THE SPOOF

A New Technique for Analyzing the Effects of Faults on Logic Networks

by

Frederick W. Clegg

August 1970

Technical Report No. 11

Prepared under

Joint Services Electronics Programs
U.S. Army, U.S. Navy, and U.S. Air Force
under
Contract N-00014-67-A-0112-0044 and
National Science Foundation Grant GJ 165

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY . STANFORD, CALIFORNIA

SEL-70-073

THE SPOOF

A New Technique for Analyzing the Effects of Faults on Logic Networks

by

Frederick W. Clegg

August 1970

Technical Report No. 11

Prepared under

Joint Services Electronics Programs
U. S. Army, U. S. Navy, and U. S. Air Force
under
Contract N-00014-67-A-0112-0044
and
National Science Foundation Grant
GJ 165

Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford University Stanford, California

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 SPOOF or structure-and parity-observing output function described in this report 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 which illustrate the ease with which the effects of any "stuck-at' fault - single or multiple - on the functional characteristics of a logic network are determined using SPOOF's.

CONTENTS

THE SPOOF
A NEW TECHNIQUE FOR ANALYZING
THE EFFECTS OF FAULTS ON LOGIC NETWORKS

1. Introduction

One area of ever increasing importance and concern in the technology of computers and other digital systems is that of reliability. In order to approach the problems associated with reliability in a more thorough and general manner than usual, this author and several of his colleagues at Stanford have, over the past several years, investigated the fundamental effects of failures on logic circuitry. The approaches taken have been algebraic in nature -- allowing a more rigorous and generally applicable treatment of the subject than might have been otherwise possible.

An algebraic approach to the study of the effects of faults in logic networks has already yielded a number of highly interesting and powerful results, many of which are described in a previous Digital Systems Laboratory Report, Algebraic Properties of Faults in Logic Networks [1], co-authored by E. J. McCluskey and the present writer. The techniques described in the present report are further results of the same study. Thus this report may be regarded as a sequel to the work cited above and will presume a familiarity with its contents, as the same nomenclature, symbology, and modelling schemes used in the earlier report will be employed here throughout.

2. The Need for Knowledge of Network Structure in Determining the Effects of Faults on a Network's Function

Theorems 6.9(a) and (b) of [1] provide a certain amount of knowledge about the effects of certain faults on particular networks and their output functions and about functional equivalence relations between these faults given only knowledge of the Boolean function which the network implements. Unfortunately, however, one generally needs 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 2.1

Figure 2.1 shows two irredundant realizations of the exclusive-OR function. The familiar structure of Fig. 2.1 employs four two-input NAND gates while the structure of Fig. 2.1a utilizes three two-input NAND gates in addition to two inverters.

If the realization of Fig. 2.1a 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 $\mathcal{F}$ for the structure shown in Fig. 2.1b there is no fault at all which 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 2.1 demonstrates the fact that, in order to know fully how the output function of a given network may be affected by various faults

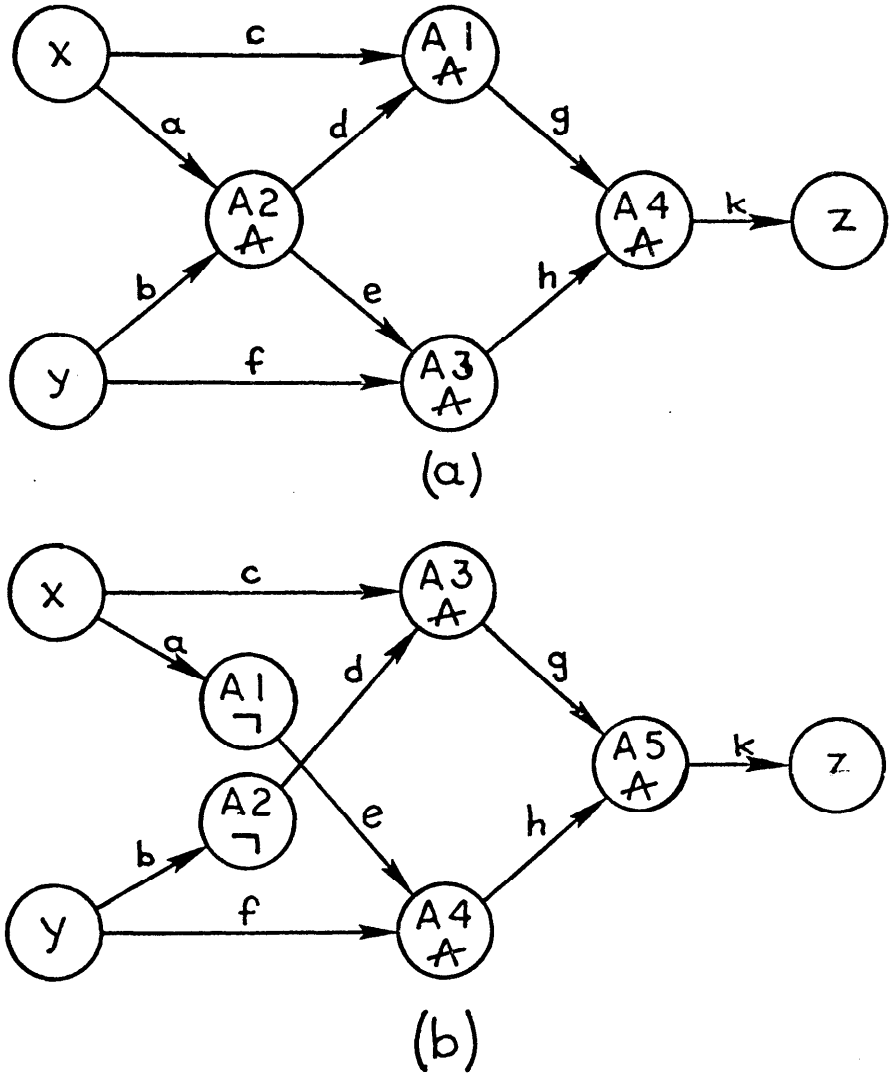


Fig. 2.1

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 $z = x \vee \bar{y}$.

(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.

3. A Means of Characterizing Network Structure in 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 exclusive-OR network whose structure is shown in Fig. 2.1a. The logic signal at the output vertex of this structure is just the logic signal on edge k . 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 logical model G of the network) we proceed writing

$$\begin{aligned} z &= z_k \\ &= \overline{(z_g \; z_h)}_k \\ &= (\bar{z}_g \vee \bar{z}_h)_k \\ &= z_{\bar{g}k} \vee \bar{z}_{hk} . \end{aligned}$$

