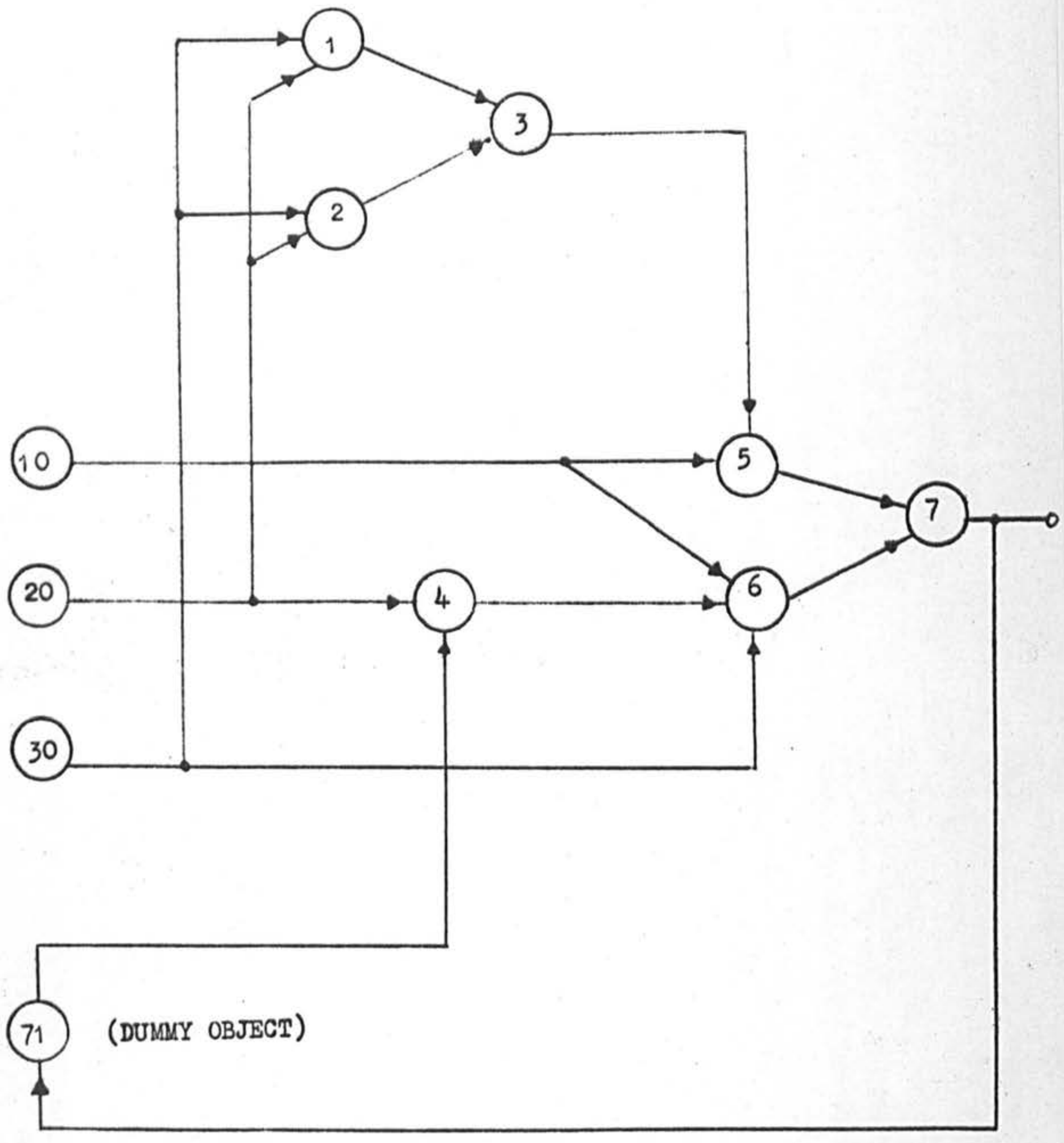


Fig.4.2.7.

$$
\begin{array}{rc|cccc|ccccccc|c}
 & & 10 & 20 & 30 & 71 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \\
\hline
1 & (& 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
2 & (& 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
3 & (& 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 &) \\
4 & (& 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
5 & (& 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 &) \\
6 & (& 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 &) \\
7 & (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 &) \\
71 & (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 &)
\end{array}
$$

Fig. 4.2.8.

```
(10)
(20)
(30)
( 1  20  30)
( 2  20  30)
( 3   1   2)
( 4  20)
( 5   3  10)
( 6  10   4  30)
( 7   5   6)
```

Table 4.2.1.

MODULE A

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Structure	-	-	YES/NO Option:- Processing List

Table 4.2.2

$$\begin{array}{c|ccc} & 4 & 6 & 7 \\ \hline 4 & 0 & 0 & 1 \\ 6 & 1 & 0 & 0 \\ 7 & 0 & 1 & 0 \end{array}$$

Fig. 4.3.1.

$$\begin{array}{c|cccccccc} & P & Q & R & S & T & U & W & X \\ \hline P & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ Q & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ R & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ T & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ U & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ W & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ X & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Fig. 4.3.2.

$$\begin{array}{c|cccccc}
 & P & R & S & T & W & X \\
\hline
P & 0 & 0 & 1 & 0 & 0 & 0 \\
R & 1 & 0 & 0 & 0 & 0 & 0 \\
S & 0 & 1 & 0 & 0 & 0 & 0 \\
T & 0 & 0 & 1 & 0 & 0 & 1 \\
W & 1 & 0 & 0 & 1 & 0 & 0 \\
X & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

Fig. 4.3.3.

MODULE B

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Structure	-	-	Nº of state variables or List of objects with which state variables are associated. Processing List

Table 4.3.1.

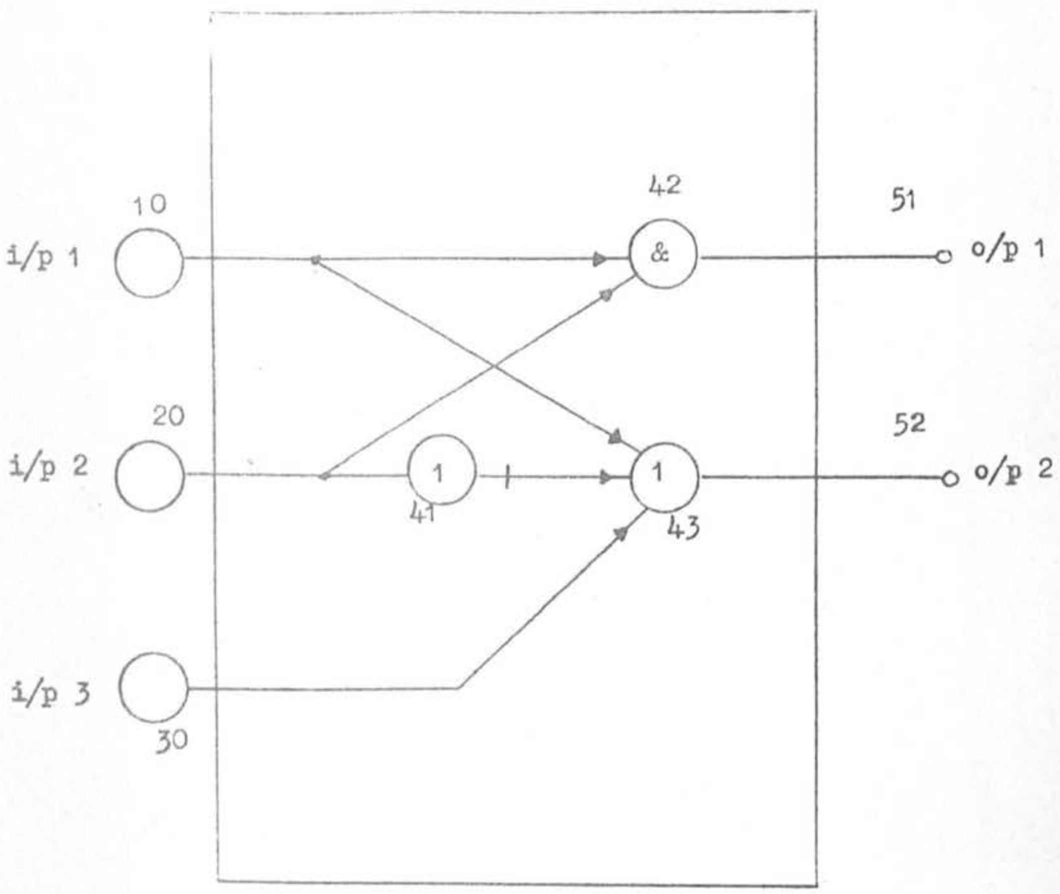


Fig.4.4.1.

```
(CIR3)    /title/
(1  INPUT  1)
(2  INPUT  2)
(3  INPUT  3)
(4  INPUT  4)
(5  INPUT  5)
(6  INPUT  6)
(7  INPUT  7)
(8  INPUT  8)
(9  INPUT  9)
(100  CIR2  1  2  3)
(200  CIR2  4  5  100/2)
(300  CIR2  6  7  200/2)
(400  CIR2  8  9  300/2)
(10  OUTPUT  1  100/1)
(11  OUTPUT  2  200/1)
(12  OUTPUT  3  300/1)
(13  OUTPUT  4  400/1)
(14  OUTPUT  5  400/2)
```

Table 4.4.1.

```
(CIR2)
(1  INPUT  1)
(2  INPUT  2)
(3  INPUT  3)
(10  NOR  1)
(20  NOR  2)
(30  NOR  3)
(40  NOR  100/1)
(50  NOR  300/1)
(100  CIR1  1  2  10  20)
(200  CIR1  30  40  100/1  3)
(300  CIR1  3  100/1  1  2)
(4  OUTPUT  1  200/1)
(5  OUTPUT  2  50) .
```

Table 4.4.2.

```
(1  INPUT  1)
(2  INPUT  2)
(3  INPUT  3)
(4  INPUT  4)
(10  AND  1  2)
(20  AND  3  4)
(30  NOR  10  20)
(5  OUTPUT  1  30)
```

Table 4.4.3.

