Use of SPOOF's in the Analysis of Faulty Logic Networks

FREDERICK W. CLEGG

Abstract—In general, one cannot predict the effects of possible failures on the functional characteristics of a logic network without knowledge of the structure of that network.

The structure- and parity-observing output function (SPOOF) described in this paper provides a new and convenient means of characterizing both network structure and output function in a single algebraic expression.

A straightforward method for the determination of a SPOOF for any logic network is demonstrated. Similarities between SPOOF's and other means of characterizing network structure are discussed. Examples are presented that show several useful applications of this new tool. It is shown that the SPOOF provides an easy means by which the effects of any "stuck-at" fault—single or multiple—on the functional characteristics of a logic network can be determined. In addition, one may, using SPOOF's, determine just which faults, if any, can occur in a network to affect its output function in a given way. Other likely applications of SPOOF's, not yet fully explored, are indicated and areas for future research are suggested.

Index Terms—Fault detection and diagnosis, fault equivalence, fault-tolerant computing, reliability of digital systems, SPOOF's.

I. INTRODUCTION

THIS paper will present a new tool that is of potential use to those who wish to study the characteristics of switching circuits in which one or more of the circuit's components are subject to failure. The techniques presented are applicable to gate-type combinational logic networks. The type of failures considered is that of "stuck-at" faults—conditions whose logical effect can be specified by saying that the levels on some signal lines in the network in which they are present are stuck-at constant logical values. The condition, say $F_i$, in which some line $i$ in a network is stuck-at a constant logical 0 and some line $k$ in the same network is simultaneously stuck-at a logical 1 will be written as $F_i = j/0, k/1$. The set of all possible stuck-at faults that can occur in a logic network is designated as $F$. In this and in all other respects, the notations used in this paper are compatible with those used [1]–[3].

In the analysis of logic networks, it is possible to obtain a certain amount of knowledge about the effects of certain faults on particular networks and their output functions given only knowledge of the Boolean functions that these networks implement. Unfortunately, however, one generally needs infor-

1 See, for example, [2, theorems 6.9(a) and (b)].
Fig. 1. An illustration of a case where knowledge of network structure is necessary to determine the possible effects faults may have on network output function. (a) In the presence of \( F = a/1, d/1, f/1 \), this structure realizes the function \( z = x \vee \bar{y} \). (b) The design function of this structure is the same as that for the structure of (a) viz. \( z = x \oplus y \) — but there are no stuck-at faults at all which can occur in this structure to cause the output function to become \( z = x \vee y \).

Information about the structure of a logic network as well in order to be able to predict the effects of various possible faults on the network’s output function. This fact is illustrated in the following example.

Example 1: Fig. 1 indicates the structure of two logic networks. The manner in which directed graphs have been employed as models or descriptions of network structure is relatively self-evident; this modeling scheme is, however, discussed fully in [1]. The \( x, y \), and \( z \) vertices correspond in each case to the input and output terminals of the network and are called input vertices and output vertices, respectively. The edges of the graphs correspond to network signal lines. In Fig. 1(b), the “\( \overline{\text{T}} \)” symbols associated with vertices \( A1 \) and \( A2 \) indicate that these vertices correspond to inverter circuits in the network described. The “\( \text{A} \)” symbol associated with vertices \( A3 \) through \( A5 \) indicate that these vertices correspond to gates realizing the complement of the \( \text{AND} \) (A) function—i.e., to \( \text{NAND} \) gates. Such a graph describing the structure of a network will be called the logical model of the network.

Both of the networks described in Fig. 1 are implementations of the \text{EXCLUSIVE-OR} function, \( z = x \oplus y \). We say that the \text{EXCLUSIVE-OR} function is the design function of both networks.

The familiar structure of Fig. 1(a) employs four two-input \text{NAND} gates while the structure of Fig. 1(b) utilizes three two-input \text{NAND} gates in addition to two inverters.