The reader will notice that we are affixing complement designations (over-bars) 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

$$z = \bar{z}_{\bar{g}k} \vee z_{\bar{h}k}$$

$$= \overline{(\bar{z}_{\bar{c}} \vee \bar{z}_{\bar{d}})_{\bar{g}k}} \vee \overline{(\bar{z}_{\bar{e}} \vee \bar{z}_{\bar{f}})_{\bar{h}k}}$$

$$= z_{c\bar{g}k}\, z_{d\bar{g}k} \vee z_{e\bar{h}k}\, z_{f\bar{h}k} \quad .$$

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

$$z = z_{c\bar{g}k}\, z_{d\bar{g}k} \vee z_{e\bar{h}k}\, z_{f\bar{h}k}$$

$$= x_{c\bar{g}k}\, (\bar{z}_{\bar{a}} \vee \bar{z}_{\bar{b}})_{d\bar{g}k} \vee (\bar{z}_{\bar{a}} \vee \bar{z}_{\bar{b}})_{e\bar{h}k}\, y_{f\bar{h}k}$$

$$= x_{c\bar{g}k}\, (\bar{x}_{\bar{a}} \vee \bar{y}_{\bar{b}})_{d\bar{g}k} \vee (\bar{x}_{\bar{a}} \vee \bar{y}_{\bar{b}})_{e\bar{h}k}\, y_{f\bar{h}k}$$

$$z = x_{cgk}\, \bar{x}_{\bar{a}\, d\, \bar{g}\, k} \vee x_{c\bar{g}k}\, \bar{y}_{\bar{b}\, d\, \bar{g}\, k}$$

$$\vee \bar{x}_{\bar{a}\, e\, \bar{h}\, k}\, f\bar{h}k \vee \bar{y}_{\bar{b}\, e\, \bar{h}\, k}\, y_{f\bar{h}k}$$

This resulting expression of the output function of the network will be called a disjunctive SPOOF. The word SPOOF is an acronym for structure- and parity - observing output function and is in no way intended to indicate that the thus-designated Boolean expression should not be taken seriously. The adjective disjunctive indicates that we have used the distributive property $u\,(v \vee w) = uv \vee uw$ to obtain an expression in disjunctive normal form (i.e. "sum-of-products" form). By using instead, the distributive property $u \vee VW = (u \vee v)\,(u \vee w)$, one can

obtain a similar expression for the network output function in "product of sums" form called a conjunctive SPOOF,

It is of essential importance to note that the properties of ordinary Boolean algebra $u \bar{u} = 0$ and $u \vee \bar{u} = 1$ must not be used to "cancel" terms in the derivation of a SPOOF.

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 [2] 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 "true" or complemented) associated with that literal on the line associated with that subscript.

The SPOOF which we have introduced also, it may be noted, exhibits similarities to the output functions expressed in terms of "literal propositions" as developed by Poage [3]. To readers familiar with the work of Reference [3], 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 which is 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 [4]. Before we formally define SPOOF's, we introduce the concepts of S P 0 - P sets and S P 0 - S sets.

4. SPO-P Sets, SPO-S Sets, and SPOOF'S

Definition 4.1:

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 (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; (2) $\mathfrak{Z}_F[G]$ 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 $\mathfrak{F}$ for the network which has no components affecting 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; (3) when any literal is removed from the set, condition 2 no longer holds; (4) whenever, along the path described 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*, the symbol a is overbarred; and (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 4.2:

A set of subscripted literals is an SPO-S set (structure - and parity - observing S set) of a network whose logical model is G if and only if (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; (2) $\mathfrak{Z}_F[G]$ 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 $\mathfrak{F}$ for the network which has no components affecting 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; (3) when

* A-vertices, V-vertices, and l-vertices will hereafter be called inverting vertices. Certain other types of vertices (e.g. @-vertices) cannot be classified as "inverting" or "non-inverting." Therefore all formal results concerning SPO-P sets, SPO-S sets, and SPOOF's in this work will be applicable only to networks containing only AND, OR, NAND, NOR, and NOT gates. Experience with numerous examples indicates, however, that SPOOF's are also useful in dealing with networks not meeting this restriction.

any literal is removed from the set, condition 2 no longer holds; and (4) the set meets conditions 4 and 5 of Definition 4.1.

Example 4.1

For the network whose logical model is shown in Fig. 2.1a, the SPO-P sets are

$$\{x_{cgk} ; \bar{x}_{\bar{a}\bar{d}\bar{g}k}\}$$
$$\{x_{cgk} ; \bar{x}_{\bar{b}\bar{d}\bar{g}k}\}$$
$$\{\bar{x}_{\bar{a}\bar{e}\bar{h}k} ; y_{\bar{f}\bar{h}k}\}$$
$$\{\bar{y}_{\bar{b}\bar{e}\bar{h}k} ; y_{\bar{f}\bar{h}k}\} .$$

The SPO-S sets for the network are

$$\{x_{cgk} ; \bar{x}_{\bar{a}\bar{e}\bar{h}k} ; \bar{y}_{\bar{b}\bar{e}\bar{h}k}\}$$
$$\{x_{cgk} ; y_{\bar{f}\bar{h}k}\}$$
$$\{\bar{x}_{\bar{a}\bar{d}\bar{g}k} ; \bar{y}_{\bar{b}\bar{d}\bar{g}k} ; \bar{x}_{\bar{a}\bar{e}\bar{h}k} ; \bar{y}_{\bar{b}\bar{e}\bar{h}k}\}$$
$$\{\bar{x}_{\bar{a}\bar{d}\bar{g}k} ; \bar{y}_{\bar{b}\bar{d}\bar{g}k} ; y_{\bar{f}\bar{h}k}\} .$$

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 4.3:

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

Definition 4.4:

The conjunctive SPOOF of a network structure is a Boolean expression in conjunctive normal form, having one factor for each SPO-S set of the network. Each such factor consists of a disjunction of the subscripted literals of the corresponding SPO-S set.

Example 4.2:

In our introductory discussion of SPOOF's, we derived the disjunctive SPOOF for the network structure of Fig. 2.1a:

$$z = x_{c@}\,\bar{x}_{adgk} \vee x_{cgk}\,\bar{y}_{bdgk} \vee \bar{x}_{aehk}\,y_{fhk} \vee \bar{y}_{behk}\,y_{fhk}$$

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

$$\begin{aligned}
z &= z_k \\
&= \bar{z}_{gk} \vee \bar{z}_{hk} \\
&= \overline{(\bar{z}_c \vee \bar{z}_d)_{gk}} \vee \overline{(\bar{z}_e \vee \bar{z}_f)_{hk}} \\
&= z_{cgk}\,z_{dgk} \vee z_{ehk}\,z_{fhk} \\
&= (z_{cgk} \vee z_{ehk})(z_{cgk} \vee z_{fhk}) \\
&\quad \wedge (z_{dgk} \vee z_{ehk})(z_{dgk} \vee z_{fhk}) \\
&= (x_{cgk} \vee (\bar{z}_a \vee \bar{z}_b)_{ehk})(x_{cgk} \vee y_{fhk}) \\
&\quad \wedge ((\bar{z}_a \vee \bar{z}_b)_{dgk} \vee (\bar{z}_a \vee \bar{z}_b\,e_{hk})) \\
&\quad \wedge ((\bar{z}_a \vee \bar{z}_b)_{dgk} \vee y_{fhk})
\end{aligned}$$

$$= (x_{c\bar{g}\bar{k}} \vee \bar{x}_{\bar{a}e\bar{h}\bar{k}} \vee \bar{y}_{\bar{b}e\bar{h}\bar{k}})$$
$$\wedge (x_{c\bar{g}\bar{k}} \vee y_{f\bar{h}\bar{k}})$$
$$\wedge (\bar{x}_{\bar{a}d\bar{g}\bar{k}} \vee \bar{y}_{\bar{b}d\bar{g}\bar{k}} \vee \bar{x}_{\bar{a}e\bar{h}\bar{k}} \vee \bar{y}_{\bar{b}e\bar{h}\bar{k}})$$
$$\wedge (\bar{x}_{\bar{a}d\bar{g}\bar{k}} \vee \bar{y}_{\bar{b}d\bar{g}\bar{k}} \vee y_{f\bar{h}\bar{k}})$$

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 (factors) 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 4.2. 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 (factors) of the SPOOF. This is in fact the method whereby the SPO-P sets and SPO-S sets of the structure shown in Fig. 2.1a which were presented in Example 4.1 were derived.