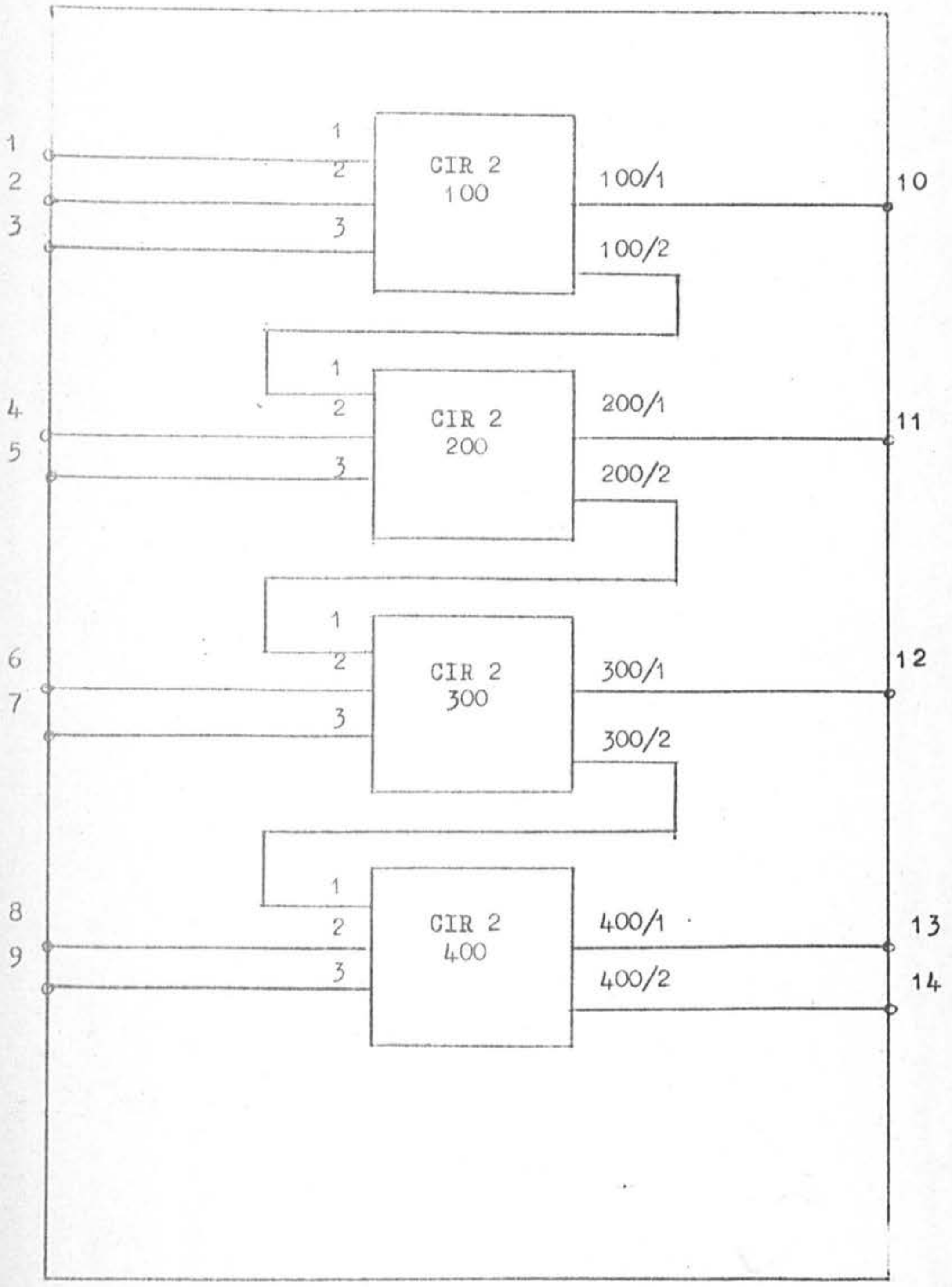

1
2
3
4
5
6
7
8
9
CIR 2
100
CIR 2
200
CIR 2
300
CIR 2
400
100/1
100/2
200/1
200/2
300/1
300/2
400/1
400/2
10
11
12
13
14

Fig.4.4.2.

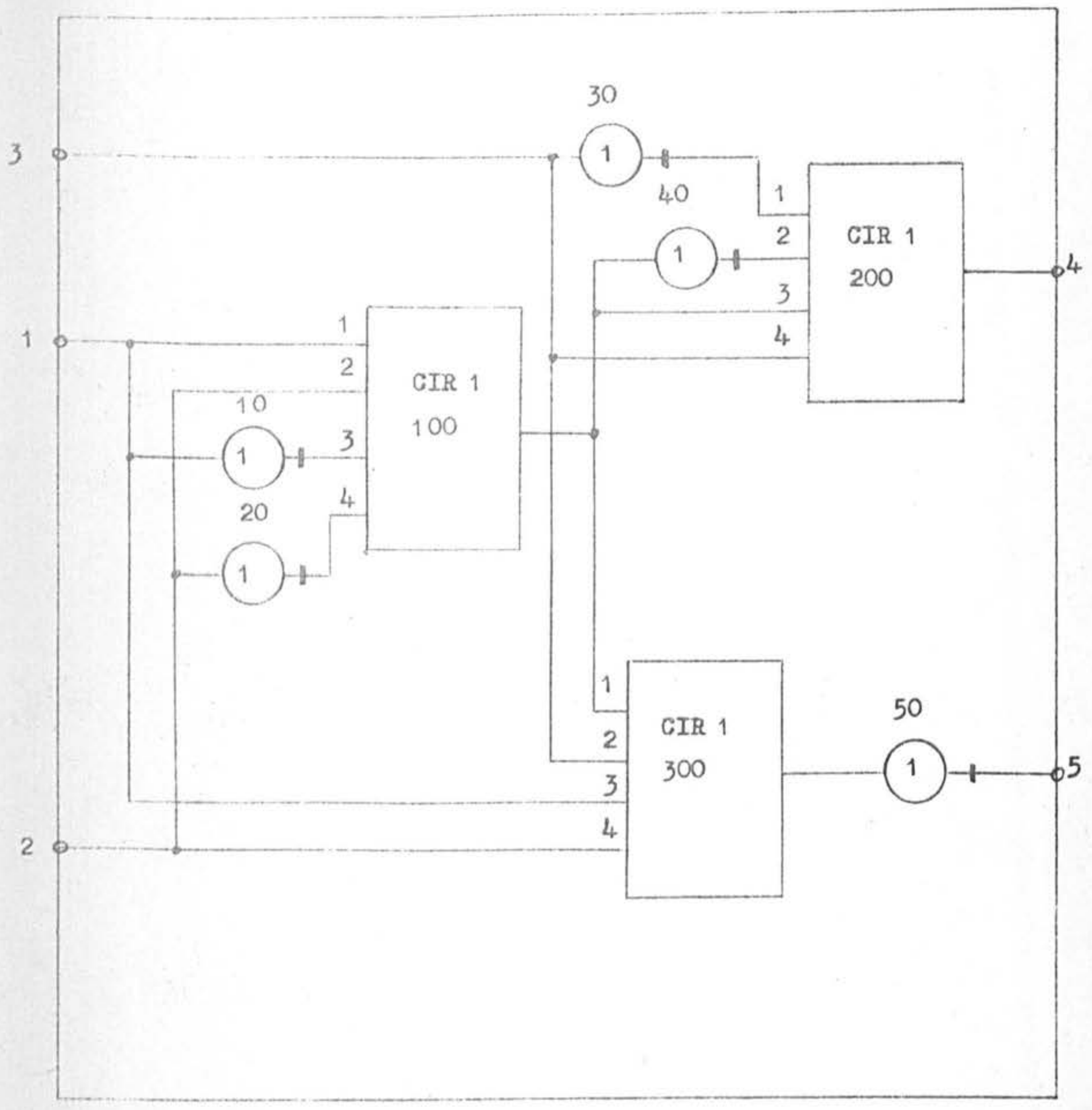


Fig.4.4.3.