If the realization of Fig. 1(a) is afflicted by the fault \( F = a/1, d/1, f/1 \), the output function of this structure will become \( z = x \vee \bar{y} \).

On the other hand, in the entire set \( \mathcal{G} \) for the structure shown in Fig. 1(b) there is no fault at all that can cause the network output function to change to \( z = x \vee \bar{y} \). (The reader who does not believe this assertion is invited to find such a fault.)

Example 1 demonstrates the fact that, in order to know fully how the output function of a given network may be affected by various faults (or, alternatively, in order to know what fault functions are possible), one must in general have some knowledge of the structure of that network. Knowledge merely of the design function of the network is not, in general, adequate for this purpose.

II. A MEANS OF CHARACTERIZING NETWORK STRUCTURE OF A BOOLEAN EXPRESSION

A convenient and effective technique for characterizing the structure of a logic network to an extent sufficient to permit analysis of the effects of all possible stuck-at faults on the output function of that network will now be presented. This technique will employ somewhat modified Boolean expressions for the output function of a given network which themselves bear information about the details of that network’s structure.

Let us look again at the \text{EXCLUSIVE-OR} network whose structure is shown in Fig. 1(a). The logic signal at the output terminal of this network is just the logic function associated with edge \( k \) of the graph of Fig. 1(a). We denote this by

\[ z = z_k. \]

Using subscripted \( z \)'s to indicate the logic signals ordinarily present on the various lines of the network (or edges of the graph model of the network) we proceed writing

\[ z = z_k = (z_k z_h)_k \]
\[ = (z_c z_d)_{fg} (z_c z_f)_{hi} \]
\[ = z_c z_k z_d z_h \]
\[ = z_c z_k z_d z_h \]

The reader will notice that we are affixing complement designations (overbars) to certain of the subscripts as well as to the signal variables themselves. This is an essential aspect of the technique being developed here. Continuing, we have

\[ z = z_c z_k z_d z_h \]
\[ = x_c z_k (z_c z_d)_{fg} (z_c z_f)_{hi} \]
\[ = x_c z_k z_c z_d z_h \]
\[ = x_c z_k z_c z_d z_h \]

The signals on edges \( c \) and \( f \) are just the input signals \( x \) and \( y \), respectively, so we continue with

\[ z = x_c z_k z_d z_h \]
\[ = x_c z_k z_d z_h \]
\[ = x_c z_k z_d z_h \]
\[ = x_c z_k z_d z_h \]

This resulting expression of the output function of the network will be called a disjunctive \text{SPOOF}. The structure- and parity-observing output function (SPOOF) with the adjective disjunctive indicates that we have repeatedly used the distributive property \( u(v \lor w) = u v \lor u w \) to obtain an expression in disjunctive normal form (i.e., “sum-of-products” form). By using instead, the distributive property \( u \lor (v \land w) = (u \lor v) \land (u \lor w) \), one can obtain a similar expression for the network output function in “product-of-sums” form called a conjunctive \text{SPOOF}. 

IEEE TRANSACTIONS ON COMPUTERS, MARCH 1973
It is of essential importance to note that the properties of ordinary Boolean algebra \( u \bar{u} = 0 \) and \( u \lor \bar{u} = 1 \) must not be used to “cancel” terms in the expression.

The reader may note at this point that the disjunctive SPOOF of a logic network as informally introduced above bears a close resemblance to the equivalent normal form (e n f) of that network as developed by Armstrong [4] to facilitate generation of tests for the network. The essential difference between the disjunctive SPOOF and the e n f for a given network is the presence or absence of overbars over each of the subscript symbols of each literal indicating the parity of the signal (i.e., whether the signal appears in “true” or complement form) associated with that literal on the line associated with that subscript.

The SPOOF that we have introduced, it may be noted, also exhibits similarities to the output functions expressed in terms of “literal propositions” as developed by Poage [5]. To readers familiar with the work of [5], it will be apparent, however, that the SPOOF provides a much more compact and somewhat more tractable and more easily derived notation for the information needed for complete analysis of the effects of possible faults on the functional characteristics of a given network.

