FAULT SECURE MULTIPLE-VALUED LOGIC NETWORKS

James E. Smith

Dept. of Elect. and Comp. Eng.
University of Wisconsin-Madison

Jean Dussault

Western Electric Eng. Res. Ctr.
Princeton, New Jersey

Abstract

Fault secure multiple-valued logic networks have their outputs encoded in an error-detecting code so that assumed failures either result in no output error or in a detectable (noncode) output. In this paper we discuss a way of constructing fault secure combinational networks as well as a generalization that we define to be "strongly fault secure." The only constraint on the networks themselves is that they be made of positive unate gates and use unordered output encodings.

The fault assumptions are quite general; they include stuck-at faults as well as many other failures which seem reasonable in a multiple-valued technology. Protection is provided against both permanent and intermittent faults. Various unordered codes and their properties are discussed as is the construction of check circuits that are themselves fault secure or strongly fault secure.

I. Introduction

In theory, multiple-valued logic networks enjoy many advantages over two-valued networks [1]. Nevertheless, two-valued networks are used almost exclusively in practice, and a major reason is that two-valued logic networks are more reliable. To take one example, multiple-valued networks may have more critical noise tolerances than two-valued networks. In order for multiple-valued logic with its attendant advantages to be of practical use, the reliability problem must be solved.

There are two fundamental approaches to increasing the reliability of logic networks. The first is to simply increase the reliability of the individual components (e.g. gates). The second approach is applied at a higher level and involves interconnecting components so that reliability is increased due to the structure of the interconnection. A very simple example of the second approach is the use of triplication and voting [2]. Of course, these two approaches can be used separately, or they can be used in combination.

This paper presents an approach to the reliability problem that falls into the second of the above categories. We discuss a class of combinational multiple-valued logic networks that have their output signals encoded in an error-detecting code. These networks are designed so that assumed failures (faults) either result in a detectable error (i.e. a noncode output) or in no error at all. This approach is based on the philosophy that the costly errors are those that go undetected. Errors can still occur using the proposed method, but they are immediately detected so that undesirable consequences can be minimized.

It appears that intermittent faults may be of great concern in multiple-valued networks due to such things as reduced noise tolerance. An additional advantage of the networks discussed here is that they are very effective at combating intermittent faults, and it is felt that the method given is one of the few practical ways of dealing with intermittent faults.

Networks that have the error-detecting property informally described above have studied for two-valued logic and are said to be "fault secure" [3]. Formal definitions of the fault secure property and a useful generalization are given and discussed in the sequel. The theory developed here is in some ways a straightforward extension of existing theory developed for the two-valued case [3-8]. However, in the area of fault modeling we depart from the obvious extension, and it is this departure that leads to what we feel are intersting and practical results.

II. Preliminaries

A. Basic Definitions

We consider multiple input, multiple output combinational logic networks which are formed as acyclic interconnections of multiple-valued gates. Let G be an m-valued logic network. Then if G has n primary input lines, the m^n vectors of length n form the input space, X, of G. The output space, Y, is similarly defined to be the set of m^p vectors of length p where G has p primary outputs. During normal, i.e. failure-free, operation G receives only a subset of X called the input code space, A, and produces a subset of Y called the output code

space, B. Members of a code space are called code words. Under faults, noncode words may be produced.

B. Post Algebras

The logic networks to be studied here are based on Post algebras [16,17]. For an m-valued system we use Post chains formed from the set of integers [0,1,2, ..., m-1] and the natural partial ordering $\geq$. A function or order n (i.e. n variables) possesses m^n entries in its truth table so that there exist m^{m^n} truth tables (and functions) or order n.

Post showed that the following two operators form a functionally complete set:

1) $f+g \triangleq \max(f,g)$

2) $p^{\rightarrow} \triangleq (p+1) \bmod m$.

Another important operator, not needed for functional completeness, but equally simple to generate and helpful in constructing functions is the minimum operation:

3) $f \cdot g \triangleq \min(f,g)$.

With the min and max operators being the greatest lower bound and least upper bound, respectively, the lattice of truth values is a chain lattice, an example of which is shown in Figure 1a. Figures 1b and 1c show the lattices pxp and pxpxp and indicate a partial ordering among m-valued vectors or vertices.

Although forming a complete set of operators with max and min functions, the cycling operation is not the only unary function of interest. For an m-valued system, there are m^m such functions. As an example, consider the 27 possible ternary functions of a single variable:

X	$\vec{X}$
0	000000000111111111222222222
1	000111222000111222000111222
2	012012012012012012012012012
	*** ** * ** * *

Unary functions will be denoted as $X^{(a_0,a_1,\ldots a_{m-1})}$ and will take value a_i if X=i, $0 \leq i \leq m-1$. For example $X^{\rightarrow} = X^{(120)}$ in ternary algebra. For constructing logic networks it is desirable to limit the number of unary operators for the usual reasons associated with modularity. Due to other considerations, however, it may not always be possible to use a minimum number.

The partial ordering relation indicated in Figure 1 can be reformulated into the concept of covering.

Definition 1: A vertex $x_1x_2\ldots x_n$ covers a vertex $y_1y_2\ldots y_n$, denoted as $x_1x_2\ldots x_n \geq y_1y_2\ldots y_n$ if $x_i \geq y_i$ for all i.

Using the max and min operators one can say $x_1x_2\ldots x_n \geq y_1y_2\ldots y$ if $\max(x_i,y_i)=x_i$ for all i, and $x_1x_2\ldots x_n \geq y_1y_2\ldots y_n$ iff $\min(x_i,y_i)=y_i$ for all i.

If two vertices do not satisfy in any way the binary ordering relation, they are said to be incomparable. Functions that have outputs that are ordered in essentially the same way as their arguments are called linear monotonic or unate.