5. The Derivation of SPOOF's

There remains at this point the question of whether the expressions derived in our introductory discussion and in Example 4.2 are in fact the SPOOF's of the structure shown in Fig. 2.1a according to Definitions 4.3 and 4.4. This question is resolved in the following theorem.

Theorem 5.1

Let z be the Boolean expression in disjunctive (conjunctive) normal form obtained by the following method: working from the output vertex of a network structure back through the structure to the input vertices of that structure; expressing the signal appearing on the edges incident out from each vertex in terms of (1) the signals appearing on edges incident into that vertex if that vertex corresponds to a gate in the network, or (2) the signal coming into the structure at that vertex if that vertex is an input vertex; describing the path through which each such signal propagates to the output vertex by a subscript consisting of a string of symbols denoting edges in that structure; describing the parity of each such signal on each edge by an overbar over the subscript symbol denoting that edge if, along the path described by the subscript, there are an odd number of inverting vertices between the edge in question and the output vertex and by no overbar otherwise; using the distributive property $u \vee vw = (u \vee v)(u \vee w)$; and not cancelling any literals using the properties $u\bar{u} = 0$ and $u \vee \bar{u} = 1$. Then z is the disjunctive (conjunctive) SPOOF of that network structure.

Proof:

Without loss of generality we restrict our explicit attention here to the disjunctive expression. We must show that each term of the expression derived by the method of the theorem is in fact an SPO-P set according to Definition 4.3.

Clearly condition 1 of the Definition 4.3 is met. The theorem stipulates that at every point in the derivation of the expression, the path through which the signal under consideration propagates to the output be described by a string of symbols indicating which edges comprise that path in the network structure,

The theorem stipulates that the expression be in disjunctive normal form as obtained by using the distributive property $\cdot \vee = \cdot$. By the very nature of an expression in disjunctive normal form, each term expresses a condition guaranteeing that the value of the expression will be 1 when all the literals in that term have values of 1. Even if the network under consideration

contains faults, as long as none of these faults affects any edges in paths indicated by subscripts of the literals in that term, the signal paths required for this guarantee to still hold will still be intact and the value of the network output signal will be 1 whenever the values of all the literals in the term are 1. Thus condition 2 is met.

If any literal of a given term has value 0 and none of the edges indicated in subscripts of any literals in that term are affected by faults, then that term cannot cause the value of the output function to be 1. If this is the case, since it is possible that every other term in the expression may have value 0, the value of the output function itself may in this case be 0. Therefore condition 3 of Definition 4.3 is met.

Conditions 4 and 5 are met directly by stipulations of the theorem. Hence we may conclude that the expression derived by the method of the theorem is a disjunctive SPOOF.

The proof of the theorem as it regards the derivation of conjunctive SPOOF's is exactly analogous.

6. 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 given as a theorem.

Theorem 6.1:

Let G be the logical model of a network and let z be a SPOOF of that network. Then for a fault $F = a_1/\ell_1, a_2/\ell_2, \ldots, a_k/\ell_k$ $(\ell_i \in \{0,1\})$, $\mathfrak{Z}_F[G]$ may be determined as follows:

(1) Whenever any symbols a_i appear as a subscript to a literal in z, let a_j 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_j for that literal must necessarily be unique.)

(2) Replace the literal by ℓ_j, if no overbar appears over a_j in the subscript, and by $\overline{\ell_j}$ otherwise.

(3) When steps 1 and 2 above 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.

Proof:

Suppose z is the disjunctive SPOOF of the network. By definition, each term of z will be a conjunction of the elements of an SPO-P set and thus expresses a condition which can cause the output function $\mathfrak{z}_F[G]$ to have a value of 1. This condition is that all the literals of the term have a value of 1 and that none of the edges of G indicated by the subscripts of the literals of the term be affected by the fault F. The requirement that the edges indicated by the subscripts be unaffected by faults stems, of course, from the fact that the subscripts of the literals of each term designate the paths through G along which the input signals indicated by the literals of that term must propogate if these input signals above are to force the output to assume a value of 1.

The reader will observe that, in any SPOOF, the last (rightmost) symbol in every subscript will indicate that edge of G which is incident into the output vertex. This is because the last edge in any path through G from an input vertex to the output vertex will necessarily be the edge which is itself incident into the output vertex. Furthermore, we note that the last symbol in any subscript of any SPOOF will never possess an overbar. This is an immediate consequence of the definition of SPO-P sets and the fact that there will certainly never be any inverting vertices along the "path" from the edge incident into the output vertex and the output vertex itself.

If We nowdirect our attention backwards through G , from the output vertex toward the input vertices, we can make the following observation: if all the literals of an **SPO-P** set have a value of 1 and none of the edges of G designated by symbols in subscripts of the literals of that set are affected by a fault, then every line in the network corresponding to an edge in G denoted by these subscript symbols will have on it a signal of value 0 if the corresponding subscript symbol possesses an overbar, and a signal value of 1 otherwise. This follows from our convention that assigns a subscript symbol an overbar if it designates an edge such that, along the path indicated by the subscript, an odd number of inversions take place between the edge in question and the output.

Let us now consider the effect on the network output function of a fault having components affecting edges in the path through G described by the subscript of a particular literal, say x^* , of a particular SPO-P set, say P . Suppose a_j is the closest edge affected by a fault component to the output vertex in the path through G described by the subscript of x^* . Let the fault component affecting a_j be a_j/ℓ_j . Then, if $\ell_j = 1$ [$\ell_j = 0$] and the symbol a_j bears no overbar [bears an overbar] in the subscript of x^* , the value of the signal on each edge in the path described by the subscript of x^* downstream of a_j will be that value of the signal which would have resulted, in the fault-free network, from an input causing the literal x^* itself to have value 1 . This will be the case whether or not the input signals to the faulty network in fact cause the value of x^* to be 1 . In other words, the signals on all lines downstream of a_j in the path described by x^*'s subscript are same as they would be if x^* had value 1 , whether or not x^* does in fact have value 1 . Thus we could replace the value of x^* -- be it 1 or 0 -- with the value 1 without affecting $\mathfrak{Z}_F[G]$.

Thus we see that the existence of any SPO-P set, say P , of G satisfying both the following conditions will cause $\mathfrak{Z}_F[G]$ to have a value of 1 :

(a) every literal of P whose subscript contains no symbols designating edges of G which are affected by components of F has value 1 ;

and

(b) for every literal of P whose subscript contains symbols designating edges of G which are affected by components of F , if a_j is the closest such edge to the output (as discussed above) in the path described by that literal's subscript and ℓ_j is the value at which a_j is stuck by F , then $\ell_j = 1$ if a_j does not possess an overbar in that literal's subscript and $\ell_j = 0$ otherwise.

But the existence of such an SPO-P set directly implies that there will be a term having value 1 in the Boolean expression resulting from application of the technique of the theorem to the disjunctive SPOOF of G . Therefore, the value of the expression obtained by the technique of the theorem will be 1 whenever the value of $\mathfrak{Z}_F[G]$ is 1 .