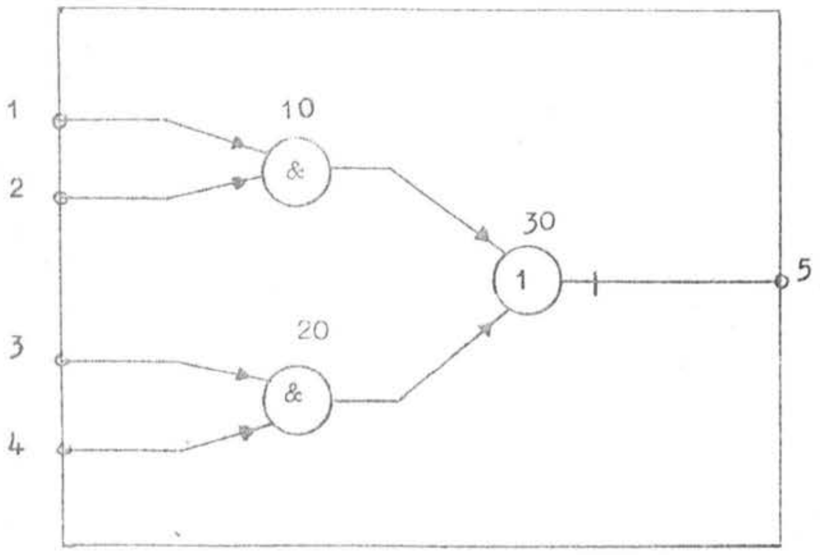


Fig.4.4.4.

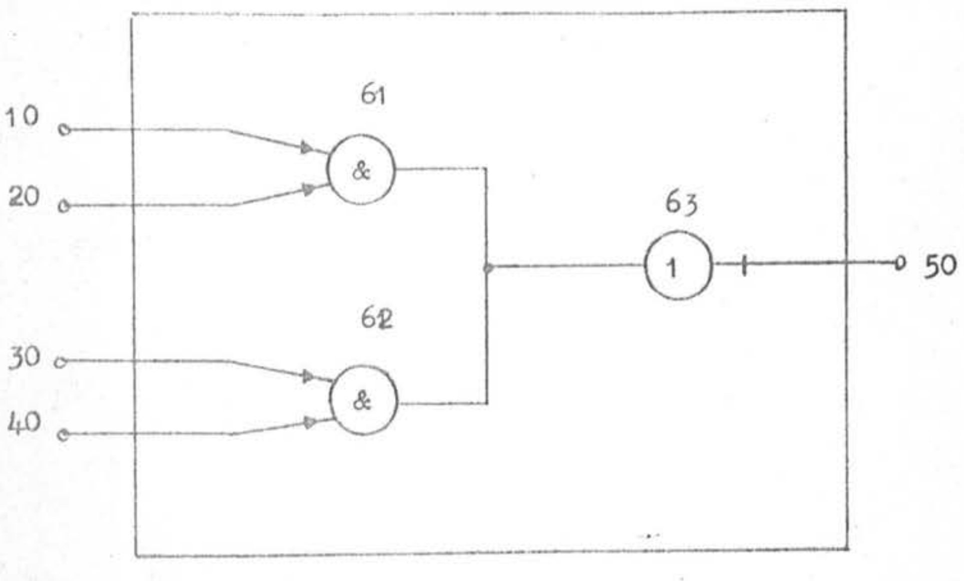


Fig.4.4.5.

MODULE C

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Structure + function parameters	–	–	YES/NO Homogeneous data structure; same with function parameters

Table 4.4.4.

MODULE D

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Structure + load parameters	–	–	NO/identification of faulty object and type of error

Table 4.5.1.

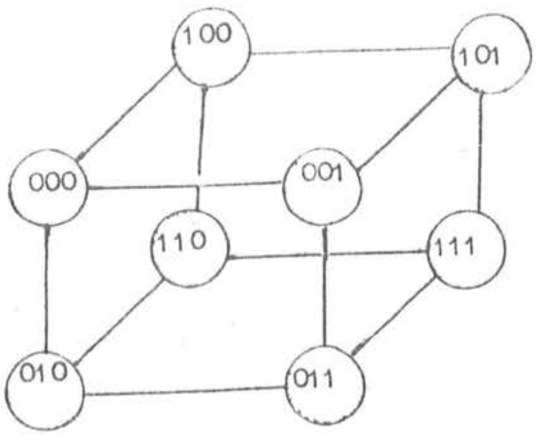


Fig.4.6.1.

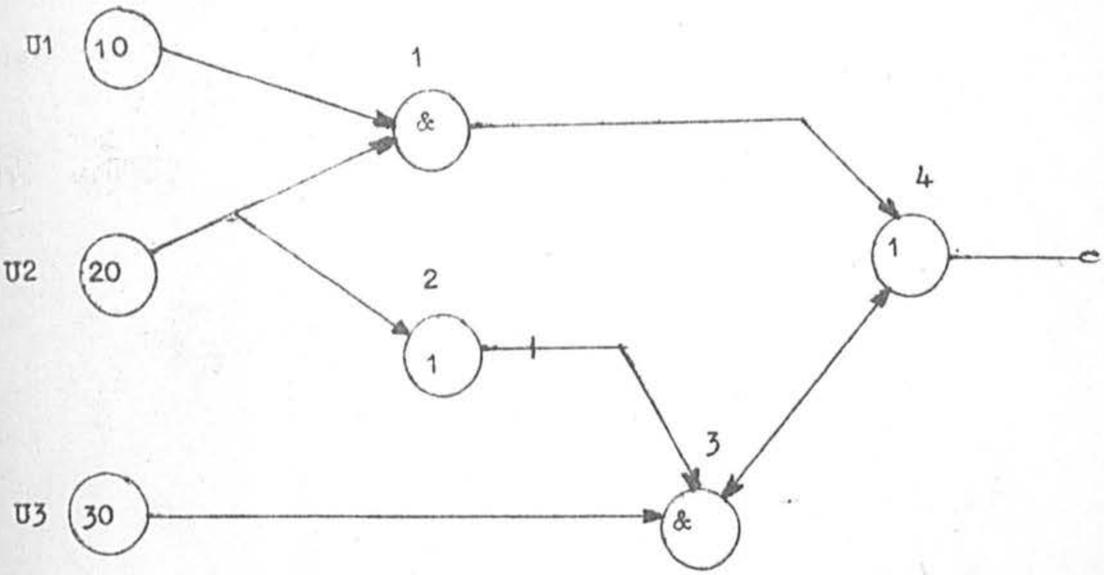


Fig.4.6.2.

dU_1	= 0 1 0 1 0 1 0 1
dU_2	= 0 0 1 1 0 0 1 1
dU_3	= 0 0 0 0 1 1 1 1
d1	= 0 0 0 1 0 0 0 1
d2	= 1 1 0 0 1 1 0 0
d3	= 0 0 0 0 1 1 0 0
d4	= 0 0 0 1 1 1 0 1

Table 4.6.1.

MODULE E

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Structure + function parameters + Processing List	Optional; FIXED or tabulated inputs	Standard network or Equations or Truth table	YES/ list of faulty output bits. Optional: total table

Table 4.6.2.

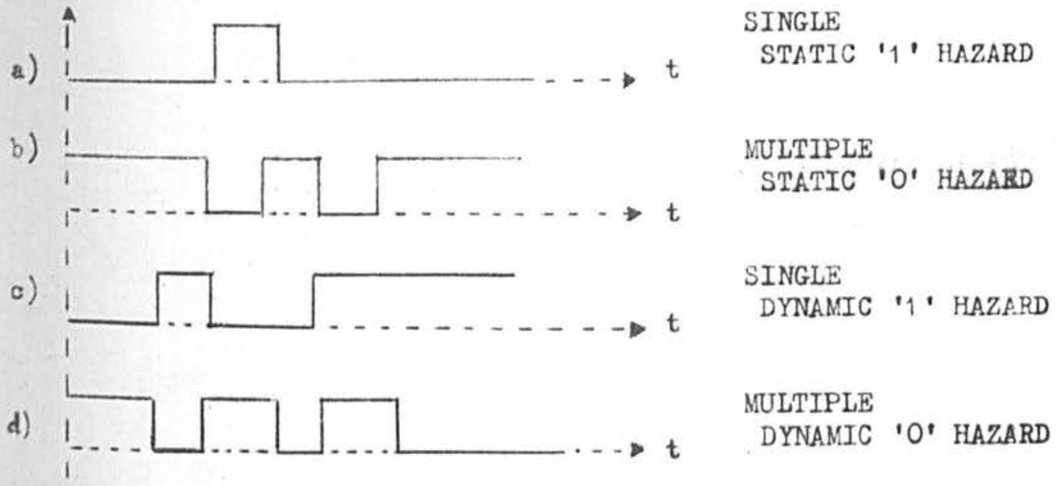


Fig.4.7.1.