Definition 2: A function $F(x_1x_2\ldots x_n)$ is positive unate if $x_1x_2\ldots x_n \geq y_1y_2\ldots y_n$ implies $F(x_1x_2\ldots x_n) \geq F(y_1y_2\ldots y_n)$.

This notion can be readily extended to multiple output functions by requiring that each output function be positive unate.

In Section 3 we construct combinational logic networks from positive unate gates. Any such network (typically with multiple outputs) must also be positive unate.

Theorem 1: Any combinational logic network composed solely of positive unate gates must realize a positive unate function.

For generality, we do not insist on a particular set of positive unate gates, however, one might be interested in sets of gates that can implement all positive unate functions.

Definition 3: A set of positive unate operators $\{U_1,U_2,\ldots U_s\}$ is positive unate complete, or for our purposes simply unate complete, if the operators can be composed to form all positive unate functions. The use of logical constants is allowed.

Since the cycling, min, and max operators are functionally complete they can be used to construct all positive unate functions. However, the cycling operator is not positive unate, so this set of operators is not unate complete. It follows that other unary operators must be used with the max and min operators in order to get a unate complete set. In the table of single variable ternary functions, the positive unate functions are marked with an asterisk (*). Some positive unate unary functions can be obtained from others, for example $\max(X^{002},X^{012}) = X^{(012)}$, and therefore one of the arguments is not essential. We do not address the problem of finding the smallest unate complete set(s). Nevertheless, it should be apparent that the min, max, and all positive unate unary operators are sufficient to implement all unate functions, and hence form a unate complete set of operators.

The partial ordering relation $\geq$ can be used to define a class of codes with which we are interested.

Definition 4: An unordered code is a set of mutually incomparable vertices of similar degree.

The code {002,011,101,020,110,200} is an example of an unordered code. Specific unordered codes are discussed in section IV.

C. Fault Modeling

All failures that can occur in a network G are assumed to be modeled logically as faults. A fault is some transformation in the logic function realized by the network. By looking at the logical effects of a hardware failure, one can deal with failures within the structure of Post algebras.

For two-valued logic networks, the most common fault assumption is that failures can be modeled logically as lines "stuck-at" logical values (either 0 or 1) [9]. This stuck-at model has led to many useful results in two-valued logic. Nevertheless, we feel that a stuck-at model is inadequate for multiple-valued networks. This is primarily because a good fault model is technology dependent, and it is not all clear that the stuck-at model will model the most likely failures in a multiple-valued technology. For example, due to signal degradation in a gate (a reasonable failure mode) output voltage levels could become shifted downward. Consequently, outputs may have incorrect logic values (they are consistently too low), but are clearly not stuck. Faults of the type just described, as well as those where an upward voltage shift occurs, will be referred to as "skew faults".

In this paper we use two very general fault models. For a given network G, we define F_s to be a fault set that includes the transformation of any single gate in G so that it realizes any other single-output function. F_u is defined to be the set of single gate transformations to any positive unate function. Both F_s and F_u are very rich fault sets. F_s clearly includes all stuck-at and skew faults, and when all the gates implement positive unate functions as they do in the sequel then F_u also includes stuck-at and skew faults. These two fault models include many other failures with F_s being nearly the most general fault model possible, given that only single gates fail.

For later convenience, we define F_m to be the set of multiple gate transformations in G. Hence, $F_s \subseteq F_m$ and $F_u \subseteq F_m$. To denote a member of F_m involving two (or more) members of F_s, say f_1 and f_2, we write the fault as $f_1 \cup f_2$.

The number of faults in a digital system is typically modeled as a Poisson process [10]. As a consequence, it is assumed that fault in F_s or F_u accumulate G one at a time with some time interval separating them. A critical assumption regarding this time interval is discussed in the next section. Because faults accumulate with time, we denote this behavior by defining a fault sequence $\langle f_1, f_2, \ldots f_n \rangle$ to represent the event where f_1 occurs, followed latter by f_2, and so forth until f_n occurs.

III. Fault Secure Network Structures

A. Definitions

We now formally define fault secure combinational networks and an interesting generalization, strongly-fault secure networks. In order to make concise definitions possible, we denote the output of the network G under input $x \in X$ and fault $f \in F_M$ as $G(x,f)$. Under the fault-free condition, the output is denoted as $G(x,\emptyset)$.

The following definition is due to D. A. Anderson [3]. It refers to the functional block G with input code space $A \subseteq X$, output code space $B \subseteq Y$, and with some assumed fault set F.

Definition 5: G is fault secure (FS) with respect to F if

$$\forall f \in F \; \forall a \in A \quad G(a,f) = G(a,\emptyset) \text{ or } G(a,f) \notin B.$$

If G is FS and an assumed fault occurs, then any erroneous output can be detected by simply checking for code or noncode network outputs. Hence, a fault secure network appears to offer complete protection against undetected errors caused by modeled failures. Unfortunately, this is not necessarily true. As time passes, the network may pick up a second fault, f_2, in addition to the first, f_1. If $f_1 \cup f_2 \notin F$ then some code input may cause the output to be an undetected incorrect code word.

A partial solution to this problem is to repair the network when the first noncode output appears. If f_1 can cause a noncode output and a sufficient time passes between f_2 and f_1 for a subset of A to be applied that tests for f_1, then this solution is satisfactory. A further problem, however, is that f_1 may be such that $\forall a \in A \; G(a,f_1) = G(a,\emptyset)$, i.e. there is no input code word that "tests" for the presence of f_1.

Despite the above complications a FS network is still useful from a practical standpoint if it is periodically tested offline (possibly using noncode inputs). Using this technique, an undetected error can occur only if between two off-line tests.

1) a fault occurs that is not tested by a code input,
2) a second fault occurs,
3) the second fault conspires with the first to give an erroneous code output.

We believe that in most situations, the probability of all three of these events occurring during a suitably chosen testing interval would be very small, and a FS network would be

effective. It would almost certainly be more reliable than an unprotected system using the same testing interval.

