Fault Detection of Binary Sequential Machines
Using R-Valued Test Machines

DONALD A. SHEPPARD, MEMBER, IEEE, AND ZVONKO G. VRANESIC, MEMBER, IEEE

Abstract—An improved method for detection of faults in completely specified synchronous sequential machines is described. The technique is algorithmic, based on the concept of embedding the given binary machine into an easily testable $R$-valued machine. Heuristic optimization of additional permutation inputs is shown to lead to considerable reduction in the length of the fault sequence. A bound on the sequence length is derived, which in most cases is significantly lower than those of comparable methods.

Index Terms—Fault detection, many-valued logic, permutation inputs, synchronous sequential machines, test machines.

I. INTRODUCTION

The design of synchronous sequential machines whose correct operation may easily be verified has been a subject of recent research interest. The question to be answered is: what minimum stimulus will generate sufficient response to ensure that the machine is correctly implementing the desired state table? The stimulus normally consists of the application of a sequence of symbols from the input alphabet, and the resulting response is compared to a known correct output. Such tests are called checking (or fault detection) experiments. The usual criterion of efficiency is the length of the test, i.e., the number of input symbols applied.

While a large number of papers on this topic can be found in the literature, most results have been improvements upon Hennie’s transition checking approach [1]. His procedure attempts to force each state table transition, and then validates this action with a previously tested distinguishing sequence. This approach is heavily dependent upon the existence of a distinguishing sequence if the length of the test is to be within some practical limits. Gonenc [2] formalized this method by introducing graph theoretic techniques, resulting in an algorithmic test generation scheme.

In order to deal effectively with machines which do not have a distinguishing sequence, it is necessary to consider the possibility of augmenting the original machine with some additional logic circuitry. Kohavi and Lavallele [3] proposed the addition of output logic. Murakami et al. [4] modified the input logic by adding a permutation input that can provide a homogeneous distinguishing sequence. Kane and Yau [5] improved upon this with a more suitable output assignment under the permutation input. They also considered using additional inputs to provide transitions needed to transform the cell graph into an Eulerian circuit.

In this paper we show an improved procedure, based on the additional input logic [4], [5], which makes use of graph theoretic techniques [2].

As in previous papers, it will be assumed that only terminal measurements are possible. It will also be postulated that faults do not occur during the test, that they last at least the length of the test, and that the state count is not increased as a result of a fault.

II. TERMINOLOGY AND GENERAL APPROACH

The procedure that follows embeds the given binary machine ($M$) into an easily testable $R$-valued machine ($M'$), where $R \geq 2$. The test sequence is developed for the latter and the resultant machine may then be implemented either in a many-valued or some encoded form.

Let the machines be described as in Table I where $\{ \}$ indicates an alphabet in base $K$. The machine $M$ is assumed to be finite-state, deterministic, synchronous, and described in Mealy form. It is noted that, practically speaking, these restrictions do not limit the usefulness of the technique. The bases of the input and output alphabets of the test machine will not be restricted to a particular
value for test generation purposes, i.e., \( R_1 \geq 2 \) and \( R_0 \geq 2 \).

A sequence is denoted by \( S = (s_0, s_1, \ldots, s_h) \), where the symbols are those of Table I. The sequence of checking symbols will be referred to as the fault sequence (FS), and it will be of the preset type (i.e., completely determined prior to the application of the test). Other sequences, such as the distinguishing sequence (DS), the homogeneous DS (HDS), the homing sequence (HS), and the synchronizing sequence (SS) are defined in the standard manner [6]. The length of any sequence \( S \) is indicated by \( L(S) \), and the aim will be to minimize \( L(FS) \) at the least possible cost.

Let a cell be defined, in the manner of Gonenc [2], as the input sequence \((x_d, x_s)\) applied to the test machine in any state \( q_i \), where \( x_d \) represents the DS used for transition validation purposes. The state table and graph associated with such cells are of interest and will be referred to as the cell table and graph, respectively.

Test procedures using the transition checking approach are usually broken into three subtests. The first, an initializing sequence (IS), maneuvers the test machine into the desired starting state. Next, a permutation sequence (PS), consisting of repeated applications of a DS, is applied in order to count the states and check the outputs associated with DS cells. Finally, a set of transition checking segments (i.e., cells) is added to the FS. These cells must be linked into a continuous sequence, and it is found that additional transitions between cells are usually required to link disjoint groups of cells (i.e., paths on the cell graph). In order to formalize these considerations (following the model of [2]) it is useful to define the following.