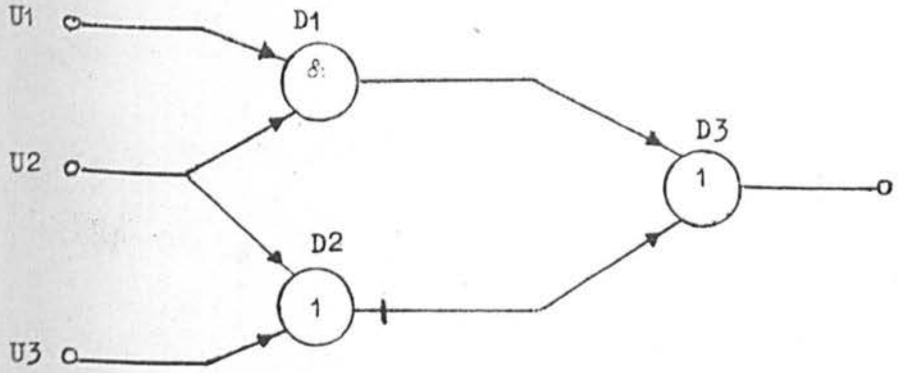


Fig.4.7.2.

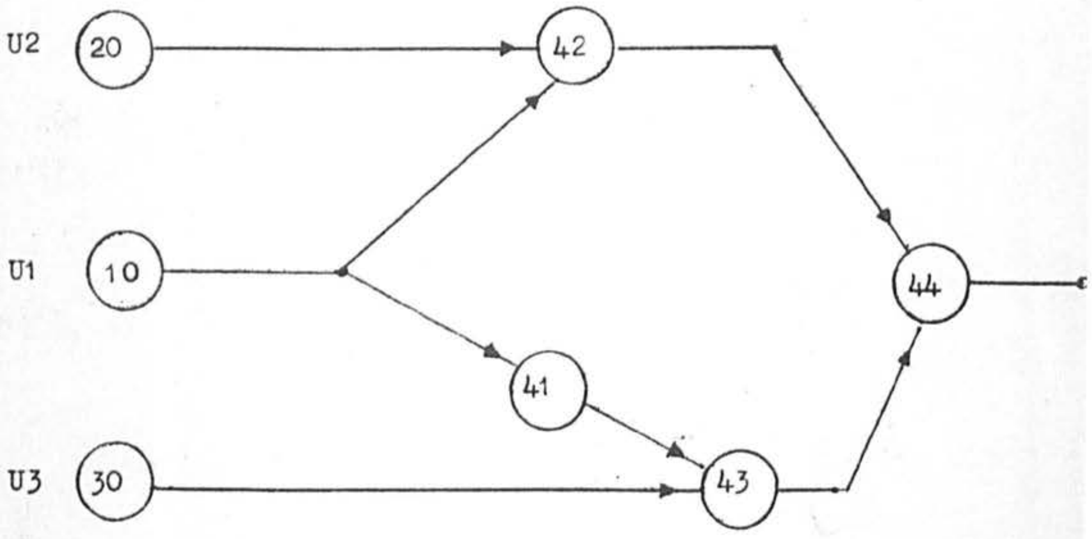


Fig.4.7.3.

U1	U3	$\frac{\Delta D3}{\Delta U2}$
0	0	$0.U2 + \overline{U2 + 0} = 0 + \overline{U2} = \overline{U2}$
0	1	$0.U2 + \overline{U2 + 1} = 0 + \overline{1} = 0$
1	0	$1.U2 + \overline{U2 + 0} = U2 + \overline{U2}$
1	1	$1.U2 + \overline{U2 + 1} = U2 + \overline{1} = U2$

Table 4.7.1.

MODULE F

Specification of new design	Specification of operating signals	Specification of behaviour	Output
structure + function parameters + Processing List	-	-	No/list of hazardous gates; Option:definition of i/p leading to hazard

Table 4.7.2.

Ref.	1	2	3	41	42	43	44
0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0
2	0	1	0	1	0	0	0
3	1	1	0	0	1	0	1
4	0	0	1	1	0	1	1
5	1	0	1	0	0	0	0
6	0	1	1	1	0	1	1
7	1	1	1	0	1	0	1

Table 4.7.3.

0	→	1,	2,	4
1	→	0,	3,	5
2	→	3,	0,	6
3	→	2,	1,	7
4	→	5,	6,	0
5	→	4,	7,	1
6	→	7,	4,	2
7	→	6,	5,	3

Table 4.7.4

0	→	1,	2,	4
1	→	–,	3,	5
2	→	3,	–,	6
3	→	–,	–,	7
4	→	5,	6,	–
5	→	–,	7,	–
6	→	7,	–,	–
7	→	–,	–,	–

Table 4.7.5.

Gate No.	0→1	0→2	0→4	1→3	1→5	2→3	2→6	3→7	4→5	4→6	5→7	6→7
10	0 1	0 0	0 0	1 1	1 1	0 1	0 0	1 1	0 1	0 0	1 1	0 1
20	0 0	0 1	0 0	0 1	0 0	1 1	1 1	1 1	0 0	0 1	0 1	1 1
30	0 0	0 0	0 1	0 0	0 1	0 0	0 1	0 1	1 1	1 1	1 1	1 1
41	1 0	1 1	1 1	0 0	0 0	1 0	1 1	0 0	1 0	1 1	0 0	1 0
42	0 0	0 0	0 0	0 1	0 0	0 1	0 0	1 1	0 0	0 0	0 1	0 1
43	0 0	0 0	0 1	0 0	0 0	0 0	0 1	0 0	1 0	1 1	0 0	1 0
44	0 0	0 0	0 1	0 1	0 0	0 1	0 1	1 1	1 0	1 1	0 1	1 1

Table 4.7.6.

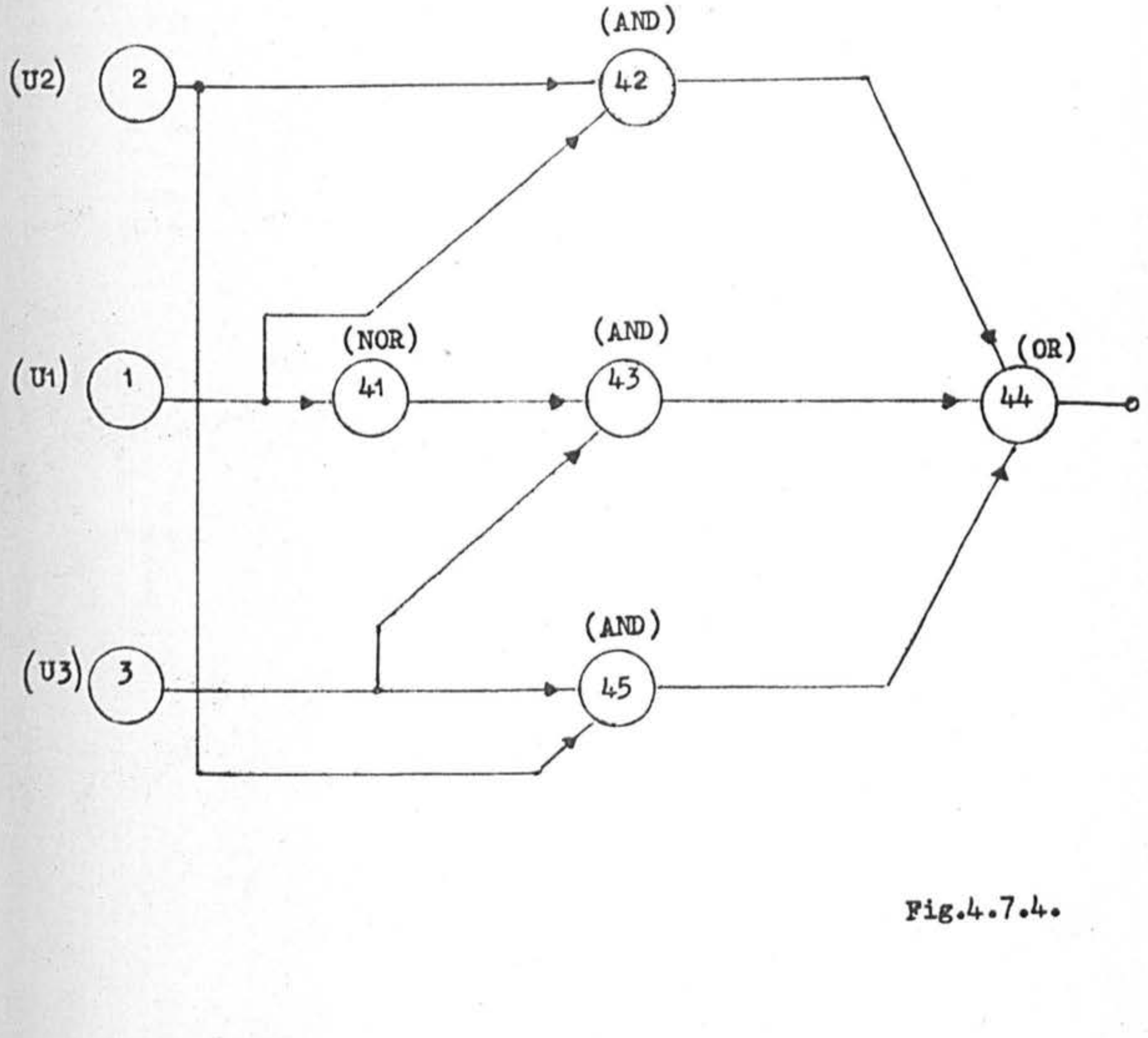


Fig.4.7.4.