To develop another solution to the above problem, we first observe that G may be FS with respect to $f_1 \cup f_2$ even though $f_1 \cup f_2 \notin F$. Then, if $f_1 \cup f_2$ results in a noncode output for some code input, the faults are detected and the network can be repaired. The following definitions (from [5] formalize this property for fault sequences of arbitrary length.

Definition 6: For a fault sequence $<f_1, f_2, \ldots f_n>$, $f_i \in F$, let k be the smallest integer for which there exists a code input $a \in A$ such that $G(a, {}^{k} f_j) = G(a, \emptyset)$. If there is no such k, set k=n. Then G is strongly fault secure (SFS) with respect to the fault sequence if

$$\forall a \in A \text{ either } G(a, \overset{k}{\underset{j=1}{\cup}} f_j) = G(a, \emptyset) \text{ or}$$

$$G(a, \overset{k}{\underset{j=1}{\cup}} f_j) \notin B.$$

Definition 7: The network G is strongly fault secure (SFS) with respect to the fault set F if G is SFS with respect to all fault sequences whose members belong to F.

It can be shown that if a sufficient time elapses between faults so that the complete set of code inputs is applied, then the first erroneous output in response to any assumed fault (s) in a SFS network must be a noncode word. Consequently, if the above conditions are met and repairs are made whenever a fault is detected, then no undetected errors will be emitted from the network. In addition, no periodic offline testing is required; however, in many instances offline testing may be advisable to ensure that a sufficient test set (code inputs) has been applied between faults.

Two-valued SFS logic networks where k is 1 for all fault sequences have been called "totally self-checking" and have been studies extensively [3,6-8,11-14]. The typical (equivalent) definition given for these networks is:

Definition 8: A network G is totally self-checking with respect to F if

1) it is fault secure with respect to F and
2) $\forall f \in F \; \exists a \in A$ such that $G(a,f) \notin B$(the self-testing property [3]).

We have chosen not to emphasize totally self-checking networks because we believe that our fault models are so general (particularly F_s) that they make the construction of totally self-checking networks very difficult. There are two arguments to support this contention.

First, in order to self-test all members of F_s it is necessary that all possible input combinations be applied to all gates by members of A, and for each combination a path must be sensitized [15] to at least one output. The totally self-checking networks studied elsewhere have been rather difficult to construct, and they are based on stuck-at fault assumptions which only require some proper subset of gate input combinations be applied to each gate with the gate's output being sensitized to network outputs.

Second, the most straightforward way of constructing totally self-checking networks involves transforming a network with a non-self-tested stuck-at fault into one realizing the same function without undetectable faults. This method involves the removal of certain "redundant" parts of the network [14], i.e. the untested lines and some associated gates. For the fault models we propose here, no such transformation is apparent.

B. Main Theorems

In this section we show that any network made of positive unate gates which uses an unordered output code space must be FS with respect to F_s and SFS with respect to F_u. Consequently, a very general design method for these types of networks is to

1) choose some unordered output encoding;
2) implement the desired function using network of positive unate gates.

Using this method, the designer has a great deal of freedom both when choosing codes and designing a network. Further, later in this section we will discuss conditions under which the restriction to positive unate gates can be relaxed.

Theorem 2: Any combinational network G composed only of positive unate gates and using an unordered output code space is FS with respect to F_s.

Proof: Without loss of generality, say the fault $f \in F_s$ is present in G, and it affects gate g (refer to Fig. 2a). Cut the output line of g and treat it as a network input line. Then we have a new network G' with inputs $x_1, x_2, \ldots, x_p$, g which must realize a positive unate function since it contains only positive unate gates. Apply any input $a \in A$ to G, and let $g(a,\emptyset)$ be the normal output of gate g (no fault is present). Let g(a,f) be the output of g when it is faulty. Then, $g(a,\emptyset) \geq g(a,f)$ or $g(a,f) \geq g(a,\emptyset)$ (the ordering of elements of the algebra is total. If $a \circ g(a,\emptyset)$ represents the concatenation of the input word a with $g(a,\emptyset)$, then $a \circ g(a,\emptyset) \geq a \circ g(a,f)$ or $a \circ g(a \circ g(a,f) \geq a \circ g(a,\emptyset)$. By considering the structure of G' shown in Fig. 2b, we can deduce that

(1) $G'(a \circ g(a,\emptyset),\emptyset) = G(a,\emptyset)$, and
(2) $G'(a \circ g(a,f),\emptyset) = G(a,f)$.

Since G' is positive unate,

$G'(a \circ g(a,f),\emptyset),\emptyset) \geq G'(a \circ g(a,\emptyset),\emptyset)$

or $G'(a \circ g(a,\emptyset),\emptyset) \geq G'(a \circ g(a,f),\emptyset)$.

Substituting from (1) and (2);

$G(a,f) \geq G(a,\emptyset)$

or $G(a,\emptyset) \geq G(a,f)$.

Consequently, either $G(a,f)=G(a,\emptyset)$ or $G(a,f) \notin B$ since all members of B are unordered. Q.E.D.

Theorem 3: Any combinational network G composed only of positive unate gates and using an unordered output code space is SFS with respect to F_u.

Proof: For an arbitrary fault sequence $\langle f_1, f_2, \ldots f_n \rangle$, $f_i \in F_u$, let k be the smallest integer for which $\exists a \in A$ such that $G(a, \bigcup_{j=1}^{k} f_j) = G(a,\emptyset)$. If there is no such k, then it follows that G is SFS with respect to the sequence by Definition 6. If there is such a k, then $G(a, \bigcup_{j=1}^{k-1} f_j) = G(a,\emptyset)$ for all code inputs. Let G' denote the network G with the transformations of gates indicated by $\bigcup_{j=1}^{k-1} f_j$. Then G' is effectively made of positive unate gates and $\forall a \in A\ G'(a,\emptyset) = G(a, \bigcup_{j=1}^{k-1} f_j)$. By Theorem 2 G' is FS with respect to F_s and $F_u \subseteq F_s$; consequently, $\forall a \in A\ G'(a,f_k) = G(a, \bigcup_{j=1}^{k} f_j) = G(a,\emptyset)$ or $G'(a,f_k) \notin B$. Then G is SFS for the fault sequence. Since we chose an arbitrary fault sequence, the above argument must hold for all sequences. Q.E.D.