**Definition 1:** A path link (PL) is a transfer sequence between two disjoint paths (each path containing one or more cells).

**Definition 2:** The total path link (TPL) represents the set of PL’s required to link all paths with a particular machine into an Eulerian graph.

**Definition 3:** An easily testable sequential machine is one which possesses a minimal length HDS for the given alphabet, and for which \( L(TPL) \) is minimal.

The complete sequence of cells plus PL’s will be referred to as the cell sequence (CS). The three subtests, when concatenated, form the FS, so that

\[
FS = IS \parallel PS \parallel CS.
\]

III. MACHINE AUGMENTATION

Three possibilities are evident for the reduction of \( L(FS) \). 1) Shorten \( L(X_d) \) since it is applied once for each cell in the CS. 2) Make \( X_d \) an HDS in order to allow overlapping in the PS, and ensure its cyclic character in order to eliminate transfer sequences in the PS. 3) Minimize \( L(TPL) \) in order to shorten the CS. Moreover, the FS should be designed to eliminate transfer sequences (other than PL’s) whenever possible.

A. The Permutation Input

The second possibility stated above was implemented by Murakami et al. [4] through introduction of the permutation input \( x_r \), such that:

\[
\delta'(x_r, q_i) = q_{i+1}, \quad i = 1, 2, \ldots, n - 1
\]

\[
\delta'(x_r, q_n) = q_1.
\]

Furthermore, their output assignment under \( x_r \) was

\[
\lambda'(x_r, q_i) = 0, \quad i = 1, 2, \ldots, n - 1
\]

\[
\lambda'(x_r, q_n) = 1.
\]

Thus, a HDS consisting of \( n \) applications of \( x_r \) is created.

Kane and Yau [5] pointed out that a shorter HDS can be obtained if the output assignment under \( x_r \) (when \( p < n \)) corresponds to a \( p \)-ary shift register sequence of length \( n \) and minimum degree, which may be found using Smith’s algorithm [7].

We note that the above choice of state transitions under \( x_r \) is not essential for it is only necessary to provide a cyclic format. It will be shown later how this flexibility can be used to advantage.

The length of the DS may be reduced by expanding the output alphabet to equal the number of states, whenever necessary. Thus \( p' \geq n \), and a unique output symbol can be assigned to each transition of the permutation input. This reduces \( X_d \) to a single symbol \( (X_d = x_r) \) and yields a cell length of 2. This is obviously minimal, because at least one forcing and one checking symbol will always be required. Since the design of the FS is independent of the particular output assignment used, circuit cost may be reduced by properly choosing the outputs. Such assignments may also be used to alter the form of the output sequence. It should be emphasized, however, that expansion of the output alphabet may be objectionable from a practical point of view. This may particularly be so if outputs are binary coded, although the increase in component cost is often small.

In view of the current developments in many-valued logic realizations, it is reasonable to expect that many-valued circuits can be used effectively for simple functions such as the ones arising from simple output assignments. This is particularly promising due to the fact that the output function under \( x_r \) is used for simple matching purposes only (during the test); hence most level tolerance difficulties normally involved in construction of complex multistage circuits (requiring considerable driving capability) are not a significant problem. Furthermore, it should be pointed out that such functions are well suited for realization with multithreshold elements which have been reported recently [8], [9].

B. The Weight Table

It is known from graph theory [2], [5] that the nodes of a graph can be partitioned into the following three sets:

\[
E = \{q \in Q \mid TI_i = TO_i\}.
\]
Similarly, from the given machine, the number of transitions entering state \( q_i \) is 
\[ T_{I_i} = \text{the number of transitions entering state } q_i, \]
and the number of transitions leaving state \( q_i \) is 
\[ T_{O_i} = \text{the number of transitions leaving state } q_i, \]
and 
\[ E + N + P = Q. \]

A graph of transition testing cells may be defined for any given machine. A path in such a graph is a series of concatenated cells; a covering is a set of disjoint paths in which all cells are represented once and only once; and a circuit is a covering consisting of a single path beginning and ending at the same node.

Our aim will be to generate an Eulerian cell graph, for which a circuit may be found from any starting state. Modifications will be made by adding PL’s until this condition is obtained. It will, in general, be necessary to add \( K \) path links, where 
\[ K = \sum_p(T_{I_i} - T_{O_i}) = \sum_N(T_{O_i} - T_{I_i}), \]
in order to accomplish the desired alteration. It is in the generation of short PL sequences that changes in the permutation input have a marked effect.