To complete the proof of the theorem, suppose that there exists no SPO-P set of G satisfying both (a) and (b) above. Then every SPO-P set, P , of G will satisfy one of the following conditions:

(c) at least one literal of P whose subscript contains no symbols designating edges of G affected by components of F has value 0 ;

or

(d) for at least one literal of P whose subscript contains symbols designating edges affected by F , if a_j and ℓ_j for that literal are defined as above, $\ell_j = 0$ if a_j does not bear an overbar in that literal's subscript, and $\ell_j = 1$ otherwise.

Clearly an SPO-P set satisfying (c) cannot cause the value of the output function to be 1 . This is because the path described by the subscript of at least one of the literals of P is unaffected by any components of F . Thus, it is not possible for the signals on all the paths through G described by the

subscripts of the literals of P to have such values as are required to force $\mathcal{F}_F[G]$ to have value 1 .

Suppose P satisfies (d) but not (c) . Then the signal on a_j for the literal(s) of P for which (d) holds has a value opposite the value which must appear at that point if the signals on all the paths indicated by the subscripts of the literals of P are to have the values necessary to cause the output function to have a value of 1 .

Since there is therefore no SPO-P set whose literals' subscripts describe a set of paths through G , the signals on which have the values necessary to cause the output function to have a value 1 , we must conclude that, under circumstances in which every SPO-P set of G satisfies (c) or (d) above, $\mathcal{F}_F[G]$ will have value 0 .

But clearly any term in the disjunctive SPOOF of G corresponding to an SPO-P set satisfying (c) or (d) above will yield a term of value 0 in the expression obtained by applying the technique of the theorem to the SPOOF. This is an immediate consequence of the fact that any SPO-P set satisfying (c) or (d) will either contain a literal which has value 0 because of the given input conditions or a literal which will be replaced by 0 by the technique of the theorem.

Therefore, whenever conditions exist such that $\mathcal{F}_F[G]$ has value 0 , the value of the expression obtained from the technique of the theorem will also have value 0 .

This completes the proof of the theorem as applied to the disjunctive SPOOF of the network Verification of the technique as applied to the conjunctive SPOOF of a network may be carried out in an exactly analogous fashion by virtue of the principles of duality.

The utility of this technique is illustrated in the following example.

Example 6.1

As we have seen, the disjunctive SPOOF of the network structure shown in Fig. 2.1a is

$$z = x_{c\bar{g}k}\ \bar{x}_{\bar{a}d\bar{g}k} \vee x_{c\bar{g}k}\ \bar{y}_{\bar{b}d\bar{g}k} \vee \bar{x}_{\bar{a}e\bar{h}k}\ y_{f\bar{h}k} \vee \bar{y}_{\bar{b}e\bar{h}k}\ y_{f\bar{h}k}$$

Suppose we wish to know the output function $\mathfrak{Z}_F[G]$ where $F = a/1, d/1, f/1$. Using the technique of Theorem 6.1 we may find it easily:

$$\begin{aligned}\mathfrak{Z}_F[G] &= x_{c\bar{g}k}\ 1 \vee x_{c\bar{g}k}\ 1 \vee 0\ y_{f\bar{h}k} \vee \bar{y}_{\bar{b}e\bar{h}k}\ 1 \\ &= x \vee x \vee 0 \vee \bar{y} \\ &= x \vee \bar{y}\end{aligned}$$

We may determine $\mathfrak{Z}_{F_1}[G]$, where $F1 = c/1$, as

$$\begin{aligned}\mathfrak{Z}_{F_1}[G] &= 1\ \bar{x}_{\bar{a}d\bar{g}k} \vee 1\ \bar{y}_{\bar{b}d\bar{g}k} \vee \bar{x}_{\bar{a}e\bar{h}k}\ y_{f\bar{h}k} \vee \bar{y}_{\bar{b}e\bar{h}k} y_{f\bar{h}k} \\ &= \bar{x} \vee \bar{y} \vee \bar{x}\ y \vee \bar{y}\ y \\ &= \bar{x} \vee \bar{y}\end{aligned}$$

Also, for $F_2 = f/1$,

$$\mathfrak{Z}_{F_2}[G] = x_{cgk}\,\bar{x}_{adgk} \vee x_{cgk}\,\bar{y}_{bdgk} \vee \bar{x}_{aehk}\,1 \vee \bar{y}_{behk}\,1$$
$$= x\,\bar{x} \vee x\,\bar{y} \vee \bar{x} \vee \bar{y}$$
$$= \bar{x} \vee \bar{y}$$

Thus we see again that these latter two faults F_1 and F_2 are functionally equivalent.

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 6.2:

Let G be the structure shown in Fig. 2.1a. Suppose we wish to ascertain for which faults F_j, if any, $\mathfrak{Z}_{F_j}[G] = \bar{x} \vee y$.

Let us number the literals of the disjunctive SPOOF of the network as follows:

$$z = \underset{①}{x_c}\,\underset{②}{\bar{x}_{adgk}} \vee \underset{③}{x_{cgk}}\,\underset{④}{\bar{y}_{bdgk}} \vee \underset{⑤}{\bar{x}_{aehk}}\,\underset{⑥}{y_{fhk}} \vee \underset{⑦}{\bar{y}_{behk}}\,\underset{⑧}{y_{fhk}}$$

Next, let us adopt the following shorthand notation

$(i) \rightarrow \ell$	means "change literal (i) to ℓ (where $\ell \in \{0,1\}$)"
$(i) \not\rightarrow \ell$	means "do not change literal (i) to ℓ"
$\underline{(i)}$	means "do not change literal (i) at all"

Using this notation, we see that any faults F_j for which $\mathfrak{Z}_{F_j}[G] = \bar{x} \vee y$ 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:

(1) We must get a term containing $\bar{x}$ but neither $\bar{y}$ nor x .

and

(2) We must get a term containing y but neither $\bar{x}$ nor $\bar{y}$.

and

(3) We must not get the terms x , $\bar{y}$ or x $\bar{y}$.

Condition (1) will be fulfilled if we do the following:

(α) ① $\longrightarrow$ 1 , ②

or

(β) ⑤ , ⑥ $\longrightarrow$ 1

Condition (2) will be fulfilled if we do:

(γ) ⑤ $\longrightarrow$ 1 , ⑥

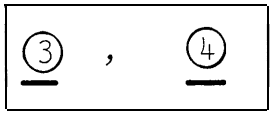
or

(δ) ⑦ $\longrightarrow$ 1 , ⑧

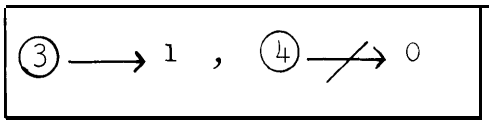
And condition 3) will be violated if we do:

(ε) ② $\longrightarrow$ 1 , ① $\not\longrightarrow$ 0

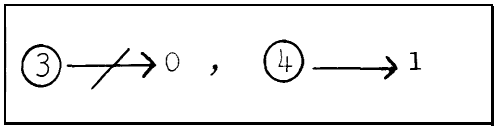
or

(ζ) 

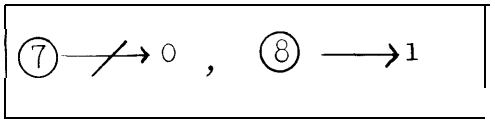
or

(η) 

or

(θ) 

or

(ι) 

We can do (α) only if we have a fault with a component c/1. If we are to have c/1 and not violation (η), we must also have a component b/1 or d/0, but we cannot have d/0 if we do (α). Thus if we are to have c/1, we must also have b/1. If we have b/1, however, we cannot do (δ), and hence must do (γ). If we are to do (γ), we must have a component a/0 or e/1, but if we have a/0, we cannot do (α) 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 $\mathfrak{z}_F[G] = \bar{x} \vee y$, viz. F = b/1,c/1,e/1. If one attempts to fulfil condition (1) by doing (β) instead of (α), he will discover this to be impossible under the given restraints. From this, we may

in turn conclude that F = b/1,c/1,c/1 is the only fault in the set $\mathfrak{F}$ for the network of Fig. 2.1a for which $\mathfrak{Z}_F[G] = \bar{x} \vee y$.