C. Implementations with Nonunate Gates

Thus far we have discussed networks made of positive-unate gates, and this has been the only restriction on network structure. We have not insisted on any particular set of gates; any unate-complete set will do. As a consequence, one could take any single-output subnetwork and call it a "gate". Since it is made of positive unate gates, it will also be positive unate. As long as its function is positive unate, any internal realization of the "gate" can be used. Hence, some nonunate gates can be interconnected to form a single-output positive unate "gate" in one of our networks.

Conceptually, the network satisfies Theorem 2 and is FS with respect to F_s, the set of all transformations of positive unate gates. It can also be seen that F_s includes all transformations of all the nonunate component gates as well. When considering F_u as in Theorem 3, some restrictions must be made on faults occurring in the nonunate component gates. In most cases, however, we feel that an adequate fault model results.

D. Interconnections of Networks

In order to be of practical use, it must be possible to interconnect FS ans SFS networks in order to form larger FS and SFS networks. As before, we consider combinational networks.

In an interconnection of networks, the output code space of one network may be the input code space of other network(s). Consequently, there must not be a conflict between the use of unordered input code spaces and the restriction that networks be made of positive unate gates. Fortunately, as the following Lemma points out, unordered input codes and positive unate gates fit together quite well.

Lemma 1: Any multiple-output function whose inputs are unordered can be embedded in a positive unate function if don't care inputs are properly assigned.

The proof of the Lemma is very straightforward and has been omitted.

Another consideration when interconnecting the proposed networks is that when placed in a particular system, not all members of A may ever reach the inputs of a network. The following Lemma shows that this causes no problems.

Lemma 2: Let G be a network made of positive unate gates with an unordered output code space B and with input code space A. Then the same network G with a new input code space $A' \subset A$ is FS with respect to F_u.

Proof: Since $A' \subset A$, B' is the new output code space, and $B' \subseteq B$. Therefore, B' must be unordered, and the Lemma follows from Theorems 2 and 3.

We close this section with a theorem showing that essentially no difficulties are encountered when interconnecting the networks we propose.

Theorem 4: Given a set of networks $N = \{N_1, N_2, \ldots\}$ that are each FS (SFS) with respect to F_s (F_u) according to Theorem 2 (3), any acyclic interconnections of the networks in N forms a network that is also FS (SFS) with respect to F_s (F_u).

E. Intermittent Faults

An intermittent fault is one that does not remain permanently in a network after it first appears. After its first occurence, we say that an intermittent fault is either 'present' or 'nonpresent'. When a member of F_s is intermittent and it is present, the network must be FS since a present intermittent fault behaves in the same way as a permanent fault. A nonpresent intermittent fault does not affect the network at all, so it is trivially FS under this condition. Consequently, a network G as described in Theorem 2 is FS

with respect to all intermittent members of F_s as well as the permanent ones.

When considering the SFS property for intermittent faults, there is some difficulty, primarily because some members of a fault sequence could be present, and others nonpresent. In short, the SFS property does not easily extend to intermittent members of F_u.

IV. Unordered Codes

In this section, we define some specific unordered codes and examine some of their properties.

A. Fixed-Weight Codes

Consider a vertex $x_1x_2\ldots x_n$, $0\leq x_i\leq m-1$. Then the weight of the vertex is defined to be $\sum_{i=1}^{n} x_i$. Then, as their name indicates, fixed-weight codes are made up of vertices of identical weights. In pxp and pxpxp of Fig. 1b and c, each horizontal row forms a fixed-weight code space. For example, the code {002,011,101,020,110,200} is a fixed-weight code where the weight is 2. Larger fixed-weight codes can be obtained by performing weight preserving concatenation products of smaller lattices. For the binary case, fixed-weight codes are often called k-out-of-n codes where k is the number of 1's.

In an m-valued systems, the maximum weight that a length n code word can have is $(m-1)\cdot n$. We define the fixed-weight code of weight $\left\lfloor \frac{(m-1)\cdot n}{2} \right\rfloor$ to be the half-weight code. The binary analog is the $\lfloor \frac{n}{2} \rfloor$-out-of-n code.

Theorem 5: In an m-valued system, a length n unordered code with the maximum number of code words is the half-weight code.

The proof of Theorem 5 is a generalization of the one used to show that $\lfloor \frac{n}{2} \rfloor$-out-of-n codes are optimal binary unordered codes [18]. As a consequence of the theorem, if one wants to minimize the number of digits used to encode information the half-weight codes will lead to such optimal encodings.

A special case of the fixed-weight codes are the "two-rail codes." The most basic two-rail code word has length 2, and these two digits, ab, have the property that $b=a^{(m-1,m-2,\ldots,1,0)}$; i.e. b is the negation of a. Its code words all have weight m-1. Larger two-railed code spaces can be obtained by simple concatenation of the basic two-rail codes. For example, {0202,0211,0220,1102,1111,1120,2002, 2011,2020} is the 3-valued length 4 two-rail code.

B. Berger Codes

Berger codes [19] were first constructed for binary asymmetric channels; the following is a multiple-valued generalization. A Berger code in our generalized sense is a separable code consisting of separate information digits to which are concatenated check digits. Separable codes are useful when the extraction of encoded information is to be done most efficiently.

