Strongly Fault Secure Logic Networks

JAMES E. SMITH, MEMBER, IEEE, AND GERNOT METZE

Abstract—**Strongly fault secure logic networks are defined and are shown to include totally self-checking networks as a special case. Strongly fault secure networks provide the same protection against assumed faults as totally self-checking networks, and it is shown that when stuck-at faults are assumed a strongly fault secure network can be easily modified to form a totally self-checking network. A class of strongly fault secure networks is defined in terms of network structure. This structural definition of these "path fault secure" networks facilitates their design and implies other interesting properties. Finally, networks that are strongly fault secure with respect to single stuck-at faults are discussed. A large class of these networks is shown to be easily characterized, and network behavior under nonmodeled faults is considered.**

Index Terms—**Error-detecting codes, fault-secure networks, path sensitization, self-checking networks, self-testing networks.**

I. Introduction

"SELF-CHECKING" is a generic term used to describe logic networks that allow on-line detection of hardware failures. In recent years, several methods have been proposed for constructing self-checking logic networks [1]–[4]. These methods typically involve encoding network outputs in an error detecting code. A noncode output word indicates that the output is incorrect and that a fault is present in the network.

A widely studied type of self-checking network is the "totally self-checking" network first proposed by Carter and Schneider in [1] and later studied by Anderson [2]. The theory of totally self-checking networks is based upon certain assumptions regarding modeling of faults and the time interval that separates network faults. Given the fault assumptions, a totally self-checking network always produces a noncode word as the first erroneous output due to a fault. This behavior is obviously desirable and will be referred to as the "totally self-checking goal."

It is possible to construct logic networks that fail to be totally self-checking, but which achieve the totally self-checking goal under precisely the same fault assumptions. Accordingly, we define the "strongly fault secure" property. Every totally self-checking network is strongly fault secure, and if stuck-at faults are assumed, every strongly fault secure network that is not totally self-checking can be converted into a totally self-checking network by simplifying its structure.

In order to construct strongly fault secure networks, it is necessary to consider characteristics of network structures and their associated output encodings. For this reason, we define the "path fault secure" property in terms of network structure and output codes. The path fault secure property implies the strongly fault secure property and has the advantage of being easily verified. Many different path fault secure network structures can be demonstrated, and a few examples are given.

Next we discuss networks that are strongly fault secure for single stuck-at faults since this is a frequently encountered fault model. A set of critical single faults are identified which lead to a sufficient condition for strong fault secureness that is less restrictive than the path fault secure property for single faults. A theorem then follows that allows some modification of strongly fault secure networks while preserving the strongly fault secure property. Finally, the effects of some nonmodeled faults are discussed.

II. Preliminaries

A. Basic Definitions

We consider multiple-input, multiple-output combinational logic networks that are formed as acyclic connections of AND, NAND, OR, and NOR gates and of fan-out points. Inverters are considered to be single-input NAND or NOR gates. If a network G has r primary input lines, then the 2^r binary vectors of length r form the *input space* X of G. The *output space* Y is similarly defined to be the set of 2^q binary vectors of length q, where G has q 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. Fault Assumptions

Except where stated otherwise, we consider *faults* to be sets of gate input and/or gate output lines which become permanently stuck-at a logical 1 or stuck-at a logical 0. Such a fault is denoted as $f_i = \{l_{i1}/d_1, l_{i2}/d_2, \cdots, l_{ik}/d_k\}$ where the l_{ij} are gate input or output lines, $d_j \in \{0, 1\}$, and l_{ij}/d_j denotes the line l_{ij} stuck-at-d_j. We say that k is the *multiplicity* of the fault. Since only gate input and output lines can become stuck, a stuck-at fault on a primary input line must be modeled as a combination of stuck input lines on gates that the primary input feeds.

Manuscript received August 1, 1977; revised February 21, 1978. This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract DAAB-07-72-C-0259. A portion of this research was performed while J. E. Smith was with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

J. E. Smith is with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706.

G. Metze is with the Department of Electrical Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

0018-9340/78/0600-0491$00.75

For the network G, $\boldsymbol{F}_M$ is defined to be the set of all multiple faults. That is, $\boldsymbol{F}_M$ contains all sets of stuck lines in G with the only restriction being that any line only appears once in any given fault. The set of most probable faults in G is a subset of $\boldsymbol{F}_M$ and is called its *assumed fault set*. Two examples of assumed fault sets are *unidirectional faults*, $\boldsymbol{F}_u$, where for any fault all the d_j are the same and any multiplicity is allowed; and single faults, $\boldsymbol{F}_s$, where the multiplicity of each fault is one.