The reader may also note that each term of the disjunctive SPOOF looks very similar to a conjunction of the elements of a \( P \) set, as the latter are defined by McCluskey [6]. Before we formally define SPOOF’s, in fact, we shall introduce some useful extensions of the concepts of \( P \) sets and \( S \) sets.

III. SPO-P SETS, SPO-S SETS, AND SPOOF’S

Definition 1: A set of subscripted literals is an SPO-P set (structure- and parity-observing \( P \) set) of a network whose logical model is \( G \) if and only if the following conditions hold.

Condition 1: Each literal of the set possesses a subscript consisting of a sequence of symbols corresponding to a sequence of edges constituting a path through \( G \) from an input vertex to an output vertex.

Condition 2: If we denote as \( \mathcal{Z}_F[G] \) the output of the network modeled by \( G \) whenever that network contains the fault \( F \), then \( \mathcal{Z}_F[G] \) has a value of 1 whenever all of the literals of the set have values of 1, where \( F \) is any member of the set \( \mathcal{F} \) for the network that does not involve any of the lines of the network corresponding to edges in \( G \) denoted by symbols appearing in one or more of the subscripts of the literals of the set.

Condition 3: When any literal is removed from the set, Condition 2 no longer holds.

Condition 4: Whenever, along the path described by the subscript of a literal in the set, there are, between an edge \( a \) in that path and the output vertex of \( G \), an odd number of inverting vertices,\(^2\) the symbol \( a \) is overbarred.

Condition 5: No subscript symbol not overbarred in accordance with Condition 4 is overbarred.

Principles of duality probably lead the reader to anticipate the definition of SPO-S sets.

Definition 2: A set of subscripted literals in an SPO-S set (structure- and parity-observing \( S \) set) of a network whose logical model is \( G \) if and only if the following conditions hold.

Condition 1: Each literal of the set possesses a subscript consisting of a sequence of symbols corresponding to a sequence of edges constituting a path through \( G \) from an input vertex to an output vertex.

Condition 2: \( \mathcal{Z}_F[G] \) has a value of 0 whenever all of the literals of the set have values of 0, where \( F \) is any member of the set \( \mathcal{F} \) for the network that does not involve any of the lines of the network corresponding to edges in \( G \) denoted by symbols appearing in one or more of the subscripts of the literals of the set.

Condition 3: When any literal is removed from the set, Condition 2 no longer holds.

Condition 4: The set meets Conditions 4 and 5 of Definition 1.

Example 2: For the network whose logical model is shown in Fig. 1(a), the SPO-P sets are

\[
\begin{align*}
\{x_{cgk}; \bar{x}_{adgk}\} \\
\{x_{cgk}; y_{bdgk}\} \\
\{\bar{x}_{aehk}; y_{fhk}\} \\
\{y_{behk}; \bar{y}_{fhk}\}.
\end{align*}
\]

The SPO-S sets for the network are

\[
\begin{align*}
\{x_{cgk}; \bar{x}_{aehk}; \bar{y}_{behk}\} \\
\{x_{cgk}; y_{fhk}\} \\
\{\bar{x}_{adgk}; \bar{y}_{bdgk}; \bar{x}_{aehk}; \bar{y}_{behk}\} \\
\{\bar{x}_{adgk}; \bar{y}_{bdgk}; y_{fhk}\}.
\end{align*}
\]

We defer until later a discussion of how the SPO-P sets and SPO-S sets of a network are obtained.

We shall rarely use the SPO-P sets and SPO-S sets of a network themselves in this work. We have introduced them here primarily for two reasons: 1) to illustrate a useful extension of the concepts of \( P \) sets and \( S \) sets as defined by McCluskey; and 2) to facilitate the rigorous definition of SPOOF’s.