To satisfy the unordered property the weight of the check digits should decrease as the weight of the information digits increases. There are many ways of doing this, we consider only one useful case. A Berger code is a code consisting of information digits to which are appended the value in base m-1 of the weight of the m-1's complements of the information digits. Berger codes can be characterized by three numbers: The number of truth values, m; the number of information digits, n; and the number of check digits, k. Clearly, $k=\lceil \log_m(n.(m-1)+1) \rceil$. If $\lceil \log_m(n.(m-1)+1) \rceil = \log_m(n.(m-1)+1)$ then the code is maximal.

Example: A maximal Berger code with m=3, n=4, and k=2.

Info. digits	2's comp.	Weight_{10}	Weight_3	Codeword
0000	2222	8	22	000022
0001	2221	7	21	000121
0002	2220	6	20	000220
0010	2212	7	21	001021
:	:	:	:	:
1021	1201	4	11	102111
1022	1200	3	10	102210
1100	1122	6	20	110020
:	:	:	:	:
2220	0002	2	02	222002
2221	0001	1	01	222101
2222	0000	0	00	222200

Theorem 6: In an m-valued system, a length n separable unordered code with the maximum number of code words is the Berger code.

As with Theorem 5, a proof is a straightforward generalization of the one used for the binary case [19].

V. Check Circuits

In the networks proposed here, faults are detected by checking the network outputs for noncode words. Consequently, one must be able to construct check circuits to perform this code checking. A complication is that any such check circuit is presumably as prone to failure as the circuits being checked.

In this section, we consider checkers that are themselves fault secure or strongly fault secure. Checkers are functional blocks that in addition to satisfying the previously defined properties also possess the code disjoint property.

Definition 9: A functional block is code disjoint if it maps all noncode inputs to noncode outputs.

A checker may have a code space input consisting of a product of several codes spaces, but generally has a single and small output code space.

Fault secure check circuits are for the most part easy to construct. One can simply use a single output line with one output value indicating that the checker is examining a code word, and with the other values indicating a noncode input is being applied. That such a checker is fault secure with respect to F_s (or actually any member of F_m) can be easily shown. One should note, however, that after a fault occurs in the checker, it may no longer be code disjoint.

Strongly fault secure checkers, on the other hand, are much more interesting. This is because we will insist that they retain the code disjoint property for all initial fault sequences which result in no detectable output errors (referring to Definition 6, the sequences of length 1,2,... k-1).

Theorem 7: In order for a checker to be SFS with respect to F_u, at least two output lines are necessary.

Proof: If only one output line is used and the line becomes stuck at a "valid" indication (this fault is in F_u) then the check circuit will not produce a noncode output for this length 1 fault sequence. However, the circuit is no longer code disjoint. Consequently, more than one output line is required. Q.E.D.

A consequence of this theorem is to define the size of the hardcore, i.e. the smallest number of output lines a checker may have and still possess the code disjoint and SFS properties. Two output lines are usually sufficient, and two lines are used in the next sections where checkers for specific codes are discussed.

A. SFS Checkers for Fixed-Weight Codes

From the general discussion on checkers it is apparent that the code disjoint property imposes a partition on the set of input vertices. More precisely the set of input code words will map onto a set of output codewords. Due to the positive unateness of the checker, a noncode input to the checker that covers a codeword should produce at the checker's outputs a noncode word that covers some output codeword. A dual of this statement is also true if the covering is the other way around.

The theory and implementation of checkers for binary fixed-weight codes is well developed [4, 7, 12]. Some of the work deals with the theory of Ramsey numbers from combinatorics [7], and its extension to the multivalued case is very difficult. As an alternative, we shall present a method to construct checkers based on the work of Carter and Schneider [4] and Anderson [12].

The checker will realize two functions; more precisely codewords will be mapped onto 2 output codewords (0,m-1) and (m-1,0). To design a checker for a code of fixed-weight k the input digits to the checker are partitioned into two non-empty groups A and B of n_a and n_b digits. The weight of the digits in group A(B) will be labeled w_a (w_b). Define the function $T(w_a \geq i)$ to have value m-1 if the argument is true and 0 otherwise. Let the 2 output functions be defined as follow:

$$f = \max_{i=0}^{\alpha} T(w_a \geq i) \cdot T(w_b \geq k-i) \qquad i \text{ even}$$

$$g = \max_{i=0}^{\alpha} T(w_a \geq i) \cdot T(w_b \geq k-i) \qquad i \text{ odd}$$

Where the bound α can be obtained as follow:

$$T(w_a \geq i) = 0 \qquad \text{for } i > w_a$$

$$\text{and } T(w_b \geq k-i) = 0 \qquad \text{for } i \geq k-w_b$$

hence $\alpha = \min(w_a, k-w_b)$.

To complete the design method we need a technique for implementing the threshold functions. $T(w_a \geq i)$ is to be implemented. There are n_a digits in that group labeled (say) $x_1 x_2 \ldots x_{n_a}$. First enumerate all the n_a-tuples that have weight w_a. For each variable x implement the functions $x^{(0,m-1,\ldots,m-1)}$ $x^{(0,0,m-1,\ldots m-1)}$, $\ldots x^{(0,0,\ldots 0,m-1)}$ and use them as follows. For each n_a-tuple first described realize the product $x_1^{(B_{11},B_{12},\ldots,B_{1_m})}$ $x_2^{(B_{21},B_{22}\ldots B_{2_m})}$, $\ldots x_{n_a}^{(B_{n_a 1},B_{n_a 2},\ldots B_{n_a m})}$ where

$B_{ij} = 0$ if $j < x_i$

$B_{ij} = m-1$ if $j \geq x_i$

The function $T(w_a \geq i)$ is the maximum of all the product terms thus obtained. Example: A checker for a code of length 3, weight 3 in a 3-valued system. The code is {012,102,021,111, 201,120,210}. Let A be the first 2 digits x_1 and x_2 and B be x_3.