The number of faults in a system is typically modeled as a Poisson process. Hence, it is assumed that members of a fault set $\boldsymbol{F}$ accumulate in G one at a time with some interval of time between. A critical assumption about the length of this time interval will be discussed shortly.

Because faults build up with time, we will denote this behavior by defining a *fault sequence* $\langle f_1, f_2, \cdots, f_n \rangle$ to represent the event where fault f_1 occurs, followed later by f_2 (at which point both f_1 and f_2 are present in the network), and so forth until f_n occurs. It is assumed that once a line becomes stuck at 0 or 1, it remains stuck at that value. Hence, in a fault sequence, if $l_i/d \in f_j$ then l_i does not appear as a stuck line in any f_k, $k > j$. If a stuck line were allowed to change its stuck value due to a later fault, our results would not be affected in any appreciable way, but notation would become more cumbersome.

Since a fault affects the logic function realized by a network G, the output of G under input x and fault f can be denoted as $G(x,f)$. Under the fault-free condition, $f = \varnothing$, the output is $G(x, \varnothing)$.

C. The Totally Self-Checking Property

The following definitions are due to Anderson [2] and refer to a functional block G with input code space $A \subseteq X$, output code space $B \subseteq Y$, and an assumed fault set $\boldsymbol{F}$.

Definition 1: G is *fault secure* with respect to $\boldsymbol{F}$ if for all faults in $\boldsymbol{F}$ and all code inputs, the output is either correct or is a noncode word; i.e., for all $f \in \boldsymbol{F}$ and for all $\boldsymbol{a} \in A$ $G(\boldsymbol{a}, f) = G(\boldsymbol{a}, \varnothing)$, or $G(\boldsymbol{a}, f) \notin B$.

Definition 2: G is *self-testing* with respect to $\boldsymbol{F}$ if for each fault in $\boldsymbol{F}$ there is at least one code input that produces a noncode output; i.e., for all $f \in \boldsymbol{F}$ there is an $\boldsymbol{a} \in A$ such that $G(\boldsymbol{a}, f) \notin B$.

Definition 3: G is *totally self-checking* (TSC) with respect to $\boldsymbol{F}$ if it is fault secure and self-testing with respect to $\boldsymbol{F}$.

The effectiveness of TSC networks is based on two fundamental assumptions:

1) Each failure can be modeled as a member of $\boldsymbol{F}$;

2) faults occur one at a time, and between any two faults a sufficient time elapses so that all code inputs are applied to the network.[1]

With these two assumptions, the first erroneous output due to a fault in a TSC network must be a noncode word; i.e., the TSC goal is achieved.

[1] This assumption may be somewhat stronger than necessary because in many instances a proper subset of the input code words may test for all detectable faults.

III. The Strongly Fault Secure Property

A. Definition

Based on exactly the same two fault assumptions as the TSC property, we can define a more general property which also leads to networks that achieve the TSC goal. We begin by observing that the self-testing property is required because if a network is known only to satisfy the fault secure property for $\boldsymbol{F}$, there could be some fault $f_1 \in \boldsymbol{F}$ which never produces incorrect outputs when code inputs are applied to G. Consequently, the fault would go undetected, and eventually a second fault $f_2 \in \boldsymbol{F}$ could occur. Then, the fault $f_1 \cup f_2$ is present in G, but $f_1 \cup f_2$ may not be in $\boldsymbol{F}$. Hence, there is now no assurance that a code input cannot cause the output to be an incorrect code word.

On the other hand, G may be fault secure and self-testing with respect to $f_1 \cup f_2$ even though $f_1 \cup f_2 \notin \boldsymbol{F}$. Or, to be more general, there may be a sequence of faults such that G is fault secure with respect to any initial subsequence, and is self-testing for the combination of faults in the sequence. This property is formalized in the following definition.

Definition 4: For a fault sequence $\langle f_1, f_2, \cdots, f_n \rangle$, let k be the smallest integer for which there is an $\boldsymbol{a} \in A$ such that

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) \neq G(\boldsymbol{a}, \varnothing).$$

If there is no such k, set $k = n$. Then G is *strongly fault secure* (SFS) with respect to the fault sequence if for all code words $\boldsymbol{a} \in A$ either

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) = G(\boldsymbol{a}, \varnothing) \quad \text{or} \quad G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) \notin B.$$

Definition 5: The network G is *strongly fault secure* (SFS) with respect to the fault set $\boldsymbol{F}$ if G is SFS with respect to all fault sequences whose members belong to $\boldsymbol{F}$.