Definition 3: The disjunctive SPOOF of a network structure is a Boolean expression in disjunctive normal form, having one term for each \( SPO \) set of the network. Each such term consists of a conjunction of the subscripted literals of the corresponding SPO-P set.

Definition 4: The conjunctive SPOOF of a network structure is a Boolean expression in conjunctive normal form, having one term for each \( SPO \) set of the network. Each such term consists of a disjunction of the subscripted literals of the corresponding \( SPO \) set.

Example 3: In our introductory discussion of SPOOF’s, we
derived the disjunctive SPOOF for the network structure of Fig. 1(a):

\[
z = x_{c^g k} \bar{x}_{a^g k} \lor x_{c^g k} \bar{y}_{b^g k} \lor \bar{x}_{a^e h k} y_{f^h k} \lor \bar{y}_{b^e h k} y_{f^h k}.
\]

If, in the expansion of the various expressions occurring in the derivation of a SPOOF, we use the distributive property \( u \lor w = (u \lor v)(u \lor w) \) instead of the property \( u(v \lor w) = u(w \lor u) \), the end result of the derivation will be the conjunctive SPOOF of the network. Thus

\[
z = z_k
\]

\[
= (x_{c^g k} \lor \bar{x}_{a^g k}) \lor (x_{c^g k} \lor \bar{y}_{b^g k})
\]

\[
= z_{c^g k} z_{a^g k} \lor z_{c^g k} z_{f^h k}
\]

\[
= (x_{c^g k} \lor \bar{x}_{a^e h k}) \lor (x_{c^g k} \lor \bar{y}_{f^h k})
\]

\[
= (x_{c^g k} \lor \bar{x}_{a^e h k} \lor \bar{y}_{b^e h k}) \lor (x_{c^g k} \lor \bar{y}_{f^h k})
\]

\[
= x_{c^g k} x_{c^g k} \lor \bar{x}_{a^e h k} \lor \bar{y}_{b^e h k} \lor \bar{y}_{f^h k}
\]

In the applications to which we shall put them, whenever we are interested in the SPO-P sets (SPO-S sets) of a network, we will typically be interested in all the SPO-P sets (SPO-S sets) of that network. That is to say we will be using the disjunctive (conjunctive) SPOOF’s of networks rather than the individual SPO-P sets (SPO-S sets), which represent the component terms of the SPOOF’s. These SPOOF’s will be determined by deriving expressions for the network’s output function by working from the network output back to the network inputs, keeping track as we go of the various lines through which signals propagate and the parities of the signals on these lines—i.e., by employing the techniques we used in our introductory discussion of SPOOF’s and in Example 3. Once the disjunctive (conjunctive) SPOOF of a network is thus determined, the SPO-P sets (SPO-S sets) of the network structure may be determined by inspection of the terms of the SPOOF. This is in fact the method whereby the SPO-P sets and SPO-S sets of the structure shown in Fig. 1(a) that were presented in Example 2 were derived.

There remains at this point the question of whether the expressions derived in our introductory discussion and in Example 3 are in fact the SPOOF’s of the structure shown in Fig. 1(a) according to Definitions 3 and 4. This question is answered affirmatively by [3, theorem 5.1].

IV. APPLICATIONS OF SPOOF’S

One of the most useful applications of SPOOF’s results from the fact that it is exceedingly easy to determine the effect of any stuck-at fault on the output function of a network if the SPOOF (either disjunctive or conjunctive) of that network is known. This rather obvious application is demonstrated in the following.

**Theorem 1:** Let \( G \) be the logical model of a network and let \( z \) be a SPOOF of that network. Then for a fault \( F = a_1/l_1, a_2/l_2, \ldots, a_k/l_k \ (l_i \in \{0, 1\}) \), \( Z_F[G] \) may be determined as follows.