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 which are much easier to do than to describe.

7. Enumeration of Equivalence Classes

When one sets out to study a particular network and the various effects which faults may have upon its structure and output function, it is often desirable to have some rough idea of the number of equivalence classes induced on the set $\mathfrak{F}$ for that network by the equivalence relations defined and discussed in Reference [1]. Theorems 6.11 and 6.12 of Reference [1] established exact counts of the number of classes into which the set $\mathfrak{F}_s$ -- i.e. the set of single faults -- for a network is partitioned by the relations of A-structural equivalence and $\mathfrak{R}$-structural equivalence. More recent results have made it possible, through the use of SPOOF's and other techniques introduced below, to approach this enumeration problem as it relates to the entire set $\mathfrak{F}$ (i.e. multiple as well a single faults) for a network under partitioning by the relation of functional equivalence as well as by the structural equivalence relations.

We shall first develop a technique for establishing upper bounds on, and, in some case, exact counts of the number of classes into which the set $\mathfrak{F}$ is partitioned by the structural equivalence relations. We introduce the basic approach to be employed with reference to the simple network whose logical model is shown in Fig. 7.1.

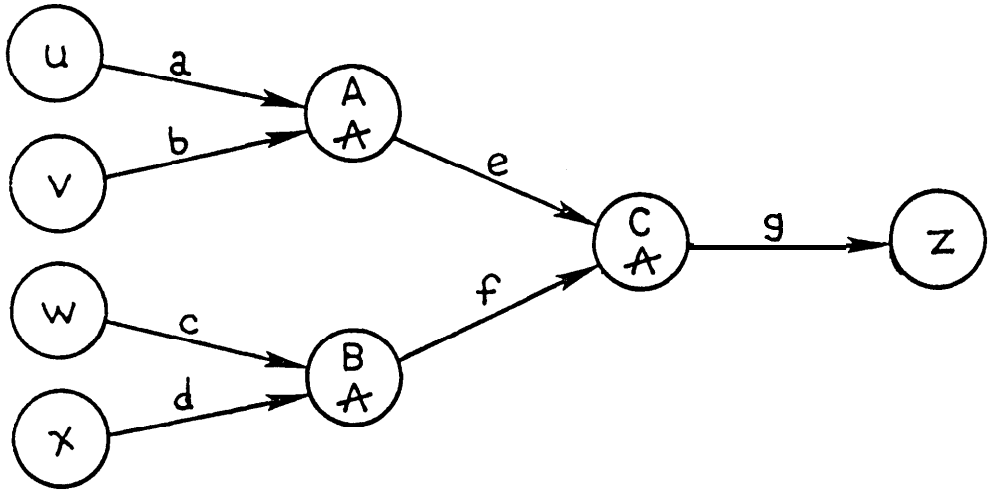


Fig. 7.1

The logical model of a simple logic network. It is shown in the text that the set $\mathcal{F}$ for this structure is partitioned into $r_d = 39$ distinct classes by the relation of d-Structural equivalence.

Let us consider vertex C and the two edges e and f which are incident into it. There are five different subgraphs which can appear as that part of the primary logical model for this network in the presence of some fault which drives* edge e Faults (or sets of fault components' yielding each of these different subgraphs are (1) λ , (2) a/1 , (3) b/1 , (4) a/l , b/l , and (5) a/O . Similarly, there are five different subgraphs which can appear as that part of the logical model which serves to drive edge f . If neither e nor f is affected by a fault component, there are 5x5 = 25 different subgraphs which can possibly drive edge g which is incident out from vertex C, in the primary logical models of the network in presence of various faults. If e is stuck at 1, then there are only 5 different subgraphs which can drive g . Similarly there are 5 different subgraphs which can drive g if f is stuck at 1. If both e and f are stuck at 1, there is only one subgraph which can drive g , viz. the subgraph containing only vertex C where C is a fault bias vertex of value 0. If either e or f is stuck at 0, the only subgraph which can drive g is that subgraph containing only vertex C where C is a fault bias vertex of value 0. Thus there are altogether 25 + 5 + 5 + 1 + 1 = 37 different subgraphs which can drive edge g in the presence of various faults. Driving the output vertex itself, there are 37+ 2 = 39 different possible subgraphs. The additional two subgraphs are those corresponding to g/0 and g/1 . Thus there are 39 different primary logical models which can result from faults affecting this network structure. In other words, the relation of A-structural equivalence partitions the

*We shall say that a subgraph $\hat{G}$ of a directed graph G "drives" an edge e (or vertex V) of G if and only if there is a path from every edge and vertex in $\hat{G}$ through $\hat{G}$ to e ( )V.

$3^7 = 2187$ elements of the set $\mathfrak{F}$ for this network into 39 equivalence classes.

We go now to the general application of this technique of finding the number of subgraphs which can drive an edge incident out from a vertex in terms of the numbers of subgraphs which can drive edges incident into that vertex.

Theorem 7.1:

Let G be the logical model of a network constructed exclusively of AND, OR, NAND, NOR, and NOT gates and let V denote some vertex of G corresponding to a gate in that network. Let $\theta_1,\theta_2,\ldots,\theta_n$ be the edges of G which are incident into V and $\psi_1,\psi_2,\ldots,\psi_m$ be the edges incident out from V.

Define the function H for the edges ψ_j by

$$H_{\psi_j} = \prod_{i=1}^{n} (1 + H_{\theta_i}) + 1 \ , \ j = 1,2,\ldots,m$$

Furthermore, let $H_\alpha = 1$ for all edges α which are incident out from an input vertex of G.

The total number of classes into which the set $\mathfrak{F}$ for the network is partitioned by the relation of J-structural equivalence is

$$n_{\mathfrak{F}} \leq H_\phi + 2$$

where ϕ is the edge of G which is incident into the output vertex of G.

If there is exactly one path through G from each input vertex to the output vertex (i.e. if the network contains no reconverging fanout paths), then

$$n_{\mathfrak{F}} = H_\phi + 2$$

Proof:

Consider first the logical model G of a network containing exactly one path from each input vertex to the output vertex.