Under the two fundamental fault assumptions given at the end of Section II, it can be shown that any SFS network achieves the TSC goal. Furthermore, if a network is not SFS, it is always possible to produce an erroneous code output prior to a noncode output without violating the two fundamental assumptions. Hence, SFS networks are the largest class of networks that achieve the TSC goal, given the fault assumptions we made earlier.

B. The Relationship Between SFS and TSC Networks

It is fairly easy to see that every TSC network is SFS. That is, if in Definition 4, $k = 1$ for all fault sequences that can occur in a network, then the network is TSC. On the other hand, a sequence of faults can occur in any SFS network until a point is reached where any additional assumed fault must result in a noncode output for some code input. This observation suggests a method for constructing a TSC network given a SFS network.

A logic network containing a stuck-at fault can be simplified by removing lines (and possibly gates) which are made unnecessary by the fault. For example, if an input to a q-input AND gate is stuck-at-1, one can remove the line to get

a $(q-1)$-input gate. This process is called the "reduction" of the network with respect to the fault. A formal description of the reduction process can be found in [5]. Also, the R-transformation given in [6] is very similar.

Informally, we let $\hat{f}$ be the largest member of $\boldsymbol{F}_M$ such that $\hat{f} \supseteq f$ and $\hat{f}$ can be shown to be equivalent to f by considering the network's structure. If the fault $\hat{f}$ is not unique, then any $\hat{f}$ can be used. Then the *reduction* of G with respect to f is a network G' with all the stuck lines in $\hat{f}$ removed, and with any gates whose output lines are stuck in $\hat{f}$ removed. If the reduction of a network forces the removal of a primary output line, then the output is assumed to take its stuck value in $\hat{f}$, and is immune to any other faults.

Lemma 1: If G' is the reduction of G with respect to f, then for all $x \in X$ $G'(x, \varnothing) = G(x, f)$.

The proof of Lemma 1 is straightforward and can be found in [5].

After a network has been reduced, several faults in the assumed fault set of the original network are no longer possible since they involve lines that have been removed. For this reason, we define a fault set $\boldsymbol{F}'$ for the reduced network to be the *simplification* of the fault set $\boldsymbol{F}$ with respect to the fault f, where $\boldsymbol{F}' = \{f_i \mid f_i \in \boldsymbol{F}$ and no stuck line in f_i is a stuck line in $\hat{f}\}$.

Theorem 1: If G is SFS with respect to $\boldsymbol{F}$ and there is some $f \in \boldsymbol{F}$ such that for all $\boldsymbol{a} \in A$ $G(\boldsymbol{a}, f) = G(\boldsymbol{a}, \varnothing)$, then let G' be the reduction of G with respect to f and $\boldsymbol{F}'$ be the simplification of $\boldsymbol{F}$ with respect to f. Then

1) G' is SFS with respect to $\boldsymbol{F}'$;
2) for all $\boldsymbol{a} \in A$ $G'(\boldsymbol{a}, \varnothing) = G(\boldsymbol{a}, \varnothing)$.

Proof: A rigorous proof of 1) is rather lengthy and depends on a formal definition of reduction which has not been included here. Informally, we have simulated the effects of the fault f in G by removing portions of G rendered inactive by the fault. Any subsequent faults in $\boldsymbol{F}'$ behave exactly as if they had occurred in G simultaneously with f. If any fault sequence $\langle f_1, f_2, \cdots, f_n \rangle$ violates the SFS definition for G', the longer sequence $\langle f, f_1, f_2, \cdots, f_n \rangle$ must violate the definition for G, which is a contradiction.

The proof of 2) follows immediately from Lemma 1 and the fact that for all $\boldsymbol{a} \in A$ $G(\boldsymbol{a}, f) = G(\boldsymbol{a}, \varnothing)$. □

Theorem 1 can be used to construct a TSC network from an SFS network. As long as the fault set for the SFS network contains a fault f such that for all $\boldsymbol{a} \in A$ $G(\boldsymbol{a}, f) = G(\boldsymbol{a}, \varnothing)$, one should iteratively reduce G with respect to f and simplify $\boldsymbol{F}$ with respect to f. Since $\boldsymbol{F}$ is finite, this process must terminate, and the final network must be SFS (by Theorem 1). If we let G'' be the final reduced network and $\boldsymbol{F}''$ be the final simplified fault set, then for all faults in $\boldsymbol{F}''$ there is some $\boldsymbol{a} \in A$ such that $G''(\boldsymbol{a}, f) \neq G''(\boldsymbol{a}, \varnothing)$. Hence, G'' must be TSC with respect to $\boldsymbol{F}''$ since k in Definition 4 must be 1 for all nonempty fault sequences.