**Step 1:** Whenever any symbols \( a_i \) appear as a subscript to a literal in \( z \), let \( a_i \) denote that symbol which designates the closest edge to the output vertex of all the edges designated by symbols \( a_i \). (Since the subscript is a sequence of symbols describing a path through \( G \) from an input vertex to the output vertex, if any of the \( a_i \) appear in a literal’s subscript; the symbol designated \( a_i \) for that literal must necessarily be unique.)

**Step 2:** Replace the literal by \( l_j \), if no overbar appears over \( a_i \) in the subscript, and by \( \bar{l}_j \) otherwise.

**Step 3:** When Steps 1 and 2 have been completed for all \( a_i \) affected by the fault, remove all of the subscripts from the thus-modified SPOOF and simplify the resulting expression by the techniques of Boolean algebra.

This result is proved in [3, theorem 6.1]. Its utility is illustrated in the following example.

**Example 4:** As we have seen, the disjunctive SPOOF of the network structure shown in Fig. 1(a) is

\[
z = x_{c^g k} x_{a^g k} y_{b^g k} y_{f^h k} x_{a^e h k} y_{f^h k} y_{b^e h k} y_{f^h k}
\]

Suppose we wish to know the output function \( Z_F[G] \) where \( F = a_1/l_1, d_1/l_1, f_1/l_1 \). Using the technique of Theorem 1 we may find it easily

\[
Z_F[G] = x_{c^g k} \lor x_{c^g k} \lor 0 \lor y_{b^e h k} \lor y_{f^h k} \lor x_{a^e h k} y_{f^h k} y_{b^e h k} y_{f^h k}
\]

\[
= x \lor x \lor 0 \lor y_{b^e h k} \lor y_{f^h k} \lor x_{a^e h k} y_{f^h k} y_{b^e h k} y_{f^h k}
\]

\[
= x \lor y_{b^e h k} \lor y_{f^h k} \lor x_{a^e h k} y_{f^h k} y_{b^e h k} y_{f^h k}
\]

\[
= x \lor y_{b^e h k} \lor y_{f^h k} \lor x_{a^e h k} y_{f^h k} y_{b^e h k} y_{f^h k}
\]

We may determine \( Z_{F_1}[G] \), where \( F_1 = c/l_1 \), as

\[
Z_{F_1}[G] = x_{c^g k} \lor x_{c^g k} \lor 1 \lor y_{b^e h k} \lor y_{f^h k} y_{b^e h k} \lor y_{f^h k}
\]

\[
= x \lor y \lor x \lor y \lor y \lor y \lor y
\]

\[
= x \lor y
\]

Also, for \( F_2 = f_1 \),

\[
Z_{F_2}[G] = x_{c^g k} x_{c^g k} \lor x_{c^g k} \lor y_{b^e h k} \lor y_{b^e h k} \lor 1 \lor y_{f^h k}
\]

\[
= x \lor x \lor x \lor x \lor x \lor y
\]

\[
= x \lor y
\]

Thus we see that these latter two faults \( F_1 \) and \( F_2 \) are functionally equivalent in the terminology of [1] and [2].

The SPOOF’s of a network can also be used without great difficulty to find just which faults, if any, can affect that network’s output function in a given way, as we see in the following example.

**Example 5:** Let \( G \) be the structure shown in Fig. 1(a). Suppose we wish to ascertain for which faults \( F_i \), if any, \( Z_{F_i}[G] = \bar{x} \lor y \).

Let us number the literals of the disjunctive SPOOF of the network as follows:
Next, let us adopt the following shorthand notation.

1 \rightarrow 1 \  \text{Change literal} \ 1 \ \text{to} \ 1 \ (\text{where} \ l \ \in \ \{0, 1\}).
2 \rightarrow 1 \  \text{Do not change literal} \ 1 \ \text{to} \ 1.
3 \  \text{Do not change literal} \ 1 \ \text{at all.}

Using this notation, we see that any faults \( F \) for which \( \mathbf{Z}_{\text{F}}[G] = x \lor y \) must be such that, in using the SPOOF to find that fault's effect on the network output function, the following conditions must be fulfilled.