A	w_a	$x_1^{(B_{11},B_{12},B_{13})} \cdot x_2^{(B_{21},B_{22},B_{23})}$
01	1	$x_1^{(222)} \cdot x_2^{(022)=2} \cdot x_2^{(022)=x_2^{(022)}}$
10	1	$x_1^{(022)}$
11	2	$x_1^{(022)} \cdot x_2^{(022)}$
02	2	$x_2^{(002)}$
20	2	$x_1^{(002)}$
12	3	$x_1^{(022)} \cdot x_2^{(002)}$
21	3	$x_1^{(002)} \cdot x_2^{(022)}$

B	w_b	$x_3^{(B_{31},B_{32},B_{33})}$
0	0	$x_3^{(222)=2}$
1	1	$x_3^{(022)}$
2	2	$x_3^{(002)}$

$f = T(w_a \geq 0) \cdot T(w_b \geq 3) + T(w_a \geq 2) \cdot T(w_b \geq 1)$

$= [x_1^{(022)} \cdot x_2^{(022)} + x_1^{(002)} + x_2^{(002)} + x_1^{(002)} \cdot x_2^{(022)} + x_1^{(022)} \cdot x_2^{(002)}] \cdot x_3^{(022)}$

$g = T(w_a \geq 1) \cdot T(w_b \geq 2) + T(w_a \geq 3) T(w_b \geq 0)$

$= (x_1^{(022)} + x_2^{(022)}) \cdot x_3^{(002)} + (x_1^{(002)} \cdot x_2^{(022)} + x_1^{(022)} x_2^{(002)}) \cdot 2$

The absorbtion laws hold so $x^{(002)} + x^{(002)} y^{(022)} = x^{(002)}$ and $2 \cdot x = x$, hence

$f = (x_1^{(002)} + x_2^{(002)} + x_1^{(022)} \cdot x_2^{(022)}) \cdot x_3^{(022)}$

and

$g = (x_1^{(022)} + x_2^{(022)}) \cdot x_3^{(002)} + x_1^{(002)} \cdot x_2^{(022)} + x_1^{(022)} \cdot x_2^{(002)}$

This can be implemented using 6 unary gates, 5 MAX gates and 5 MIN gates.

B. SFS Checkers for Two-rail Codes

Checkers for two-rail codes have for an input code space a product of basic two-rail codes and produce a single two-rail coded output. Let k be the degree of the product of two-rail input code spaces. Let a_k and b_k be the k^{th} two-rail input. The output functions f_k and g_k can be expressed recursively as follows:

$$f_k = f_{k-1} \cdot b_k + g_{k-1} \cdot a_k$$

$$g_k = f_{k-1} \cdot a_k + g_{k-1} \cdot b_k \quad \text{where } f_1 = a_1 \text{ and } g_1 = b_1$$

We shall not prove this result. However, it should be apparent from the example below that these formulae work in general.

Example: For k=2, the functions become

$$f_2 = a_1 b_2 + b_1 a_2$$

$$g_2 = a_1 a_2 + b_1 b_2$$

The checker's outputs f_2 and g_2 for code space inputs are as follows:

a_1	b_1	a_2	b_2	f_2	g_2
0	2	0	2	2	0
0	2	2	0	0	2
0	2	1	1	1	1
2	0	0	2	0	2
2	0	2	0	2	0
2	0	1	1	1	1
1	1	0	2	1	1
1	1	2	0	1	1
1	1	1	1	1	1

It can be verified that the code disjoint property is satisfied. Checkers for two rail codes with larger k's can be obtained by cascading in a tree fashion the two-rail checkers for k=2 hence suggesting a proof to the generalized expression for f_k and g_k. The two-rail checker for k=2 is shown in Fig. 3.

This is a MIN-MAX implementation. A MAX-MIN implementation can be obtained from the following formulae:

$$f_k = (f_{k-1} + a_k) \cdot (g_{k-1} + b_k)$$

$$g_k = (f_{k-1} + b_k) \cdot (g_{k-1} + a_k) \quad \text{with}$$

$$f_1 = a_1 \text{ and } g_1 = b_1$$

The two-rail checkers can be used as duplication comparators if one set of inputs (e.g. all a_i or all b_i) is inverted using $x^{(m-1,m-2,\ldots 1,0)}$. This unary operator, however, is negative unate, hence some of the structural properties are lost and the checker is no longer SFS with respect to F_u.

C. SFS Checkers for Berger Codes

Checkers for separable unordered codes under the structural constraints enunciated before possess little well-established theory and implementation methods. This is true for any valued system. Figure 4 gives a schematic of how checking could be achieved. It is highly likely that such an approach would result in a non-unate implementation, However, it may be the only practical implementation. Other ideas can be borrowed from [7,8].

D. General Comments

It is apparent from the previous discussions on checkers that there are several open questions concerning checkers for multi-valued unordered codes. For the fixed-weight codes it would be desirable to be able to determine the existence and the construction rules for checkers with the smallest number of levels. Also modular checkers would enhance the practicability of the codes. Clearly the results on Berger checkers need further development. These topics will be discussed in a subsequent paper.

VI. Conclusions

This paper presents a scheme for decreasing undetected errors that can be produced by faulty multiple-valued combinational logic networks. It is based on a fault model that we believe is much more realistic than the more common stuck-at model. An added feature that we feel is very important is the protection provided against intermittent faults.

Fault secure networks depend on encoded inputs and outputs, and in multiple-valued technologies where pin counts will be of less importance than in current binary technologies, these encodings should cost relatively little in relation to the savings gained from added output reliability.

Constraints on network structure are relatively loose. Future study will be directed at specific network realizations. Emphasis will be placed on more common functional blocks such as adders and check circuits.

References

[1] Su, S. Y. H., "Symposium Chairman's Message," Proceedings of Sixth International Symposium on Multiple-Valued Logic, p. i May 1976.