In the remainder of the paper, our emphasis will be on the construction of SFS networks. Nevertheless, if TSC networks are desired for stuck-at faults we have shown that SFS networks can be converted to TSC networks via the reduction process.

Before we continue, some additional comments and observations are in order. First, the use of the reduction process in the design of TSC combinational networks is not entirely original with us; a restricted version was first proposed in [10] where untested AND gate inputs were removed from two-level AND/OR networks.

Second, the success of the reduction process is based on a stuck-at fault assumption. To be more specific, it relies on the fact that simulating the effects of stuck-at faults results in a simpler network structure with fewer gates and lines. Consequently the reduction process must eventually terminate with a TSC network. For more general fault models, this may not be the case. An example of such a fault model is one that assumes gates can fail such that they realize any arbitrary logic function. For single faults of this more general type, a reduction-like process breaks down. Consider a gate that realizes the function g, and the gate failing to the function g' is an undetectable fault. The function g' can be at least as complex as g, and if we simulate the effects of the fault by replacing the gate with one that realizes g' then the new gate failing to the function g is undetectable. Hence, modification of the gate is again called for, and the process never terminates. On the other hand, for faults of this general type, bit-sliced network structures with parity codes (see Section IV-B) are SFS. This is evidence that if the fault model is general enough, it may be impossible to construct a TSC network while it is possible to construct SFS networks.

Third, one might ask if every fault secure network is strongly fault secure. If this were true, then the self-testing property in the TSC definition would be unnecessary, and a simpler definition of SFS networks would be possible. Unfortunately, it is not true. Any quadded logic network [7] is fault secure with respect to single faults. One can take nearly any quadded logic network with at least four levels and find a sequence of single faults that can lead to an incorrect code output prior to any noncode output while satisfying the two fault assumptions given earlier.

Finally, we should point out that if some particular mapping of *noncode* inputs is desired, e.g., if one wanted a code disjoint [2] network, then the reduction process may destroy this mapping. On the other hand, there are cases when it will not; the method for constructing TSC check circuits in [8] is such a case.

IV. The Path Fault Secure Property

A. Definition

The definitions of SFS and TSC networks consist of a functional description of their behavior under faults. However, in order to construct a network that is either SFS or TSC, one must be able to characterize SFS or TSC networks in terms of gate interconnection structures and codes. To this end, the functional definitions have little value.

In this section, we define the "path fault secure" property in terms of network structures and codes. Path fault secure networks are a subclass of SFS networks, and can consequently be converted into TSC networks. The advantage

of path fault secure networks is that their definition suggests techniques for their construction.

We begin by discussing a relationship between circuit structures and potential output errors using the well-known concept of path sensitization [9]. Path sensitization was originally used for single stuck-at fault detection. Here, an input vector sensitizes a path from a potentially faulty line to a primary output line if a change in the value of the line due to a stuck-at fault causes the values on all of the lines on the path (including the primary output) to change as well. A path may split at fan-out points and then reconverge at some subsequent gate along the path.

A 1-*error* (0-*error*) is an error on a line which results from the value on the line changing from a correct 0(1) to an incorrect 1(0). When no distinction is necessary, we use the term d-error where $d \in \{0, 1\}$. If a path p from some internal line to a primary output passes through a total of n NAND gates and NOR gates, then we define the *inversion parity* of p, $IP(p)$, to be n modulo 2.

We observe that if the value of a line on the sensitized path is 0(1) and the line feeds a NAND gate or NOR gate, then the gate output which is also on the sensitized path must have the value 1(0). Similarly, a 0(1) on a line feeding an AND or OR gate implies the gate output is also a 0(1) if the lines are on a sensitized path. Therefore, a d-error on a primary output line l_j resulting from the fault $\{l_i/d_i\}$ implies that there is a sensitized path p from l_i to l_j such that $d = d_i \oplus IP(p)$.

When faults with multiple stuck lines are considered, then a generalization of the preceding statement results which is given in the following lemma.

Lemma 2: A d-error on primary output l_j under fault f implies that there is a component of f, l_i/d_i, such that there is a path p from l_i to l_j and $d = d_i \oplus IP(p)$.

It should be noted that in Lemma 2 we say a *path* exists from some single component fault. A *sensitized path* does not necessarily exist from any single component fault since a number of single component faults may need to conspire to result in an error. However, in this case each component which is necessary for the error to occur satisfies the lemma.

Lemma 2 indicates a necessary condition for a d-error to occur at an output. It also provides a method for generating a superset of all erroneous outputs which can occur due to a fault. This method will now be demonstrated.