\text{Condition 1:} \  \text{We must get a term containing} \ x \ \text{but neither} \ y \ \text{nor} \ x.

\text{Condition 2:} \  \text{We must get a term containing} \ y \ \text{but neither} \ x \ \text{nor} \ y.

\text{Condition 3:} \  \text{We must not get the terms} \ x, \ y \ \text{or} \ x \ y.

Condition 1 will be fulfilled if we do the following:

\{1 \rightarrow 1, \ 2\} \ (1)

or

\{5, 6 \rightarrow 1\} \ (2)

Condition 2 will be fulfilled if we do the following:

\{5 \rightarrow 1, 6\} \ (3)

or

\{7 \rightarrow 1, 5\} \ (4)

And Condition 3 will be violated if we do the following:

\{2 \rightarrow 1, 1 \rightarrow 0\} \ (5)

or

\{3, 4\} \ (6)

or

\{3 \rightarrow 1, 4 \rightarrow 0\} \ (7)

or

\{3 \rightarrow 0, 4 \rightarrow 1\} \ (8)

or

\{7 \rightarrow 0, 5 \rightarrow 1\} \ (9)

We can do (1) only if we have a fault with a component \( c/1 \). If we are to have \( c/1 \) and not violation (7), we must also have a fault component \( b/1, d/0, g/1, \) or \( k/0 \). But we cannot have \( d/0, g/1, \) or \( k/0 \) if we are to do (1). Hence if we are to have (1), we must also have \( b/1 \). Next, we seek to do either (3) or (4). If we are to do (3), we must have a component \( a/0 \) or \( e/1 \), but if we have \( a/0 \), we cannot do (1) as we have supposed. Thus we must have \( e/1 \). Since we can have \( e/1 \) without requiring any other components to avoid violations of required conditions, we have found one such fault such that \( \mathbf{Z}_{\text{F}}[G] = x \lor y \), viz., \( F = b/1, c/1, e/1 \).

Let us next attempt to find a suitable fault by doing (1) and (4), rather than (1) and (3). We can do (4) only if we have a fault component \( b/0 \) or \( e/1 \), but if we have \( b/0 \), we cannot do (1) as supposed. Thus, here again, we must have \( e/1 \) and are led to \( F = b/1, c/1, e/1 \). If one attempts to fulfill Condition 1 by doing (2) instead of (1), he will discover this to be impossible under the given restraints. From this, we may in turn conclude that \( F = b/1, c/1, e/1 \) is the only fault in the set \( \mathbf{F} \) for the network of Fig. 1(a) for which \( \mathbf{Z}_{\text{F}}[G] = x \lor y \).

The technique illustrated in this example appears perhaps at first glance to be arduous. If the reader practices it with other examples of his own choosing, however, the author suspects that the reader will soon agree with him that this technique is one of the many things in life that are much easier to do than to describe.

In addition to the above applications, the SPOOF is useful in other applications as well. For example, techniques using SPOOF's for establishing upper bounds on, and in some cases exact counts of, the number of equivalence classes of faults occurring in a logic network are discussed in [3].

V. CONCLUSION AND PROJECTIONS

In this paper, we have demonstrated that knowledge of the structure of a given logic network must in general be available before one can analyze the effects of failures, not only on the structural characteristics, but also on the functional behavior of that network. We have developed the SPOOF—a convenient and compact means by which this information about network structure may be formulated in an algebraic expression resembling a disjunctive—or conjunctive-normal-form Boolean expression for the network output function.

Although the definitions and other formal results developed in this paper use the concept of “inverting vertices” and therefore apply only to networks consisting of AND, OR, NAND, NOR, and NOT gates, experience has shown that SPOOF's are equally useful in dealing with networks containing other gate types as well. Work is presently underway that will remove the restrictions imposed by the concept of inverting vertices and extend the applicability of the formal results to these more general networks.