MODULE G

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Output of module E + Processing List	–	–	No/list of hazardous gates; Option : definition of i/p leading to hazard

Table 4.7.7.

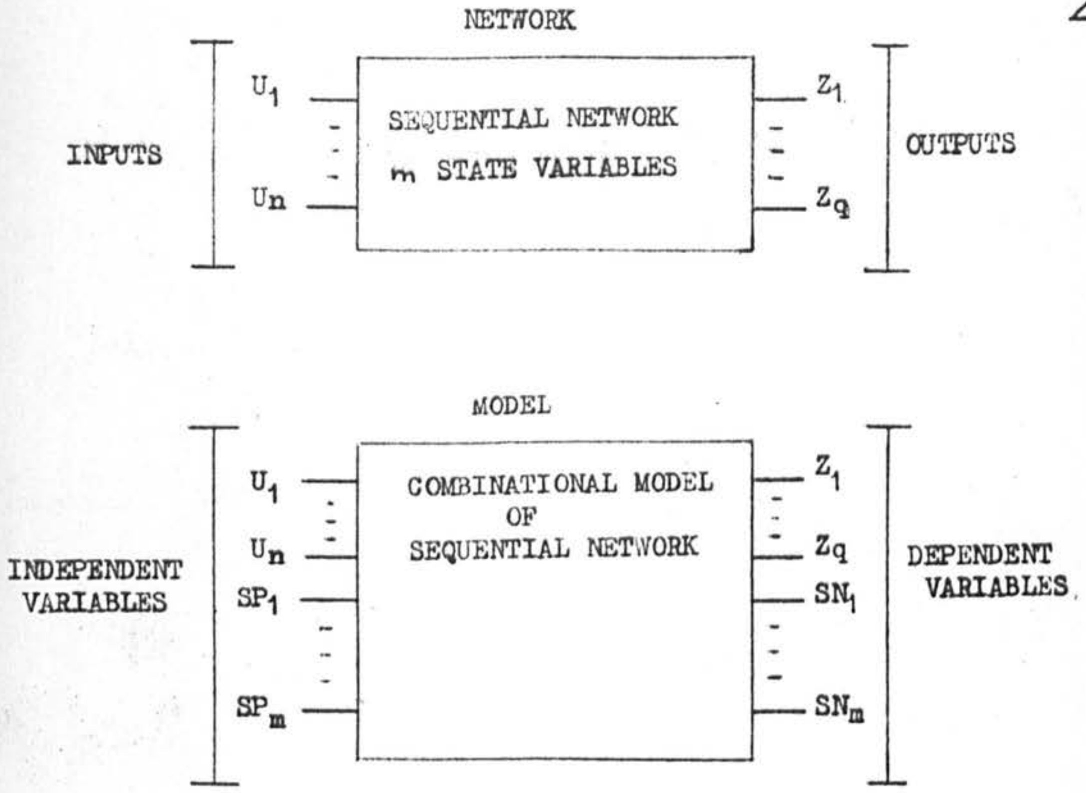


Fig.4.8.1.

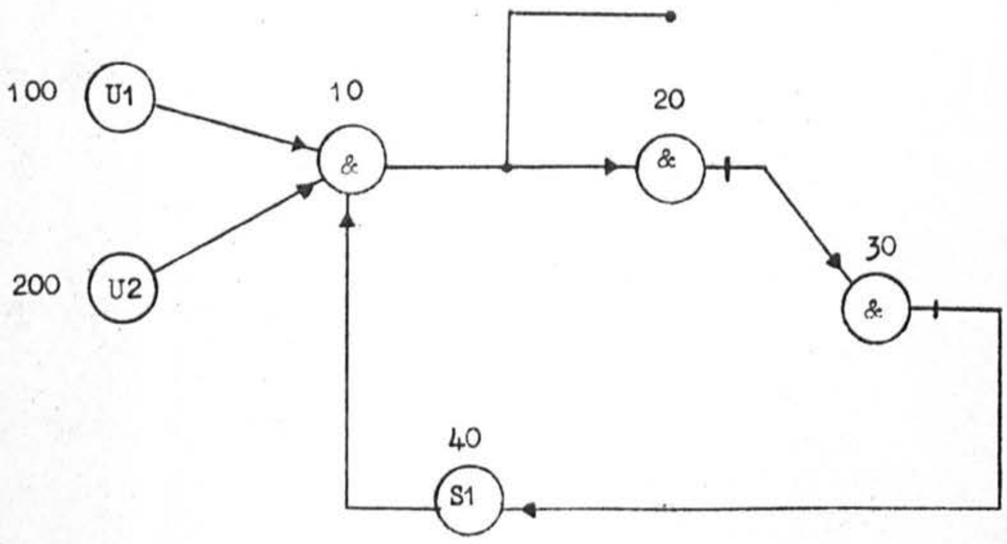


Fig.4.8.2

d40	=	0	1	0	1	0	1	0	1
d100	=	0	0	1	1	0	0	1	1
d200	=	0	0	0	0	1	1	1	1
d10	=	0	0	0	0	0	0	0	1
d20	=	1	1	1	1	1	1	1	0
d30	=	1	1	1	1	0	0	0	1

Table 4.8.1.

Ref.	U1	U2	SP1	SN1	Z1
0	0	0	0	1	0
1	0	0	1	1	0
2	1	0	0	1	0
3	1	0	1	1	0
4	0	1	0	0	0
5	0	1	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

Table 4.8.2.

MODULE H

Specification of new design	Specification of operating signals	Specification of behaviour	Output
structure + function parameters+ Processing List	Optional	standard network or equations or DIRECTORY	Yes/list of faulty lines of DIRECTORY (see Comment) Option: total DIRECTORY

Table 4.8.3.

MODULE J

Specification of new design	Specification of operating signals	Specification of behaviour	Output
DIRECTORY	-	-	List of input and state variable para-meters of stable total states

Table 4.9.1.

	Inputs		Pres. state var's		Next state var's		
Ref.	U1	U2	S1	S2	S1	S2	
0	0	0	0	0	0	0	STABLE
1	0	0	1	0	0	0	Δ m = 1
2	0	0	0	1	0	0	Δ m = 1
3	0	0	1	1	0	0	Δ m = 2
4	1	0	0	0	0	1	Δ m = 1
5	1	0	1	0	0	0	Δ m = 1
6	1	0	0	1	0	1	STABLE
7	1	0	1	1	0	1	Δ m = 1
8	0	1	0	0	0	1	Δ m = 1
9	0	1	1	0	1	0	STABLE
10	0	1	0	1	0	1	STABLE
11	0	1	1	1	1	0	Δ m = 1
12	1	1	0	0	1	1	Δ m = 2
13	1	1	1	0	1	0	STABLE
14	1	1	0	1	1	1	Δ m = 1
15	1	1	1	1	1	1	STABLE

Table 4.10.1.

MODULE K

Specification of new design	Specification of operating signals	Specification of behaviour	Output
DIRECTORY + Processing List	-	-	No/list of hazardous gates; Option: definition of independent variables leading to error

Table 4.10.2.