We begin by considering the network of Fig. 1 with the fault $f_1 = \{l_5/1\}$. We observe that there is no path from l_5 to either primary output y_1 or y_4 and that paths to y_2 and y_3 both have an inversion parity of 1 (there is an odd number of inverters). Then it is impossible for f_1 to affect either y_1 or y_4; i.e., they will always assume their normal values. On the other hand, a path may be sensitized to either or both outputs y_2 and y_3. However, any output errors must be 0-errors since the fault is a stuck-at-1 with an inversion parity of 1. To summarize, in response to any input vector one of the following conditions holds:

1) all four outputs are normal (i.e., none of the paths are sensitized),

2) outputs y_1, y_2, and y_4 are normal, but output y_3 contains a 0-error,

3) outputs y_1, y_3, and y_4 are normal, but output y_2 contains a 0-error,

4) only outputs y_1 and y_4 are normal with outputs y_2 and y_3 containing 0-errors.

One can denote each of these conditions as a q-component vector (one component for each output). Each component is either n (the output is normal), 0 (there can be a 0-error), or 1 (there can be a 1-error). For our example, the vectors are (n, n, n, n), $(n, 0, n, n)$, $(n, n, 0, n)$, and $(n, 0, 0, n)$.

More formally, the *sensitization set for a fault* f, denoted as $\sigma(f)$, is the largest set of vectors $(\alpha_1, \alpha_2, \cdots, \alpha_q)$ such that $\alpha_k \in \{0, 1, n\}$ and $\alpha_k \neq n \Rightarrow$ there is a stuck line $l_i/d_i \in f$ and there is a path p from l_i to primary output line l_k where $\alpha_k = d_i \oplus IP(p)$. The members of a sensitization set are called *sensitization vectors.*

The set $\sigma(f)$ represents all potential sets of sensitized paths which can occur as a result of f. In particular, if a d occurs in some position of a sensitization vector, then a path exists which, if sensitized, leads to a d-error at the corresponding output. In order to derive the output words that may actually result due to a fault, we define the operator © which operates on y, a member of the network output space, and a sensitization vector α:

$$(y_1, y_2, \cdots, y_q) \copyright (\alpha_1, \alpha_2, \cdots, \alpha_q) = (\beta_1, \beta_2, \cdots, \beta_q)$$

where

$$\beta_i = y_i \quad \text{if } \alpha_i = n$$
$$\beta_i = 0 \quad \text{if } \alpha_i = 0$$
$$\beta_i = 1 \quad \text{if } \alpha_i = 1.$$

We extend the © operator to a sensitization set $\sigma(f)$ in a straightforward manner:

$$\{y \copyright \sigma(f)\} = \bigcup_{\alpha \in \sigma(f)} \{y \copyright \alpha\}.$$

Then $\{y \copyright \sigma(f)\}$ includes all the output words that can appear at a network's output under the fault f when y is the correct output. Or, alternatively,

Lemma 3: Given a network G with input space X and multiple fault set F_M, for all $x \in X$ and for any $f \in F_M$, $G(x, f) \in \{G(x, \varnothing) \copyright \sigma(f)\}$.

Lemma 3 provides a simple way to determine a superset of erroneous outputs which can result from a given fault. The superset is determined solely by determining if paths are available to transmit the effect of faults to primary outputs. Thus, we have successfully bounded the error generation capability of a circuit by examining an easily recognized part of the circuit's structure.

We now make useful extensions of our definitions. The sensitization set for a fault set F is $\Sigma(F)$, where

$$\Sigma(F) = \bigcup_{f \in F} \sigma(f).$$

The © operator is generalized to $\Sigma(F)$ by saying that

$$\{y \copyright \Sigma(F)\} = \bigcup_{\alpha \in \Sigma(F)} \{y \copyright \alpha\}.$$

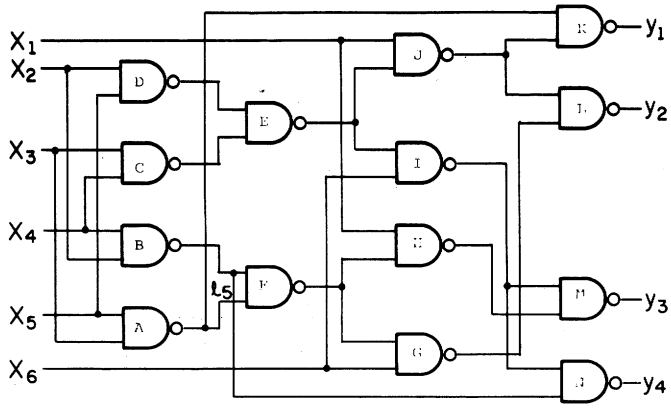


Fig. 1. The network G for Example 1.