To illustrate more clearly the possibilities involved we shall make use of a weight table, an example of which is given in Fig. 2 for machine \( M_1 \) (Fig. 1). The weight table is an enumeration of the transitions into and out of each state. It is readily seen that 
\[ W = \sum_{i=1}^{n} |w_i| \]
and 
\[ w_i = T_{I_i} - T_{O_i}, \]
where \( w_i \) is the weight associated with the \( i \)th state. It should be noted that each PL will cause a transition from a positive weight to a negative weight.

Analogously, a cell weight table may be constructed for the cell graph. The weights due to transitions resulting from input sequences \( \{x_iX_k\} \), as opposed to single symbol \( x_i \) inputs, may also be determined from the state weight table. The transition due to \( \{X_k\} \) for each state is represented as a directed arc from the initial to the final state. The difference, \( T_{I_k} - T_{O_k} \), forms the weight of the state pointed at by the arc. Fig. 3 illustrates this for machine \( M_1 \) and the permutation input \( \{x_i\} \) given.

**Theorem 1:** For a completely specified sequential machine, the weights of the cell graph are a permutation of the weights on the corresponding state diagram.

**Proof:** Consider any three states \( q_i, q_j, \) and \( q_k \) such that the transitions on the cell graph are
\[ x_iX_k \rightarrow q_i \rightarrow q_j \rightarrow q_k. \]
From the state diagram we obtain the state weights
\[ w_i = T_{I_i} - T_{O_i}, \]
\[ w_j = T_{I_j} - T_{O_j}, \]
\[ w_k = T_{I_k} - T_{O_k}. \]

Similarly, for the cell weights,
\[ w_i' = T_{I_i} - T_{O_i}, \]
\[ w_j' = T_{I_j} - T_{O_j}, \]
\[ w_k' = T_{I_k} - T_{O_k}. \]

Since the machine is completely specified it follows that 
\[ T_{O_i} = T_{O_j} = T_{O_k} = m, \]
and therefore 
\[ w_i' = w_i \quad \text{and} \quad w_j' = w_j. \]
Thus the weights associated with states \( q_i \) and \( q_j \) on the cell table are equal to those of \( q_i \) and \( q_j \), respectively, on the state table and are simply shifted. Similar reasoning may be applied to all other states, thus proving the theorem. The following corollary follows immediately.

**Corollary 1:** If a cell graph of a completely specified sequential machine is not Eulerian for any given permutation input, then it is not Eulerian for all possible permutation inputs.

Therefore, in case of completely specified machines, the only advantages to be gained from permutation input design lie in the reduction of the number of PL symbols used to modify the cell graph.

**C. Permutation Input Optimization**

Variations in permutation input format can be beneficial in \( L(FS) \) reduction. Clearly, it is possible to construct a tree of all possible state cycles and use it to determine the optimum permutation input. However, such exhaustive enumeration is practical only for machines with a relatively small number of states. A more useful alternative is to devise a heuristic procedure which reduces the computational effort and leads to nearly optimal solutions. One such procedure is described in the remainder of this section.

The optimization problem can be reduced to a process of reordering the state weights. Since our goal is to minimize \( L(TPL) \), we shall attempt to find a permutation input \( x_i \),
which minimizes distances between positive and negative weights. The distance is the number of applications of $x_i$ necessary to cause the transition from a state corresponding to a positive weight to a state corresponding to a negative weight.

The effect of the permutation input can be represented with directed arcs on the weight vector. For example we can represent the effect of $x_i$ from Fig. 3 as shown in Fig. 4(a). In order to facilitate calculation of distances it is convenient to represent $x_i$ in cyclic form as indicated in Fig. 4(b), which is done by reordering the weights corresponding to the given states. Note that in our example $A \rightarrow B \rightarrow C \rightarrow D \rightarrow F \rightarrow E \rightarrow A$, as indicated by the cyclic weight vector $w_{IR}$.

From $w_{IR}$ it is readily seen that there are four path links that are needed as indicated with dotted arcs in Fig. 4(c). It also follows directly that in our example $L(TPL) = 7$.

Thus, $w_{IR}$ provides a simple means of determining $L(TPL)$. However, the actual path links cannot be found directly from $w_{IR}$ and must be obtained from the cell weight vector $w_i'$. The following heuristic procedure has been found effective in the formation of $w_{IR}$ leading to nearly optimal permutation inputs.