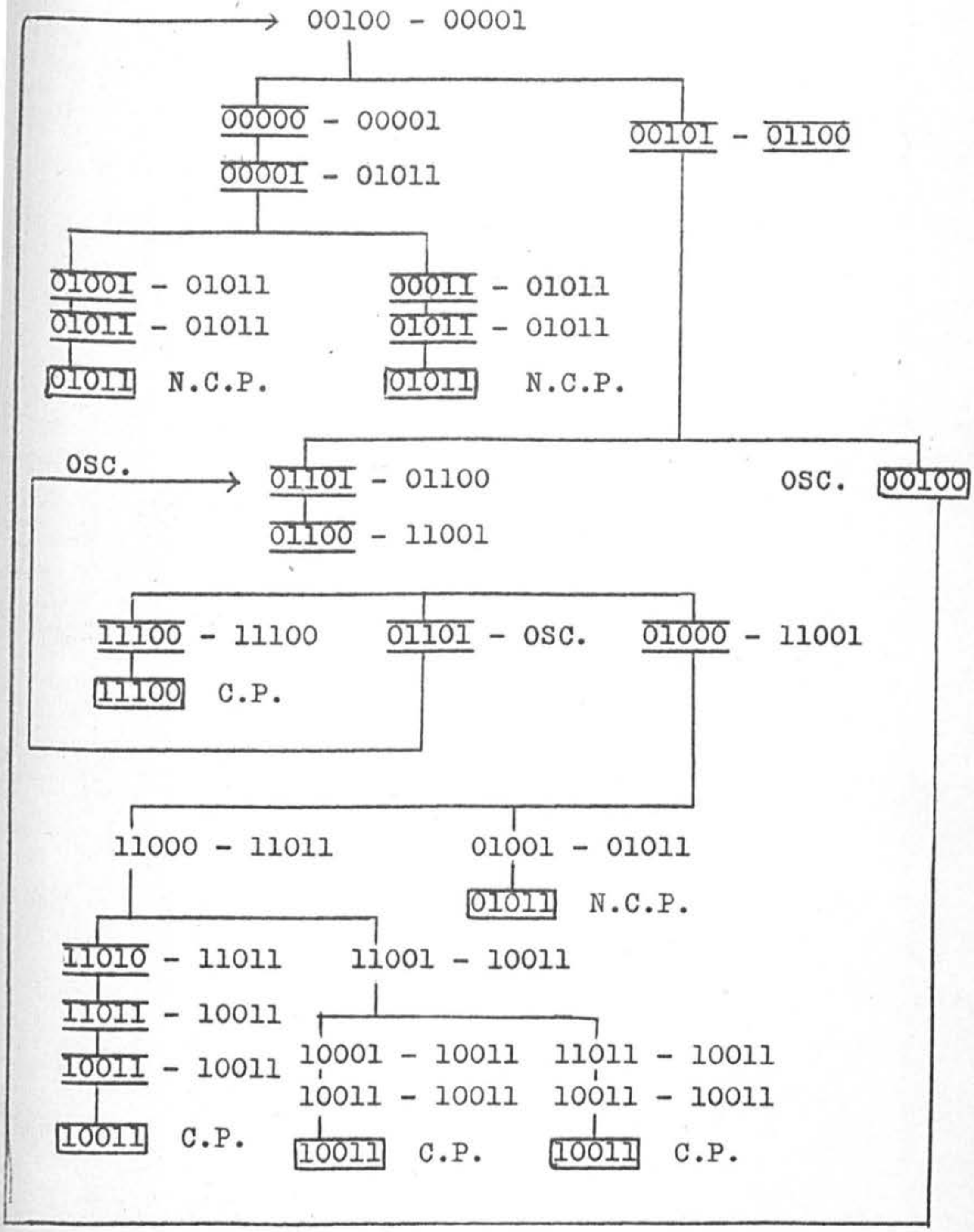
Ref.	Inputs U1	U2	Present state variables S1	S2	S3	S4	S5	Next state variables S1	S2	S3	S4	S5
32	1	0	0	0	0	0	0	0	0	0	0	1
35	1	0	1	1	0	0	0	1	1	0	1	1
36	1	0	0	0	1	0	0	0	0	0	0	1
38	1	0	0	1	1	0	0	1	1	0	0	1
39	1	0	1	1	1	0	0	1	1	1	0	0
48	1	0	0	0	0	0	1	0	1	0	1	1
49	1	0	1	0	0	0	1	1	0	0	1	1
50	1	0	0	1	0	0	1	0	1	0	0	1
51	1	0	1	1	0	0	1	1	0	0	1	1
52	1	0	0	0	1	0	1	0	1	1	0	0
56	1	0	0	0	0	1	1	0	1	0	1	1
57	1	0	1	0	0	1	1	1	0	0	1	1
58	1	0	0	1	0	1	1	0	1	0	1	1
59	1	0	1	1	0	1	1	1	0	0	1	1
100	1	1	0	0	1	0	0	0	0	1	0	0

Table 4.11.1

MODULE L

Specification of new design	Specification of operating signals	Specification of behaviour	Output
DIRECTORY	-	-	No/list of errors showing stable state, i/p transitions and adverse relative delays of racing state variables

Table 4.11.2.



00100 - 00001
00000 - 00001
00001 - 01011
00101 - 01100
01001 - 01011
01011 - 01011
01011 N.C.P.
00011 - 01011
01011 - 01011
01011 N.C.P.
OSC.
01101 - 01100
01100 - 11001
OSC.
00100
11100 - 11100
11100 C.P.
01101 - OSC.
01000 - 11001
11000 - 11011
01001 - 01011
01011 N.C.P.
11010 - 11011
11011 - 10011
10011 - 10011
10011 C.P.
11001 - 10011
10001 - 10011
10011 - 10011
10011 C.P.
11011 - 10011
10011 - 10011
10011 C.P.

Fig. 4.11.1.

MODULE M

Specification of new design	Specification of operating signals	Specification of behaviour	Output
DIRECTORY	-	-	No/list of errors showing stable state and input transitions

Table 4.11.3.

	100	200	300	400	500	4000	6000	5000	4	6	5
Ref.	U1	U2	U3	U4	U5	SP1	SP2	SP3	SN1	SN2	SN3
221	1	1	0	1	1	1	0	1	1	0	1
253	1	1	1	1	1	1	0	1	0	0	1
252	1	1	1	1	1	0	0	1	0	1	1
254	1	1	1	1	1	0	1	1	0	1	1
222	1	1	0	1	1	0	1	1	1	1	1
223	1	1	0	1	1	1	1	1	1	1	1
255	1	1	1	1	1	1	1	1	1	1	0
251	1	1	1	1	1	1	1	0	1	0	0
249	1	1	1	1	1	1	0	0	1	0	0
220	1	1	0	1	1	0	0	1	1	1	1

Table 4.11.4.