Definition 6: A network G with fault set $\boldsymbol{F}$ is *path fault secure* (PFS) with respect to $\boldsymbol{F}$ if for each output code word b, the members of $\{b \circledcirc \Sigma(\boldsymbol{F})\}$ are b and some noncode output words; i.e., for all $b \in B$ $\{b \circledcirc \Sigma(\boldsymbol{F})\} \subseteq \{b\} \cup Y - B$.

Theorem 2: If G is PFS with respect to $\boldsymbol{F}$, it is SFS with respect to $\boldsymbol{F}$.

Proof: For an arbitrary fault sequence, $\langle f_1, f_2, \cdots, f_n \rangle$, $f_i \in \boldsymbol{F}$, let k be the smallest integer for which there is an $\boldsymbol{a} \in A$ such that

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) \neq G(\boldsymbol{a}, \varnothing).$$

If there is no such k, we set $k = n$ according to Definition 4, and it must be true that for all $\boldsymbol{a} \in A$

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) = G(\boldsymbol{a}, \varnothing),$$

so G is SFS with respect to the sequence.

If there is such a k, then for all $\boldsymbol{a} \in A$

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k-1} f_j\right) = G(\boldsymbol{a}, \varnothing).$$

Now, the presence of the fault

$$\bigcup_{j=1}^{k-1} f_j$$

in G does not change the inversion parity on any path and cannot add any additional paths to G. Hence, $\sigma(f_k)$ still bounds the maximum error generating capability of f_k, even if $f_1, f_2, \cdots, f_{k-1}$ are also present. Hence, from Lemma 3 it follows that for all $\boldsymbol{a} \in A$

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) \in \left\{G\left(\boldsymbol{a}, \bigcup_{j=1}^{k-1} f_j\right) \circledcirc \sigma(f_k)\right\}.$$

Since

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k-1} f_j\right) = G(\boldsymbol{a}, \varnothing),$$

it must also be true that for all $\boldsymbol{a} \in A$

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) \in \{G(\boldsymbol{a}, \varnothing) \circledcirc \sigma(f_k)\}.$$

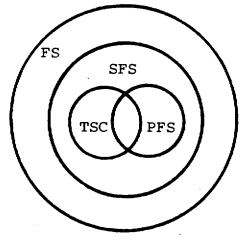


Fig. 2. The relationships that exist among the sets of fault secure, strongly fault secure, path fault secure, and totally self-checking networks.

Because G is PFS,

$$\{G(\boldsymbol{a}, \varnothing) \circledcirc \sigma(f_k)\} \subseteq \{G(\boldsymbol{a}, \varnothing)\} Y - B,$$

so for all $\boldsymbol{a} \in A$

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) \in \{G(\boldsymbol{a}, \varnothing)\} \cup Y - B.$$

This is just another way of saying that for all $\boldsymbol{a} \in A$

$$G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) = G(\boldsymbol{a}, \varnothing) \quad \text{or} \quad G\left(\boldsymbol{a}, \bigcup_{j=1}^{k} f_j\right) \notin B.$$

Therefore, by Definition 4, G is SFS with respect to the fault sequence.

Since the above argument was applied to an arbitrary fault sequence, it must hold for all. Consequently, Definition 5 is satisfied. □

To place the properties we have discussed thus far into perspective, Fig. 2 shows the relationships that exist among the sets of fault secure, strongly fault secure, path fault secure, and totally self-checking networks.

PFS networks enjoy an advantage over TSC networks in that they have a concise definition in terms of network structure. Furthermore, verifying that a network is PFS is conceptually simple although an exhaustive verification of Definition 6 could be time consuming. Nevertheless, following sections show that networks can often be shown to be PFS without resorting to exhaustive verification of Definition 6.

PFS networks have another advantage when they are to be placed in a system. A network may be designed with an input code space A, but when it is placed in an actual system environment, it may actually receive some proper subset of A, say A', during normal operation. In this situation, a network designed to be TSC may fail to be TSC because some member of $A - A'$ may be required to test for a modeled fault. On the other hand, a PFS network remains PFS as the following theorem shows.

Theorem 3: Let G have input code space A, output code space B and fault set $\boldsymbol{F}$. If G is PFS with respect to $\boldsymbol{F}$, then the same network G with input code space $A' \subset A$ is PFS with respect to $\boldsymbol{F}$.