Suppose that there are H_{θ_i} different possible subgraphs which can drive edge θ_i in theipresence of various faults. Then, if no faults afflict any of the θ_i themselves, there are $H_{\theta_1} H_{\theta_2} \ldots H_{\theta_n}$ different subgraphs which can drive the edges ψ_j. Next suppose, without loss of generality, that V is a A-vertex. If exactly one of the edges θ_i, say θ_p is stuck at 1, then there will be $H_{\theta_1} H_{\theta_2} \ldots H_{\theta_{P-1}} H_{\theta_{p+1}} \ldots H_{\theta_n}$ subgraphs which can drive the ψ_j. If θ_p and θ_q are both stuck at 1, there will be $H_{\theta_1} H_{\theta_2} \ldots H_{\theta_{P-1}} H_{\theta_{p+1}} \ldots H_{\theta_{q-1}} H_{\theta_{q+1}} \ldots H_{\theta_n}$ different subgraphs which can drive the ψ_j, and so forth. If all of the θ_i are stuck at 1, then there is only one subgraph which can drive the ψ_j, viz. the subgraph containing only V, which becomes a fault bias vertex of value 1. Lastly, if any one or more θ_i are stuck at 0, there is only one subgraph which can drive the ψ_j, viz. the vertex V, which becomes a fault bias vertex of value 0. Thus the total number of different subgraphs, including all the above cases, which can drive the ψ_J is given by

$$
\begin{aligned}
H_{\psi_j} = {} & H_{\theta_1} H_{\theta_2} \ldots H_{\theta_n} \\
& + H_{\theta_2} H_{\theta_3} \ldots H_{\theta_n} \\
& + H_{\theta_1} H_{\theta_3} \ldots H_{\theta_n} \\
& + H_{\theta_1} H_{\theta_2} \ldots H_{\theta_{n-1}} \\
& + H_{\theta_3} H_{\theta_4} \ldots H_{\theta_n} \\
& + H_{\theta_2} H_{\theta_4} \ldots H_{\theta_n} \\
& + H_{\theta_1} H_{\theta_2} \ldots H_{\theta_{n-2}}
\end{aligned}
$$

$$+ H_{\theta_1}$$

$$+ H_{\theta_2}$$

$$+ H_{\theta_n}$$

$$+ 1$$

$$+ 1$$

$$H_{\psi_j} = \prod_{i=1}^{n} (1 + H_{\theta_i}) + 1$$

Since H_θ in this case represents the total number of different subgraphs which can drive edge θ, it is clear that, if α is incident out from an input vertex, $H_a = 1$. The one subgraph which can drive α is just the input vertex itself.

Now if ϕ is the edge which is incident into the output vertex of G, there are $H_\phi + 2$ different primary logical models of G which can result from the effects of faults afflicting the network. The additional two primary logical models are the graphs $\mathcal{G}_{\phi/0}[G]$ and $\mathcal{G}_{\phi/1}[G]$. Thus it is confirmed that $n_{\mathcal{G}} = H_\phi + 2$ for networks containing only one path from each input to the output.

Suppose, on the other hand, that there exists more than one path through G from one or more input vertices to the output vertex (i.e. that the network contains reconverging fanout paths), and that we are attempting to compute the number of different subgraphs which can drive the edges ψ_j which are incident out from vertex V. Suppose further, for example, that we are trying to determine specifically the number of subgraphs which can drive the ψ_j when none of θ_i edges incident into V is afflicted by a fault. Since the network under consideration now contains reconverging fanout paths, it is possible that some of the subgraphs which can drive, say, edge θ_P will contain edges in

common with some of the subgraphs which can drive another edge θ_q in some of the primary logical models of the network in the presence of various faults. Now the product $H_{\theta_1} H_{\theta_2} \cdots H_{\theta_n}$ attempts to count all possible subgraphs which can drive the ψ_j without regard to this possible sharing of edges between the subgraphs which can drive θ_p and those which can drive θ_q. Thus the count $H_{\theta_1} H_{\theta_2} \cdots H_{\theta_n}$ may well include a combination of some subgraph which can drive θ_p and which results from the fault condition $\xi/0$, and some other subgraph which can drive θ_q and which results from the condition $\xi/1$. Of course there is no fault in $\mathcal{F}$ containing both $\xi/0$ and $\xi/1$ as components. Thus the count $H_{\theta_1} H_{\theta_2} \cdots H_{\theta_n}$ may in fact include subgraphs which cannot in reality occur under applications of the $\mathcal{A}$-transformation for members of the set $\mathcal{F}$. This establishes that we may have $n_{\mathcal{A}} < H_{\phi} + 2$ for networks containing reconverging fanout paths.

Lastly, we observe that the H function as defined counts all possible subgraphs which may drive an edge in the primary logical models of a network in the presence of various members of the set $\mathcal{F}$ -- whether or not these subgraphs can in fact result under applications of the A-transformation. Thus under no circumstances can we have

$$n_{\mathcal{A}} > H_{\phi} + 2 .$$

An example of the application of Theorem 7.1 will be given shortly, but we first should point out that we may use the same approach with slight modification to establish an upper bound on the number of classes into which the elements of the set $\mathcal{F}$ for a network are partitioned under the relation of n-structural equivalence,

Theorem 7.2:

Let G be the logical model of a network constructed exclusively of AND, OR, NAND, NOR, and NOT gates and let V denote some vertex of G corresponding to a gate in that network. Let

$\theta_1, \theta_2, \ldots, \theta_n$ be the edges of G which are incident into V and $\psi_1, \psi_2, \ldots, \psi_m$ be the edges incident out from V .

Let the function M for the edges ψ_j be given by

$$M_{\psi_j} = \prod_{i=1}^{n} (1 + M_{\theta_i}) - 1, \quad j = 1,2,\ldots,m$$

Furthermore, let $M_\alpha = 1$ for all edges α which are incident out from an input vertex of G .

The total number of classes into which the set $\mathfrak{F}$ for the network is partitioned by the relation of &structural equivalence is

$$n_{\mathfrak{R}} \leq M_{\phi} + 2$$

where ϕ is the edge of G which is incident into the output vertex of G .

If there is exactly one path through G from each input vertex to the output vertex, then

$$n_{\mathfrak{R}} = M_{\phi} + 2$$

Proof:

At any vertex V , the number of subgraphs which can drive the ψ_j in the reduced logical models of the network in the presence of various faults is two less than the number of sub-graphs which can drive the ψ_j in the primary logical models of the network in the presence of these faults. This, of course, is because fault bias vertices do not appear in reduced logical models. Thus the additional term in the expression for the M_{ψ_j} in terms of the M_{θ_i} is - 1 instead of +1.

In all other respects, the proof of this theorem is identical to that of Theorem 7.1.

Example 7.1:

In order to illustrate the ease with which upper' bounds on (or, in this case, exact counts of) the number of classes into which the

faults which can occur in a network are partitioned by the structural equivalence relations can be established using Theorems 7.1 and 7.2, let us look once more at the structure of Fig. 7.1.

According to Theorem 7.1, we have for this structure

$$H_a = H_b = H_c = H_d = 1$$

since edges a, b, c, and d are all incident out from input vertices. We next can calculate

$$\begin{aligned} H_e &= (1 + H_a)(1 + H_b) + 1 \\ &= (1 + 1)(1 + 1) + 1 = 5. \end{aligned}$$

From the symmetry of the structure of the network, we then know that

$$H_f = 5$$

as well. Next we find

$$\begin{aligned} H_g &= (1 + H_e)(1 + H_f) + 1 \\ &= (1 + 5)(1 + 5) + 1 \\ &= 37 \quad , \end{aligned}$$

From which we have

$$n_{\mathcal{S}} \leq H_g + 2 = 39$$

This network, however, contains no reconverging fanout paths; hence we in fact have an exact count

$$n_{\mathcal{S}} = 39$$

This is, of course, the same result we obtained in our introduction of this counting technique.