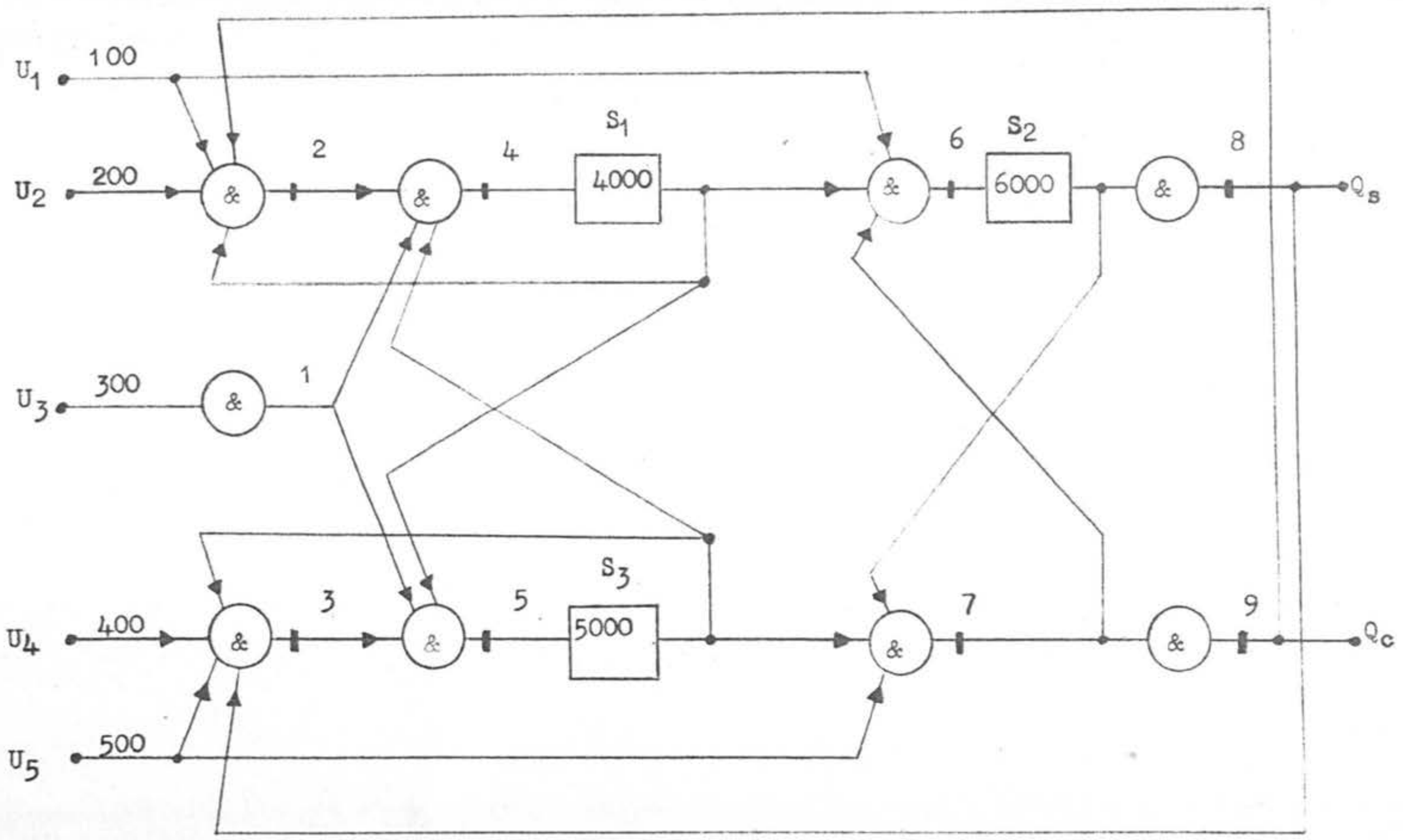

U1
100
U2
200
U3
300
U4
400
U5
500
1
2
3
4
5
6
7
8
9
S1
4000
S2
6000
S3
5000
&
Qs
Qc

Fig. 4·11·2

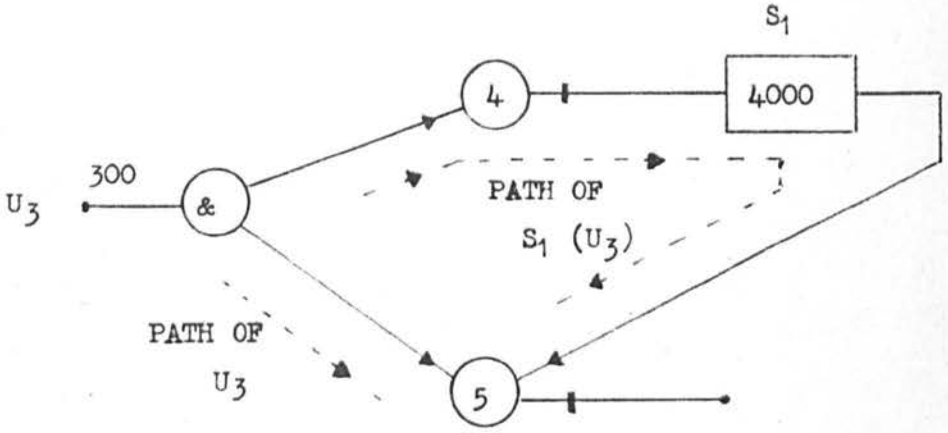


Fig.4.11.3.

INITIAL CONDITIONS	A	B	Ø	D	E	Ø	I	Ø	K	L	Ø	N	O	Ø	Q	Ø	S
.00000000	A	B	Ⓒ	D	E		I		K	L		N	O		Q		S
5.1206356	A	B	C	D	E	Ⓗ	I		K	L		N	O		Q		S
14.874813	A	B	C	D	E	H	I		◯	L		N	O		◯		S
24.909096	A	B	C	D	E	H	I			L	Ⓜ	N	O			Ⓡ	S
29.905769	A	B	C	D	E	H	I			L	M	N	◯			R	S
34.728580	A	B	C	D	E	H	I			L	M	◯				R	S
39.612494	A	B	C	D	E	H	I	Ⓙ		L	M					R	S
39.872056	A	B	C	D	E	H	I	J		L	M			Ⓟ		R	S

Table 4.12.1.

OBJECT																	
Numerical identifier (fig.4.11.2)	100	200	300	400	500	1	2	3	4	5	6	7	8	9	4000	6000	5000
Alphabetic identifier	A	B	C	D	E	H	I	J	K	L	M	N	O	P	Q	R	S
Time delay in units of 1 nsec.	|	|	|	|	|	5·1206356	9·9432654	9·7067250	9·7541773	10·098353	10·034283	9·8194893	4·9966733	5·1434768	—	—	—

Table 4.12.2.

MODULE N

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Output of modules C, J; delay parameters	stable state; input change	–	Printout of waveform on nonlinear time scale

Table 4.12.3.

MODULE P

Specification of new design	Specification of operating signals	Specification of behaviour	Output
PRIMITIVE DIRECTORY output of module H, C; delay parameter of all objects	Output of module L, M	–	VERIFIED DIRECTORY

Table 4.13.1.

Ref.	U1	U2	SP1	SP2	SN1	SN2
0	0	0	0	0	1	1
3	0	0	1	1	1	1

Table 4.14.1.

Ref.	U1	U2	SP1	SP2	SN1	SN2
0	0	0	0	0	0	1
2	0	0	0	1	1	1
3	0	0	1	1	1	1

Table 4.14.2.

					(state vector of stable state)	
Ref.	U1	U2	SP1	SP2	S1	S2
0	0	0	0	0	0	0
1	0	0	1	0	1	0
2	0	0	0	1	0	1
3	0	0	1	1	0	0
4	1	0	0	0	1	0
5	1	0	1	0	1	0
6	1	0	0	1	0	1
7	1	0	1	1	0	1
8	0	1	0	0	0	0
9	0	1	1	0	1	1
10	0	1	0	1	0	0
11	0	1	1	1	1	1
12	1	1	0	0	OSCILLATORY	
13	1	1	1	0	0	1
14	1	1	0	1	0	1
15	1	1	1	1	OSCILLATORY	

Table 4.14.3.

MODULE Q

Specification of new design	Specification of operating signals	Specification of behaviour	Output
VERIFIED DIRECTORY (output of module P)	-	-	Yes/list of conditions leading to error; Option: REDUCED DIRECTORY

Table 4.14.4.

MODULE R

Specification of new design	Specification of operating signals	Specification of behaviour	Output
REDUCED DIRECTORY (output of module Q)	-	-	Yes/list of conditions leading to oscillations

Table 4.15.1.

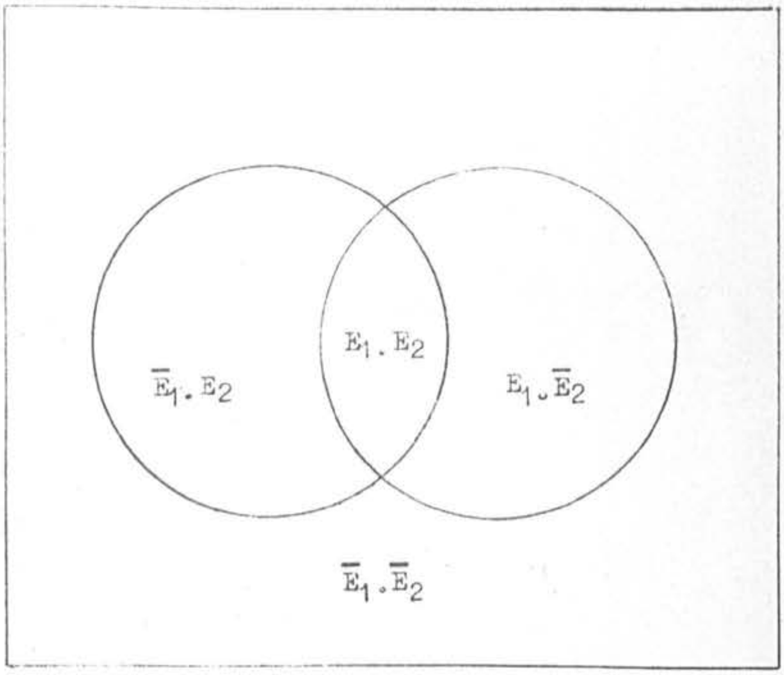


Fig.4.16.1.

MODULE S

Specification of new design	Specification of operating signals	Specification of behaviour	Output
REDUCED DIRECTORY (o/p of module Q)	starting state Y_i, parameter X	-	NO/description of Y_i and list of states not connected

Table 4.16.1.

MODULE T

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Output of modules C, J; delay parameters	output of modules L, M;	VERIFIED DIRECTORY	Failure record

Table 4.17.1.

MODULE U

Specification of new design	Specification of operating signals	Specification of behaviour	Output
Output of module D	-	-	NO/list of test conditions leading to error + description of faulty signals

Table 4.18.1.

TABLE 4.19.1

Module Name	Design Verification Question	Specification of New Design	Specification of Operating Signals	Specification of behaviour	Outputs
A	Is the state of the system fully determined by the state variables nominated?	Structure.	-	-	Yes/No. Options.
B	Which is the minimal set of state variables of the system?	Structure	-	-	Number of state variables, or list of objects with which state variables are associated. Option.
C	Can the proposed design be converted to a homogeneous structure?	Structure and function parameters.	-	-	Yes/No. Options.
D	Are any objects of the system mis-used or overloaded.	Structure and load parameters.	-	-	No/error list.
E	Is the steady-state response of the combinational network correct?	Structure and function parameters and output of module A.	Optional.	Standard network or Equations or Table.	Yes/list of errors. Option.
F	Are outputs of the combinational network subject to transient spikes?	Structure and function parameters and output of module A.	-	-	No/error list. Options.
G	Are outputs of the combinational network subject to transient spikes?	Output of module A and output of module E.	-	-	No/error list. Options.
H	Is the steady-state response of the sequential network correct?	Structure and function parameters and output of module A.	Optional	Standard network or Equations or Table.	Yes/list of errors. Options.
J	Which are the stable states of the system?	Output of module H.	-	-	List.
K	Are any of the gates of the sequential network liable to generate static hazard spikes?	Output of module A and output of module H.	-	-	No/List of errors.
L	Is the steady-state response of the system sensitive to variations of delay parameters of system objects? (secondary races)	Output of module H.	-	-	No/list of errors.
M	Is the steady-state response of the system sensitive to variations of delay parameters of system objects? (essential hazards)	Output of module H.	-	-	No/list of errors.
N	What is the simulated response of a network selected at random from a batch?	Output of module A. Output of module J, delay parameters	Stable state, input change.	-	Waveform.
P	What is the VERIFIED DIRECTORY of the sequential network?	Output of module C, output of module H delay parameters	Output of module L, output of module M.	-	VERIFIED DIRECTORY.
Q	Does the VERIFIED DIRECTORY represent the desired performance?	Output of module P.	-	Table	Yes/error list. Option.
R	Is the sequential network unconditionally stable?	Output of module Q.	-	-	Yes/list of oscillatory conditions.
S	Is the system liable to get locked in an undesirable state? or set of states?	Output of module Q.	Starting state, parameter to terminate run.	-	No/starting state and list of unconnected states.
T	What is the probability of deviation of performance from VERIFIED DIRECTORY?	Output of module A. Output of module J, delay parameters.	Output of module L, Output of Module M.	Output of module P.	Failure record.
U	Is the sequential network, specified by the VERIFIED DIRECTORY, liable to generate transient output spikes?	Output of module P.	-	-	No/list of errors.