Proof: If the input code space becomes A', the output code space must become $B' \subseteq B$, and $Y - B \subseteq Y - B'$. Referring to Definition 6, since $B' \subseteq B$, then for all $b \in B'\{b \circledcirc \sigma(\boldsymbol{F})\} \subseteq Y - B \subseteq Y - B'$. Hence, G is PFS with respect to input code space A'. □

B. Examples of PFS Network Structures

We now use the concepts introduced in the previous section to derive network structures that are PFS. We begin by considering unidirectional stuck-at fault sets.

A binary vector x *covers* binary vector y ($x \geq y$) if x has a 1 in every position y has a 1. A code C is *unordered* if no member of the code covers any other member. Finally, an *inverter-free network* is just what the name implies; it contains only AND gates and OR gates.

Lemma 4: For an inverter-free network with a unidirectional fault set F_u, $\Sigma(F_u)$ contains no sensitization vector with both a 0 and a 1.

Proof: Since there are no inversions in the network, for any p, $IP(p) = 0$. From the definition of $\sigma(f)$ we see that if f is a unidirectional stuck-at-d fault, all of the α_k which are not n must be d. Consequently, each $\sigma(f)$ contains only vectors which have d's and n's. □

Theorem 4: Any inverter-free network with an unordered output code space is path fault secure with respect to unidirectional faults.

Proof: Let B be an unordered output code space with $b \in B$. Then if σ is a sensitization vector in $\sigma(f)$ with 1's and n's only, $b \copyright \sigma = b'$ where b' has 1's in every position b has 1's. Then $b' \geq b$. Since B is unordered, $b' = b$ or $b' \notin B$. Or alternatively, $b' \in \{b\} \cup Y - B$.

We can arrive at the same conclusion if σ has 0's and n's only. Therefore, by Lemma 4, in an inverter-free network with an unordered output code space, for all $b \in B$ $\{b \copyright \Sigma(F)\} \subseteq \{b\} \cup Y - B$. □

The use of inverter-free networks with unordered codes (typically m-out-of-n codes) have been suggested for realizing the combinational portion of TSC sequential machines [10], [11]. We have just shown that such combinational networks must also be PFS.

The use of an inverter-free network forces restrictions on the input code space that can be used. However, it has been shown [12] that if an unordered input code space is used, then any mapping can be realized without inverters. Consequently, the use of unordered code spaces at both the input and output and the use of inverter-free networks lead directly to PFS networks for unidirectional faults. The fact that it is sufficient for both code spaces to be unordered also aids in the interconnection of PFS networks.

Next, we turn to a set of network structures and output codes which lead to networks that are PFS with respect to single stuck-at faults.

A network is *b-byte sliced* if the outputs can be grouped into sets of b bits each such that each set is realized by an independent network having only primary inputs in common with the networks realizing the other sets. Fig. 3 illustrates a b-byte sliced network structure. A *b-byte distance two* code is a code in which the bits are grouped into bytes of size b such that if each byte is considered to be a single code position, then any two members of the code differ in at least two positions.

Lemma 5: For a b-byte sliced network with single fault set F_s, $\Sigma(F_s)$ contains sensitization vectors with 1's and 0's in at most one byte.

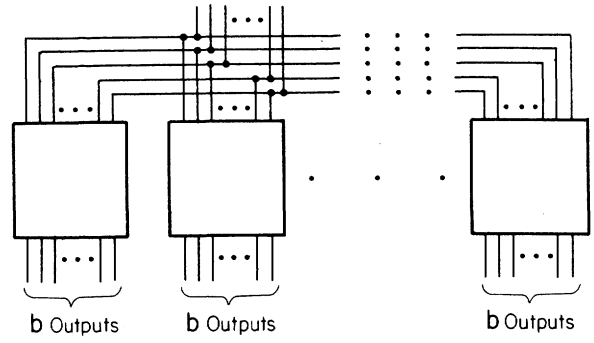


Fig. 3. A b-byte sliced network.

Proof: Since no gates are used to form outputs belonging to more than one byte, all paths from any fault site to an output end at outputs belonging to the same byte. Therefore, a sensitization vector for any single fault can only have 1's and 0's in the particular byte which the faulty gate is used to realize. □

To avoid misunderstanding Lemma 5, one should recall that single faults are defined to occur at gate inputs and outputs; primary input faults are not explicitly included.

Theorem 5: A *b-byte sliced network* with an output code space which is a b-byte distance two code where each byte of the code is produced by a single byte slice of the circuit is PFS with respect to single faults.