Procedure 1

Step 1: Enter all zero weights at the top of the $w_{IR}$ vector.

Step 2: Match pairs of equal positive and negative weights, adding each such pair to $w_{IR}$ (with positive weights uppermost). Process the $w_i$ list sequentially from the top, accepting the first suitable weights.

Step 3: Add the remaining weights in positive and negative groups such that the distance is minimized. Process these weights in descending order, and enter positive before negative groups.

The above steps can be justified by the following considerations. Each $PL$ is required to generate a transition from a positive weight to a negative weight. Since zero weights cannot be initial or terminal nodes of a $PL$ they should not be embedded into one, and this is ensured by their removal in Step 1. Similarly matched pairs are removed in Step 2 as they form self-contained paths and should be excluded from other $PL$'s. In Step 3 the largest positive and negative weights are arranged in closest proximity in order to minimize the length of the greatest number of $PL$'s.

Application of the above procedure to the state weights of Fig. 1 (for machine $M_1$) yields the result given in Fig. 5. This is seen to lead to the permutation input of Fig. 3. It should be noted that $w_d$ does not necessarily give a unique $x_i$. For example, the $w_{IR}$ in Fig. 4 may also give rise to the permutation input $A \rightarrow B \rightarrow F \rightarrow D \rightarrow C \rightarrow E \rightarrow A$.

D. Alternate Sources for Path Links

In Section III-C the permutation input was optimized in order to reduce $L(TPL)$. Since transfer sequences generated by any $DS$ can, at worst case, be $n$-1 symbols in length, it is worthwhile investigating alternate sources. Any one $PL$ can be eliminated by the use of a pseudo-Eulerian graph (i.e., a graph where $K = 1$ and the starting state is fixed). This, however, restricts the $CS$ starting state which will often necessitate additional symbols in the $IS$. Secondly, it is often possible to find suitable transfer sequences in the given machine that could be used effectively, although there is no guarantee that these will exist. Furthermore, such sequences must be tested prior to being used, which reduces the flexibility in the generation of CS. A third possibility, suggested by Kane and Yau [5], is to introduce PL cells under new inputs. Such sequences are self-checking (since they possess the form $(x_iX_d)$ and have a maximum length of $L(DS) + 1$).

IV. TEST MACHINE AND FAULT SEQUENCE GENERATION

In the previous section the possibility of optimizing the permutation input was discussed. It was also suggested that the output alphabet may be expanded for testing purposes, in order to reduce the length of $DS$. This clearly involves considerable tradeoff between the length of the test and the cost of added hardware. We feel that the importance of such tradeoffs cannot be overemphasized, hence rigid rules should be avoided.
The following procedure for generation of easily testable machines and the resultant test sequence is an improvement of Kane and Yau's method [5]. In order to present it as clearly as possible, we will consider the reduction of \( L(\text{FS}) \) as the main objective, and illustrate the cost tradeoff by means of an example. Thus, any PL of length greater than \( L(\text{DS}) + 1 \) is implemented through additional inputs. It should be noted that if an additional input is used to realize a PL, it may also be used to provide a reset mechanism. For example, let the input \( x_{r+1} \) in Fig. 6(a) be used in realization of a single PL cell. Then, it will likely be advantageous to make use of this input for initial synchronization to state B, as indicated in Fig. 6(b).

**Procedure 2**

**Step 1:** Determine the optimal (or nearly optimal) permutation input for the given machine using Procedure 1 (for completely specified \( M \)) or exhaustive examination (for incompletely specified \( M \)). Add the transitions thus derived to \( M \) under a new input symbol \( x_r \) to form a machine \( M' \).

**Step 2:** Assign output symbols to each transition under \( x_r \). If the allowable output alphabet \( p' \geq n \), then each transition yields a distinct output. For this case \( X_d = (x_r) \) and \( L(\text{DS}) = 1 \). Otherwise Smith's algorithm may be used. Then \( X_d = (x_r x_{r+} \cdots) \), where \( x_r \) is repeated \( L(\text{DS}) = \lceil \log p' n \rceil \) times. (It should be noted that Smith's algorithm is subject to lengths restrictions [7], however this is not a hindrance for practical machines.)

**Step 3:** Form a cell table (and cell graph \( G \)) for \( M' \), deleting permutation input cells, and find the cell weight vector. If \( W = 0 \) go to Step 6.