Useful applications of SPOOF's as presented in this paper include the analysis of the effects of a given fault on the output function of the network in which it occurs, and the determination of just which faults, if any, can affect a network's output in a specified way. It has been, furthermore, pointed out that SPOOF's may be used to assist one in establishing upper bounds on, and in some cases exact counts of equivalence classes.

In addition to the above-indicated applications in which the SPOOF has already shown itself to be valuable, it is expected that the SPOOF may find use in a number of applications not yet explored as a versatile and powerful tool for the analysis of faults in logic networks and their effects. This expectation is enhanced by the fact that either a disjunctive or a conjunctive SPOOF is isomorphic to the network that it describes.\(^3\)

\(^3\)The proof of this assertion is readily accomplished by demonstration of a procedure for reconstructing the logical model of a network given either of its SPOOF's. This procedure is the subject of a short forthcoming paper.
It is readily apparent that all information contained in a network's P sets and S sets, as defined and used by McCluskey [6], may be trivially obtained from that network's SPOOF's. This fact suggests that it would probably be, not only practical, but economically desirable to analyze a logic network that has been proposed for production for its characteristics in the presence of faults (perhaps with the aim of deriving diagnostic tests) at the same time it is analyzed for potential timing difficulties posed by static and dynamic hazards [6]. Such a dual-purpose analysis could begin with the derivation of the SPOOF's of the network in question (or alternatively, its SPO-P sets and SPO-S sets). Derivation of SPOOF's, in turn, may be seen to be a task likely to be amenable to being carried out by a computer. Thus the SPOOF is a tool likely to be particularly valuable to researchers seeking more sophisticated computer-aided design procedures.

It is felt that the results described in this paper, the techniques developed, and the applications suggested constitute further progress toward a goal first expressed by this author in [1]: the use of an algebraic approach to develop a general, formal theory of faulty digital systems that will be useful to those concerned with digital system reliability in the same way that the more traditional "theory of switching circuits" has been useful to those concerned only with "healthy" digital systems.

REFERENCES


Frederick W. Clegg (S'69-M'70) was born in Atlanta, Ga., on October 9, 1944. He received the B.S. degree in engineering science (magna cum laude) from Oakland University, Rochester, Mich., in 1965. From 1965 to 1966 he studied in the areas of information processing and control engineering at the Technischen Hochschule Darmstadt, Germany, under a fellowship from the Deutschen Akademischen Austauschdienst. In 1966 he returned to the United States and continued graduate studies at Stanford University, Stanford, Calif. He received the M.S. degree in electrical engineering and the Ph.D. degree in electrical engineering and computer science from Stanford University in 1967 and 1970, respectively. From 1968 to 1969 he served as a Research Assistant in Electrical Engineering and from 1969 to 1970 he served as an Instructor of Electrical Engineering at Stanford University. In 1970 he joined the Department of Electrical Engineering at the University of Santa Clara, Santa Clara, Calif., as an Assistant Professor. At Santa Clara, he has been engaged in the creation of a new digital systems laboratory for both undergraduate and graduate students and in the expansion and strengthening of the University's programs in digital systems and computer science. In 1971 he proposed and supervised the implementation of a major reorganization and enlargement of the digital systems and computer science curriculum of the Graduate School of Engineering at Santa Clara. During 1971 and 1972 he created and organized a half-credit new course at both the graduate and undergraduate levels in this area. Due largely to his efforts in these directions, the department with which he serves was renamed the Department of Electrical Engineering and Computer Science in 1972. His current research interests lie in the areas of computer reliability and equipment for academic digital laboratories. He is the author or coauthor of several papers in both these fields. He is presently working on a book, Introduction to Switching Circuits and Digital System Design, to be published by Prentice-Hall.

Dr. Clegg is a member of Eta Kappa Nu, Sigma Xi, and the American Association for the Advancement of Science. He presently serves as Treasurer of the University of Santa Clara Sigma Xi Club and as Treasurer of the San Francisco Chapter of the IEEE Computer Society.