To determine $n_{\mathcal{R}}$, we apply Theorem 7.2. Again starting with the edges incident out from input vertices, we see that

$$M_a = M_b = M_c = M_d = 1.$$

The symmetry of the network structure also simplifies calculations here and we get

$$M_e = M_f = (1 + 1)(1 + 1) - 1 = 3.$$

Next, we determine that

$$M_g = (1 + 3)(1 + 3) - 1 = 15,$$

whence we may ascertain that

$$n_{\mathcal{R}} = 17,$$

since our network contains no reconverging fanout paths.

In order to establish bounds on the number of classes into which the set $\mathcal{F}$ for a network is partitioned by the relation of functional equivalence we shall employ SPOOF's.

We begin by establishing a restriction on the possible fault functions, $\mathcal{Z}_{F_i}[G]$, which the network whose logical model is G can realize in the presence of the various faults $F_i \in \mathcal{F}$.

Lemma 7.1:

Let G be the logical model of a network structure and let z_D (z_C) be the disjunctive (conjunctive) SPOOF of that structure. Then there may exist a fault $F \in \mathcal{F}$ for that network such that $\mathcal{Z}_F[G] = \hat{z}$ only if

(1) Both the following conditions hold:

(a) $\hat{z}$ equals some expression which can be obtained by: (i) deleting some one or more literals and/or terms from z_D (deleting all the literals of a term is to be considered equivalent to deleting that term from the expression); (ii) deleting all subscripts from the

expression obtained in (i); (iii) simplifying the expression from (ii) using all the properties of Boolean algebra.

and

(b) $\hat{z}$ equals some expression which can be obtained by: (i) deleting some one or more literals and/or factors from z_C (deleting all the literals of a factor is to be considered equivalent to deleting that factor from the expression); (ii) deleting all subscripts from the expression obtained in (i); (iii) simplifying the expression from (ii) using all the properties of Boolean algebra.

or

(2) $\hat{z} = 0$ or $\hat{z} = 1$

Proof:

We know from Theorem 6.1 that $\mathcal{Z}_F[G]$ for any $F \in \mathcal{F}$ may be determined directly from either the disjunctive or the conjunctive SPOOF by replacing literals in the SPOOF by constants 0 or 1 in accordance with the rules of the theorem. If a literal in the disjunctive (conjunctive) SPOOF is thereby replaced by a 1 (0), the effect will be the same as that of deleting that literal from the expression. If a literal in the disjunctive (conjunctive) SPOOF is replaced by a 0 (1), the effect will be the same as that of deleting the entire term (factor) in which that literal appears from the expression.

Therefore, $\mathcal{Z}_F[G]$ for all $F \in \mathcal{F}$ can be obtained by deleting literals from either z_D or z_C and simplifying the resulting expression. There are no faults $F \in \mathcal{F}$ for which $\mathcal{Z}_F[G]$ cannot be obtained in this way.

Lemma 7.1, of course, does not imply that all functions obtainable

from a SPOOF of a network by simply deleting literals and simplifying the resulting expression can in fact occur as fault functions which that network may realize. This is because there may be no fault which can have the effect of deleting one literal without deleting some other literals as well, since it usually occurs that a symbol denoting a given edge in a network's logical model appears in more than one literal's subscript.

We may now give an upper bound on the number of functional equivalence classes of faults in a network.

Theorem 7.3:

The number of classes into which the set $\mathcal{F}$ for a network is partitioned by the relation of functional equivalence is bounded by

$$n_F \leq \min (n_{\mathcal{R}}, 2^{2^{n_X}}, 2^D + 1, 2^C + 1)$$

where n_X is the number of inputs to the network and where D (C) is the number of literals in the disjunctive (conjunctive) SPOOF of that network.

Proof:

That n_F cannot exceed $n_{\mathcal{R}}$ is a direct consequence of the fact that R-structural equivalence implies functional equivalence.

That $n_F \leq 2^{2^{n_X}}$ follows directly from the fact that there are only $2^{2^{n_X}}$ distinct functions of n_X variables which <u>any</u> network can possibly realize.

The number of possible distinct fault functions and hence the number of functional equivalence classes must also be bounded the minimum of (1) the number of distinct ways in which literals can be deleted from the disjunctive SPOOF of the network, and (2) the number of distinct ways in which literals can be deleted from the conjunctive SPOOF, since, by Lemma 7.1, all possible non-trivial

fault functions must be obtainable by deleting literals from either SPOOF and simplifying the resulting expression.

Now suppose the disjunctive (conjunctive) SPOOF of the network in question contains n literals. It may be possible to obtain non-trivial functions by deleting none, 1,2,..., up to n-l literals from the SPOOF, There is one way to delete no literals, n distinct ways to delete one literal, $n(n-1)/2$ ways todelete twoliterals, . . . , $\binom{n}{k}$ ways to delete k literals, Thus there are

$$\sum_{k=0}^{n-l} \binom{n}{k} = \sum_{k=0}^{n} \binom{n}{k} - \binom{n}{n}$$

$$= \sum_{k=0}^{n} \binom{n}{k} lk\ ln-k - 1$$

$$= (1+1)^n - 1$$

$$= 2^n - 1$$

ways to delete up to n-l literals. In addition to these possibly up to 2^n-1 ways to obtain non-trivial fault functions, it is always possible to have as fault functions the two trivial functions 0 and 1. Thus the total number of possible fault functions will be bounded by $2^n-1 + 2 = 2^n+1$. Therefore

$$n_F \leq 2^D + 1$$

and

$$n_F \leq 2^C + 1$$

and the proof of the theorem is complete,

The utility of Theorem 7.3 isillustrated in the following example.

Example 7.2:

Let us look once again at the two implementations of the two-variable exclusive-OR function shown in Fig. 2.1.

For the structure of Fig. 2.1a, the disjunctive and conjunctive SPOOF's are, as we indicated in Example 4.2, respectively,

$$z_D = x_{\bar{c}g\bar{k}}\, \bar{x}_{\bar{a}d\bar{g}k} \vee x_{\bar{c}g\bar{k}}\, \bar{y}_{\bar{b}d\bar{g}k} \vee \bar{x}_{\bar{a}e\bar{h}k}\, y_{\bar{f}h\bar{k}} \vee \bar{y}_{\bar{b}e\bar{h}k}\, y_{\bar{f}h\bar{k}}$$

and

$$z_C = (x_{\bar{c}g\bar{k}} \vee \bar{x}_{\bar{a}e\bar{h}k} \vee \bar{y}_{\bar{b}e\bar{h}k}) \wedge (x_{\bar{c}g\bar{k}} \vee y_{\bar{f}h\bar{k}}) \wedge (\bar{x}_{\bar{a}d\bar{g}k} \vee \bar{y}_{\bar{b}d\bar{g}k} \vee \bar{x}_{\bar{a}e\bar{h}k} \vee \bar{y}_{\bar{b}e\bar{h}k}) \wedge (\bar{x}_{\bar{a}d\bar{g}k} \vee \bar{y}_{\bar{b}d\bar{g}k} \vee y_{\bar{f}h\bar{k}}) \quad .$$

z_D contains 8 literals and z_C contains 12 literals. Thus we know that $n_F \leq 2^8 + 1 = 257$.