**Step 4:** Determine the path links required from the weight table. Generate these using repeated applications of \( x_r \). Replace any PL's that are of length greater than \( L(\text{DS}) + 1 \) with cell PL's added under new input symbols (\( x_{r+1}, x_{r+2} \cdots \)). Thus the augmented machine \( M' \) is produced, and the TPL set determined.

**Step 5:** Add the set of PL's to the cell graph \( G \), thus forming a modified graph \( G' \) possessing an Eulerian circuit.

**Step 6:** Generate an Eulerian circuit of \( G' \) beginning at the required starting state (\( x_d \) successor of the synchronization) to obtain the CS.

**Step 7:** Append the PS subtest consisting of \( n + L(\text{DS}) x_r \) symbols to the beginning of the CS, arranging the final state to coincide with the first state of the CS.

**Step 8:** Complete the FS by adding an IS to the beginning of the PS of Step 7.

V. EXAMPLE

Consider the machine \( M_1 \) given in Fig. 1. Let the output alphabet be \( 0' = \{0,1,2\} \).

**Step 1:** The permutation input is derived from the state weights using Procedure 1 as shown in Fig. 5. The resultant \( w_{ir} \) implies \( x_r = 2: A \rightarrow B \rightarrow C \rightarrow D \rightarrow F \rightarrow E \rightarrow A \).

**Step 2:** Since \( p' = 3 < n \), use Smith's algorithm to find the output assignment under \( x_r \) as shown in Fig. 7.

**Step 3:** A cell table is formed, deleting all \( x_r \) cells. This is shown in Fig. 8, along with the corresponding cell weight vector. (Note that the cell weight pattern merely follows the \( X_d \) — successor pattern of state weights.) It is seen that \( W = 8 \neq 0 \).

**Step 4:** The following PL's are required:

\[
\begin{align*}
2 & \quad B \rightarrow C \\
2 & \quad D \rightarrow F \\
22 & \quad D \rightarrow E \\
222 & \quad D \rightarrow A.
\end{align*}
\]

All PL's should be implemented under \( x_r \), and \( M'_1 \) is in fact the final augmented machine \( M'_1 \).

**Step 5:** The above set of PL's is added to the cell graph to form \( G' \) as in Fig. 9.

**Step 6:** A circuit of \( G' \) may now be generated with the
beginning. Therefore, the following state possessing \( \text{Step 7:} \) The \( n + L(DS) \) \( x_r \)'s are next appended to the beginning of the \( CS \), so that

\[
X: \quad 0 \quad 2 \quad 2 \quad 1 \quad 2 \quad 2 \quad 0 \quad 2 \quad 2 \quad 2 \quad 2
\]

\[
CS = Q; [D] B \quad C \quad D \quad A \quad B \quad C \quad B \quad C \quad D \quad F \quad C \quad D \quad F \quad B \quad C \quad D \quad F \quad E
\]

\[
Z: \quad 0 \quad 2 \quad 2 \quad 1 \quad 0 \quad 2 \quad 0 \quad 2 \quad 2 \quad 1 \quad 2 \quad 1 \quad 0 \quad 2 \quad 2 \quad 1 \quad 2
\]

\[
0 \quad 2 \quad 2 \quad 1 \quad 2 \quad 2 \quad 0 \quad 2 \quad 2 \quad 1 \quad 2 \quad 2 \quad 2 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 0 \quad 2 \quad 2
\]

\[
E \quad A \quad B \quad E \quad A \quad B \quad F \quad E \quad A \quad E \quad A \quad B \quad C \quad D \quad F \quad E \quad B \quad C \quad D \quad F \quad E \quad A \quad B \quad C \quad D
\]

\[
1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 2 \quad 0 \quad 1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 2 \quad 0 \quad 0 \quad 2 \quad 2.
\]

The starting state for the \( CS \) is the \( X_r \)-successor of the synchronization.

\[
PS = [B] C \quad D \quad F \quad E \quad A \quad B \quad C \quad D
\]

\[
2 \quad 2 \quad 1 \quad 2 \quad 0 \quad 0 \quad 2 \quad 2.
\]

Step 8: Finally, the \( IS \) should be appended to the beginning. For \( M_1 \) it is possible to have a preset \( IS \) since it possesses a synchronizing sequence, thus:

\[
1 \quad 1 \quad 1 \quad 0 \quad 1
\]

\[
IS = [q_r] \quad [B]
\]

Therefore, the length of \( FS \) is 56 symbols.