Module name	Facilities
a	Combination of facilities of A, B & C
b	as D
c	as E
d	F or G
e	as K
f	as L and M
g	as Q and U
h	as J and R
j	as S
k	as N
l	as T

Table 5.2.1.

Test module of Proposed System	Test module of Prototype System	Comment
A	SORT	
B	-	
C	MODEL	
D	-	
E	COMBIN	
F	-	
G	STATIC	
H	SEQUEN	
J	ANALYS	
K	STATIC	Facilities are similar but not identical
L	ANALYS	
M	ANALYS	
N	TIMED	
P	-	
Q	-	
R	-	Facility exists within ANALYS; based on primitive DIRECTORY
S	-	
T	-	
U	-	Facility exists within ANALYS; based on primitive DIRECTORY
-	ANALYS	"CIRCUIT OPERATIONS" based on primitive DIRECTORY
-	SERIAL	Alternative mode for SEQUEN and ANALYS

Table 5.3.1.

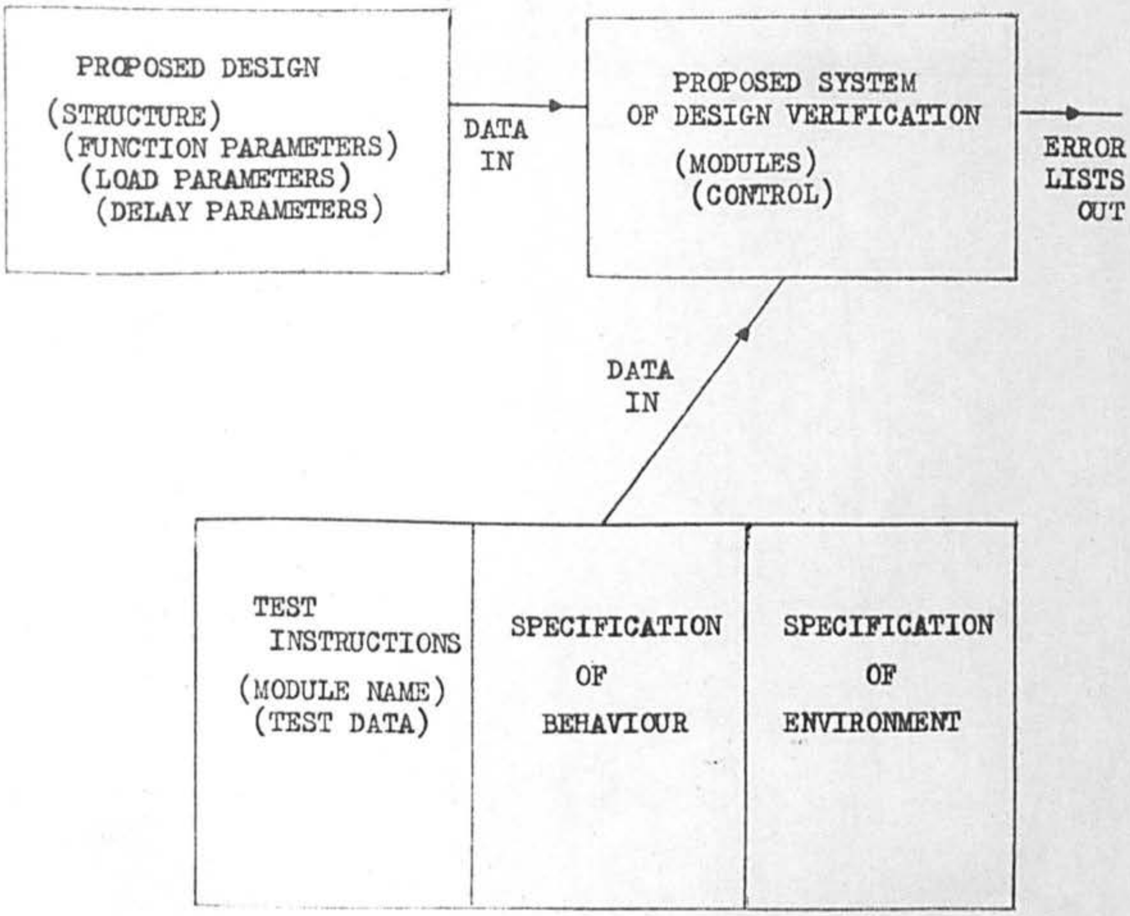

PROPOSED DESIGN
(STRUCTURE)
(FUNCTION PARAMETERS)
(LOAD PARAMETERS)
(DELAY PARAMETERS)
DATA IN
PROPOSED SYSTEM OF DESIGN VERIFICATION
(MODULES)
(CONTROL)
ERROR LISTS OUT
DATA IN
TEST INSTRUCTIONS
(MODULE NAME)
(TEST DATA)
SPECIFICATION OF BEHAVIOUR
SPECIFICATION OF ENVIRONMENT

Fig.5.1.

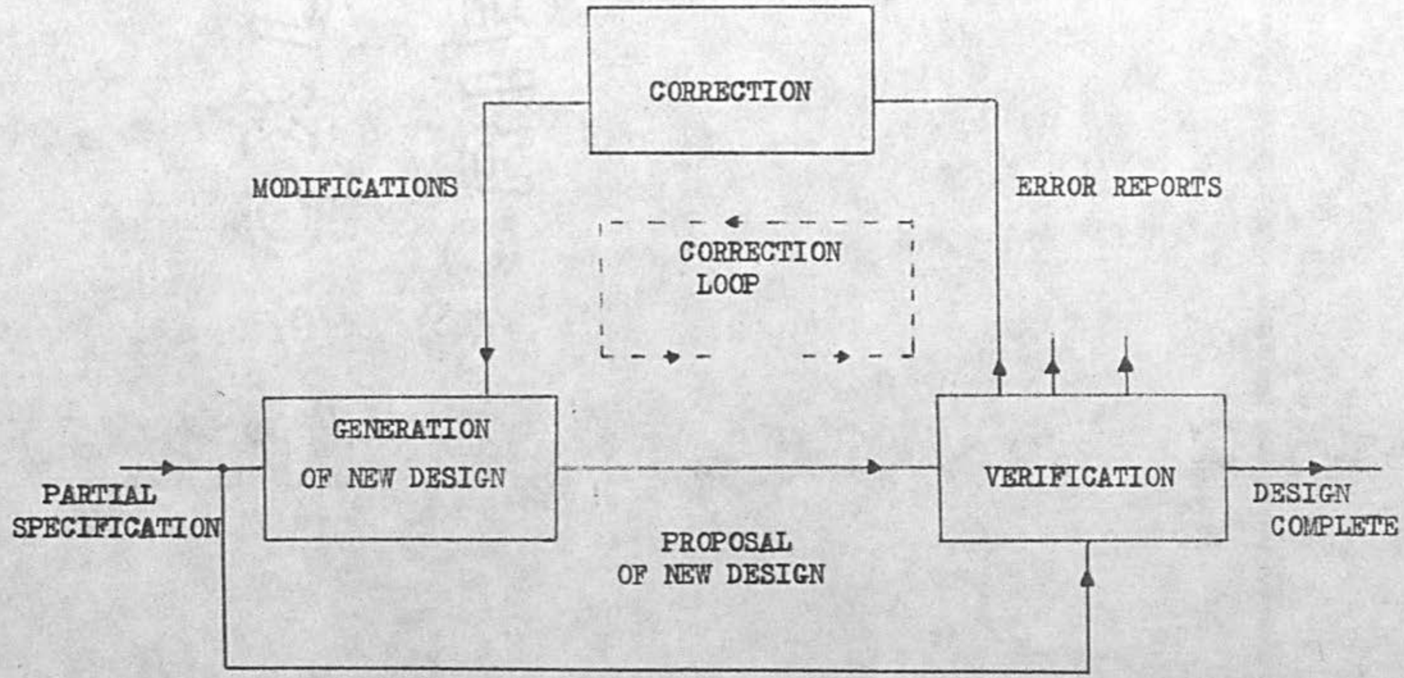

CORRECTION
MODIFICATIONS
ERROR REPORTS
CORRECTION LOOP
GENERATION OF NEW DESIGN
VERIFICATION
PARTIAL SPECIFICATION
DESIGN COMPLETE
PROPOSAL OF NEW DESIGN

Fig. 6.1.1.