If we apply Theorem 7.2, we may calculate that

$$M_a = M_b = M_c = M_f = 1$$
$$M_d = M_e = (1+1)(1+1) - 1 = 3$$
$$M_g = M_n = (1+1)(1+3) - 1 = 7$$
$$M_k = (1+7)(1+7) - 1 = 63$$

which gives us $n_F \leq N_R \leq 65$.

For the structure of Fig. 2.1b, the SPOOF's are

$$z_D = x_{\bar{c}g\bar{k}}\, \bar{y}_{\bar{b}d\bar{g}k} \vee \bar{x}_{\bar{a}e\bar{h}k}\, y_{\bar{f}h\bar{k}}$$

and

$$z_C = (x_{\bar{c}g\bar{k}} \vee \bar{x}_{\bar{a}e\bar{h}k})(x_{\bar{c}g\bar{k}} \vee y_{\bar{f}h\bar{k}}) \wedge (\bar{y}_{\bar{b}d\bar{g}k} \vee \bar{x}_{\bar{a}e\bar{h}k})(\bar{y}_{\bar{b}d\bar{g}k} \vee y_{\bar{f}h\bar{k}})$$

z_D has 4 and z_C 8 literals; thus $n_F \leq 2^4 + 1 = 17$ for this structure.

Applying Theorem 7.2, we get in this case

$$M_a = M_b = M_c = M_f = 1$$
$$M_d = M_e = (1+1) - 1 = 1$$
$$M_g = M_h = (1+1)\cdot(1+1) - 1 = 5$$
$$M_k = (1+5)\cdot(1+5) - 1 = 35$$

whence we have

$$n_F \leq n_{\mathcal{R}} \leq 37$$

The remaining bound on n_F given by Theorem 7.3 gives us

$$n_F \leq 2^{2^2} = 16$$

for both structures since both structures have two inputs. Thus in this particular instance, the tightest bound on n_F offered by Theorem 7.3 is also one of the most obvious bounds. Except in cases involving fairly complicated two- and three-input networks, however, the $n_F \leq 2^{2^{n_X}}$ bound will seldom be of any practical use at all since 2^{2^n} increases so rapidly with increasing n . In the case of the four-input network of Fig. 7.1, for example, this bound would tell us only that $n_F \leq 2^{2^4} = 2^{16} = 65{,}536$ -- a result of little use, since the entire set $\mathcal{F}$ for that structure contains only $3^7 = 2{,}187$ faults! In Example 7.1, we determined that, for the structure of Fig. 7.1, $n_{\mathcal{R}} = 17$. Since the the structure is free from reconverging fanout paths, it follows immediately from Theorem 6.8 of Reference [1] that $n_F = n_{\mathcal{R}} = 17$.

It must be pointed out that Theorem 7.3 is intended merely to give an upper bound on n_F which can be very quickly computed and thus to

provide an easy method to assess the order of magnitude of the number of functional equivalence classes which one can expect with a given network. At the expense of somewhat more work, one may take recourse directly to Lemma 7.1 and find just how many distinct $\mathcal{F}_F[G]$ there may be for a given network under the conditions of the lemma, This technique typically gives a tighter upper bound on n_F than any of the bounds given by Theorem 7.3. In table 7.1, we list all of the 16 functions of two variables and whether each function may possibly be a fault function for the structures of Figs, 2.1a and 2.7b, respectively, as determined from Lemma 7.1. We also show a representative fault which yields the corresponding fault function in each case where such a fault actually exists. As may be seen from the table, such a representative fault does exist for every function in the table which can possibly be a fault function by the criteria of Lemma 7.1. In other words, by using the lemma, we have determined a very tight upper bound on n_F indeed. We can see that this quantity is equal to 13 and 11 for the structures of Figs. 2.1a and 2.1b, respectively. We furthermore have, in Table 7.1, the fault function corresponding to each functional equivalence class, and a representative fault from each class. As a point of interest: the representative faults tabulated were all determined from SPOOF's by the technique indicated in Example 6.2.

TABLE 7.1

A tabulation of possible fault functions and of representatives of each functional equivalence class for the network structures of Figs. 2.1a and 2.1b.

Unction	Possible as fault function for structure of Fig. 2.1a?	Representative fault in structure of Fig. 2.1a.	Possible as fault function for structure of Fig. 2.16?	Representative fault in structure of Fig. 2.16.
0	yes	k/0	yes	k/0
1	yes	k/1	yes	k/1
x	yes	a/0,f/0	yes	a/1,b/0
Y	yes	b/0,c/0	yes	a/0,b/1
x	yes	b/1,f/1	yes	b/1,f/1
$\bar{y}$	yes	a/1,c/1	yes	a/1,c/1
xy	no	none	no	none
$x \vee Y$	yes	a/0	yes	a/0
$\bar{x}y$	yes	b/1	yes	b/1
$x\bar{y}$	yes	a/1	yes	a/1
$\bar{x}\bar{y}$	no	none	no	none
$\bar{x} \vee y$	yes	b/1,c/1,e/1	no	none
$x \vee \bar{y}$	yes	a/1,d/1,f/1	no	none
$\bar{x} \vee \bar{y}$	yes	c/1	yes	e/1,f/1
$x\bar{y} \vee \bar{x}y$	yes	λ	yes	λ
$xy \vee \bar{x}\bar{y}$	no	none	no	none

8. Conclusion

In this report, we have demonstrated that knowledge of the structure of a given logic network must 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 functions.

Although the definitions and other formal results developed in this report use the concept of "inverting vertices" and therefore apply only to networks consisting of AND, OR, NAND, NOR, and NOT gates, the author is confident that, in the course of future work, these definitions and results can be reformulated to embrace networks containing gates performing any arbitrary switching function.

Useful applications of SPOOF's as presented in this report 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.

Techniques more powerful and more general than those described in Reference [1] for the enumeration of the classes into which the possible faults which can occur in a network are partitioned by fault equivalence relations are also presented in this report. These techniques permit one, for the first time, to establish upper bounds on, and, in some cases, exact counts of the number of equivalence classes of multiple as

as weii as single faults in logic networks. The SPOOF is shown to be a very useful tool for enumerating fault equivalence classes under the partition induced by the relation of functional equivalence.

The results described in the present report constitute further progress toward a goal expressed in the work of Reference [1]: the use of an algebraic approach to develop a general, formal theory of faulty digital systems which 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

[1] Clegg, F. W. and E. J. McCluskey, Algebraic Properties of Faults in Logic Networks. Digital Systems Laboratory Technical Report No. 4, Stanford Electronics Laboratories Report No. SU-SEL-69-078 (1970).

[2] Armstrong, D. B., "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets," IEEE Transactions on Electronic Computers, Vol. EC-15, No. 1, pp. 66-73 (Feb. 1966).

[3] Poage, J. F., "Derivation of Optimum Tests to Detect Faults in Combinational Circuits," Proceedings of the Symposium on the Mathematical Theory of Automats, Polytechnic Institute of Brooklyn, pp. 483-528, Polytechnic Press (Brooklyn, 1963).

[4] McCluskey, E. J., Introduction to the Theory of Switching Circuits, New York: McGraw-Hill Book Co., 1965.