In order to illustrate the tradeoffs involved, let us again consider the machine \( M_1 \), but without the previous restriction on the output alphabet size (hence needing six distinct output symbols). Applying the Procedure 2 we find that the \( PL \) of length 3 can be replaced by a cell under the new input \( x_{r+1} = 3 \), giving the augmented test machine shown in Fig. 10. The resultant \( FS \) consists of 42 symbols.

Let us next consider a tighter restriction on the output alphabet, so that \( 0' = \{0,1\} \). In this case \( L(Xd) = 3 \), the only additional input is \( x_r = 2 \), and \( L(FS) = 69 \). It is of interest to compare this with a test machine generated using Kane and Yau's method, where four additional inputs are needed and \( L(FS) = 78 \).

**VI. CONCLUSIONS**

The proposed technique is a significant improvement over currently available methods of similar type. The resultant fault sequences are in most cases considerably shorter than those obtainable with Kane and Yau's [5] method, without corresponding increase in circuit complexity.

A bound on the test length is readily derived:

\[
L(FS) = L(IS) + L(PS) + L(CS)
\]

\[
\leq (n) + (n + s) + [mn(s + 1)]
\]

\[
+ m(s + 1)(n - s - 2) + m \sum_{i=1}^{s+1} i
\]

\[
\leq 2n + s + m(s + 1)(2n - s - 2) + m \sum_{i=1}^{s+1} i
\]

where \( s = L(DS) \). It is also noted that at most \( m(n - 1) - 1 \) input symbols must be added.

There are two disadvantages of the proposed technique. Firstly, the properties of the given machine are not
always employed efficiently. Thus the possibility of using transfer sequences from the given machine is ignored as well as the use of multiple overlapping distinguishing sequences.

Secondly, augmentation of input and output logic may require a number of additional leads, particularly in binary coded implementation. Clearly, the limitation is most severe in cases of single-input–single-output machines with a large number of states. This provides the incentive for the use of many-valued logic, which is particularly attractive in the pin-limited environment of integrated circuits.

However, we feel that the algorithmic nature of the technique and the reduced length of tests far outweigh the above mentioned drawbacks.

Finally, it is important to emphasize the tradeoff between the test length and circuit complexity of the test machine.

REFERENCES


Donald A. Sheppard (M’72) was born in Montreal, P. Q., Canada, on March 25, 1948. He received the B.Eng.(Elec.) degree from McGill University, Montreal, P. Q., Canada, in 1969, and the M.A.Sc. degree from the University of Toronto, Toronto, Ont., Canada, in 1972.

From June 1969 to September 1970 he worked for C. P. Telecommunications. He is presently a Communication Engineer for the Canadian National Railway. He is involved in the implementation of the CNR TRACS computer system. His interests are in reliability, fault diagnosis and, many-valued switching systems.

Zvonko G. Vranesc (S’67–M’69) was born in Zagreb, Yugoslavia, on October 4, 1938. He received the B.A.Sc. degree in 1963, the M.A.Sc. degree in 1966, and the Ph.D. degree in 1968 in electrical engineering from the University of Toronto, Toronto, Ont., Canada.

From 1963 to 1965, he was with the Northern Electric Co., Ltd., Bramalea, Ont., Canada. In 1968 he joined the faculty of the Departments of Electrical Engineering and Computer Science at the University of Toronto, where he is now an Associate Professor. His research interests include many-valued switching systems, fault tolerant computing, computer architecture, and heuristic programming.

Testing for Faults in Wiring Networks

WILLIAM H. KAUTZ

Abstract—An algorithm is derived for multiprobe testing for shorts, opens, and wiring errors in any multiterminal wiring network, such as a printed circuit board, wiring harness, multiconductor cable, or backplane wiring board. For behavioral testing the minimum number of tests required, always achievable, is equal to $p - 1 + \lceil \log_2 q \rceil$, where $p$ is the number of terminals in the largest interconnected cluster in the network, and $q$ is the total number of clusters, including isolated terminals. For structural testing the number of tests required is less, and can be as small as $\lceil \log_2 q \rceil + 1$ depending upon the assumptions made regarding the types of faults that can occur.

Index Terms—Behavioral testing, faults, structural testing, switching circuits, wiring networks.

INTRODUCTION

HUNDREDS of technical papers have appeared in the U. S. in the last decade on the subject of fault testing