[2] Von Neumann, J., "Probabilistic Logics and Synthesis of Reliable Organisms from Unreliable Components," Automata Studies, Annals of Math. Studies No. 34, C. E. Shannon and J. McCarthy, eds., Princeton University Press, Princeton, NJ, pp. 43-98, 1956.

[3] Anderson, D. A., "Design of Self-Checking Digital Networks Using Coding Techniques," Coordinated Science Laboratory Report R-527, University of Illinois, Sept. 1971.

[4] Carter, W. C., and P. R. Schneider, "Design of Dynamically Checked Computers," IFIP 68, vol. 2, Edinburg, Scotland, pp. 878-883, Aug. 1968.

[5] Smith, J. E., and G. Metze, "Strongly Fault Secure Logic Networks," to appear IEEE Trans. on Computers, June 1978.

[6] Dussault, J., "On the Design of Self-Checking Systems under Various Fault Models," Coordinated Science Laboratory Report R-791, University of Illinois, October 1977.

[7] Smith, J. E., "The Design of Totally Self-Checking Check Circuits for a Class of Unordered Codes," Journal of Design Automation and Fault-Tolerant Computing, vol. 1, pp. 321-342, October 1977.

[8] Ashjaee, M. J., "Totally Self-Checking Check Circuits for Separable Codes," Ph.D Thesis, University of Iowa, July 1976.

[9] Breuer, M. A., and A. D. Friedman, Diagnosis and Reliable Design of Digital Systems, Computer Science Press, 1976.

[10] Shooman, M. L., Probabilistic Reliability: an Engineering Approach, McGraw-Hill, 1968.

[11] Reddy, S. M., "A Note on Self-Checking Checkers," IEEE Trans. on Computers, vol. C-23, pp. 1100-1102, Oct. 1974.

[12] Anderson, D. A., and G. Metze, "Design of Totally Self-Checking Check Circuits for m-out-of-n Codes," IEEE Trans. on Computers, vol. C-22, pp. 263-269, March 1973.

[13] Diaz, M., "Design of Totally Self-Checking and Fail-Safe Sequential Machines," Digest of the Fourth Annual Symposium on Fault-Tolerant Computing, pp. 3-19 to 3-24, June 1974.

[14] Smith, J. E., "The Design of Totally Self-Checking Combinational Circuits," Coordinated Science Laboratory Report R-737, University of Illinois, Aug. 1976.

[15] Armstrong, D. B., "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Networks," IEEE Trans. on Electronic Computers, vol. EC-15, pp. 66-73, Feb. 1966.

[16] Post, E. L., "Introduction to a General Theory of Elementary Propositions," American Journal of Math., vol. 43, pp. 163-185, 1921.

[17] Rosenbloom, P. C., "Post Algebras I. Postulates and General Theory," American Journal of Math., vol. 64, pp. 167-188, 1942.

[18] Lubell, D., "A Short Proof of Sperner's Lemma," Journal of Comb. Theory, vol. 1, p. 299, Sept. 1966.

[19] Berger, J. M., "A Note on Error Detection Codes for Asymmetric Channels," Information and Control, vol. 4, pp. 68-73, March 1961.

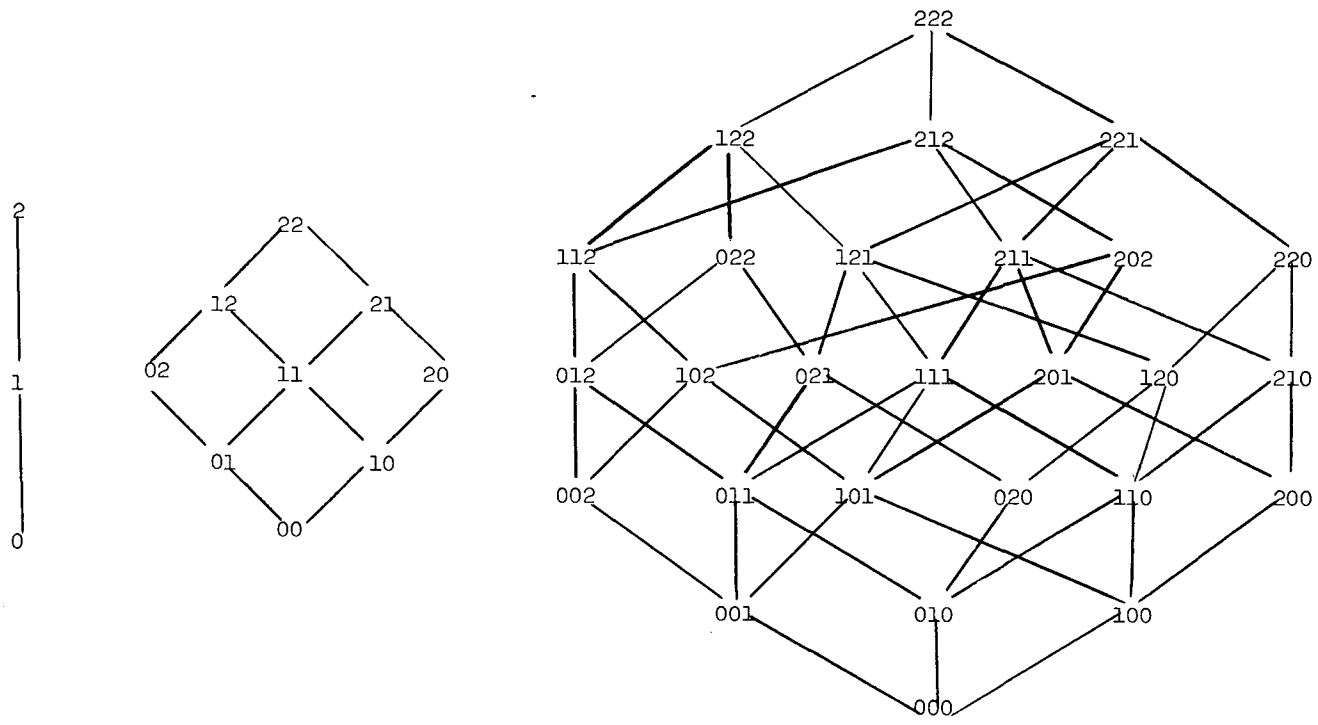

Figure 1. The lattices p, p x p, p x p x p.

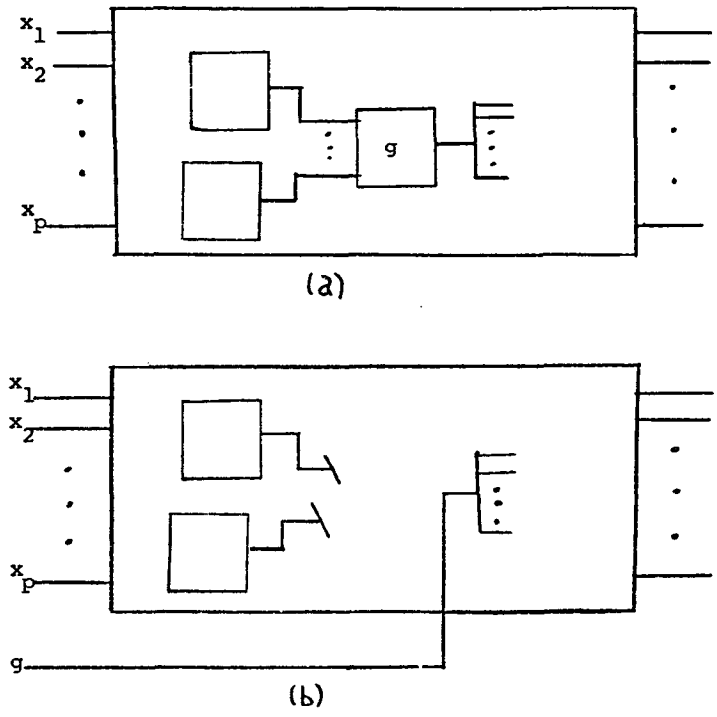

Figure 2. a) A functional block G with faulty gate g.
b) The block G' with the output of g treated as an input.

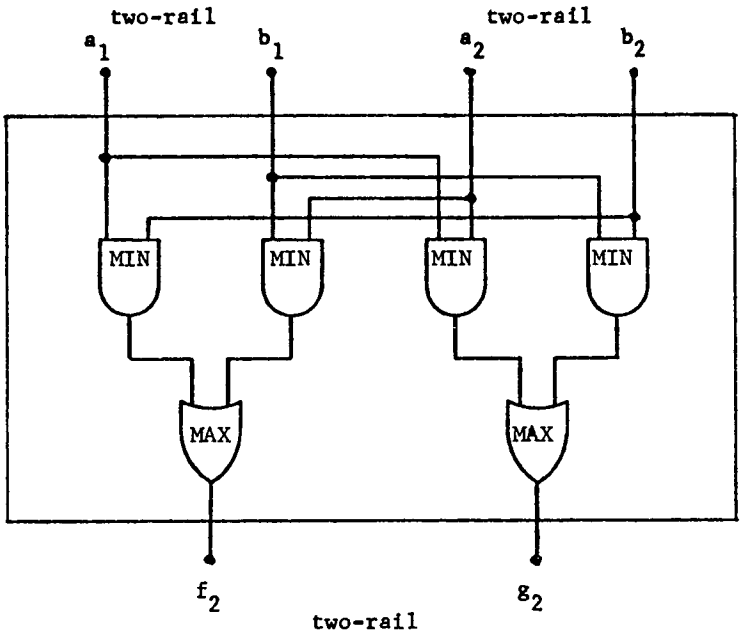

Figure 3. An SFS two-rail checker.

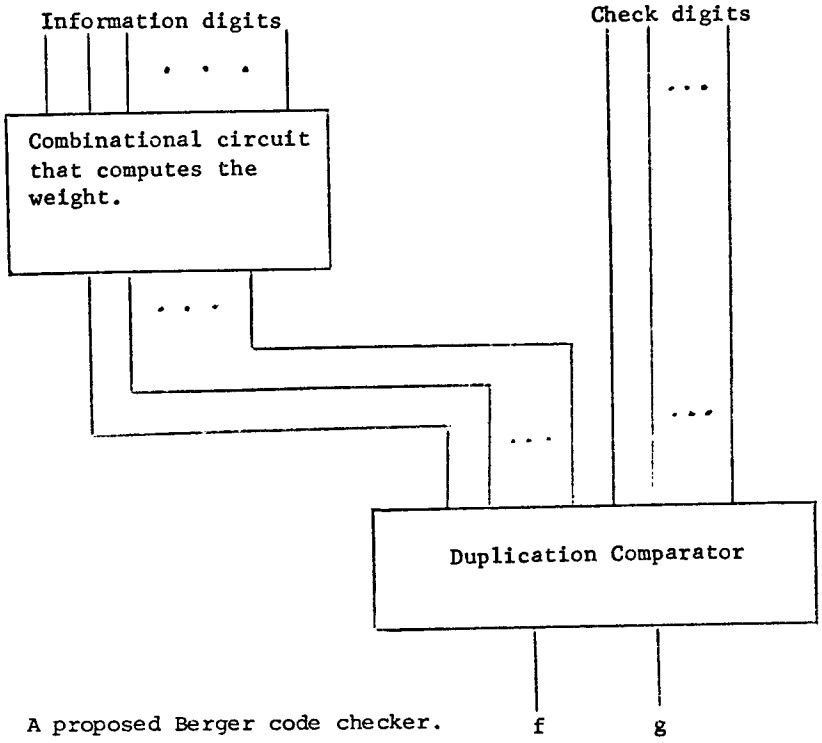

Figure 4. A proposed Berger code checker.