Proof: Let B be a b-byte distance two code space with $b \in B$. Then if σ is a sensitization vector in $\Sigma(F_s)$ with 1's and 0's in at most one byte, $b \copyright \sigma = b'$ where b' differs from b in at most one byte position. Since the code is distance two when each byte is considered to be a separate position, $b' = b$ or $b' \notin B$. Or, alternatively, $b' \in \{b\} \cup Y - B$. The same is true for all b and all sensitization vectors in $\Sigma(F_s)$ so for all $b \in B$ $\{b \copyright \Sigma(F_s)\} \subseteq \{b\} \cup Y - B$. □

There are a number of b-byte distance two codes. Checksum codes are perhaps the most widely known and have been proposed for constructing TSC adders [13]. Another such code is the distance two b-adjacent code [14]. A special case of this network structure/code is a bit sliced structure with a simple parity code at the output [2], [15].

Many other examples of path fault secure network structure/code combinations can be found. For example, a two-level network with maximum fan out of k is path-fault secure with respect to single faults if a distance $k + 1$ output code is used. This is because any sensitization vector for such a circuit can have at most a total of k 1's and 0's.

V. Single Stuck-At Fault Sets

We now discuss the SFS and PFS properties for single stuck-at fault sets. We choose to give single faults emphasis because they are perhaps the most commonly used fault assumption. We first develop a sufficient condition for a network to be SFS with respect to single faults. The condition is more general than the PFS property (more networks satisfy it), and it is computationally simpler to verify.

A structural network component of interest here is the *single-output subnetwork*. As the name implies, these subnetworks have just one output line and include so-called "tree"

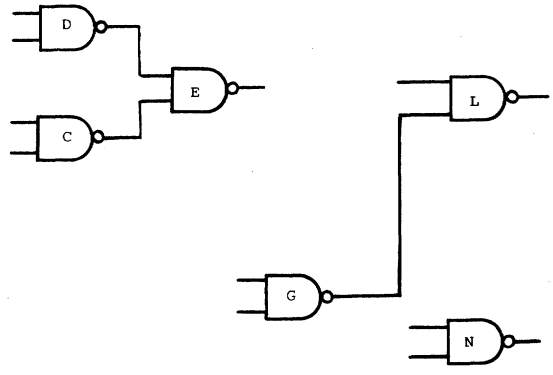


Fig. 4. Three single-output subnetworks of the network shown in Fig. 1.

subnetworks, i.e., those that contain no fan out. Also included are subnetworks with fan out, provided the fan out reconverges within the subnetwork. Fig. 4 shows various single-output subnetworks of the network shown in Fig. 1. A single-output subnetwork is *maximal* if it cannot be included in a larger single-output subnetwork.

Now, let S be the set of maximal single-output subnetworks of the network G such that all the input lines of the subnetworks in S are primary inputs of G or fan outs of primary inputs of G. In Fig. 1, S contains the subnetwork containing gates C, D, and E, and the single-gate subnetworks A and B. The maximal single-output subnetwork containing gates G and L is not in S because all of its inputs are not primary inputs. It can be shown that for a given network G, the set S is unique.

The *critical fault set* F_c for G is the set of all single stuck-at faults on the output lines of subnetworks in S. For Fig. 1, F_c consists of the six stuck-at faults on the outputs of gates A, B, and E. In general, the size of F_c is much smaller than the set of single faults F_s as suggested by the above example.

Lemma 6: For all $x \in X$ and for all $f \in F_s$ $G(x,f) \in \{G(x, \phi) \circledcirc \Sigma(F_c)\}$ (note the different subscripts on F).

Proof: The set of single faults can be partitioned into two sets: those that are on a gate in a subnetwork in S, and those that are not.

Say f is a single fault on a gate in a subnetwork belonging to S with output line l_c. All sensitized paths from f must pass through l_c. Consequently, it follows that any set of sensitized paths from f must be implied by $\sigma(\{l_c/1\}) \cup \sigma(\{l_c/0\}) \subseteq \Sigma(F_c)$, and for all $x \in X$ $G(x,f) \in \{G(x, \varnothing) \circledcirc \Sigma(F_c)\}$.

Now, say f is a single fault in some maximal single-output subnetwork not in S with output l_n. As before, any sensitized paths from f must pass through l_n, so all potential sensitized paths from f must be implied by $\sigma(\{l_n/1\}) \cup \sigma(\{l_n/0\})$. From the definition of S, it can be shown that for every maximal single-output subnetwork not in S, there must be a path beginning at the output line of some member of S and ending at the output line of the subnetwork not in S. Say l_c is the output of a member of S, and there is a path from l_c to l_n, call this path ρ. Any path from l_n to an output can be lengthened by adding ρ to get a path from l_c to the output. Say $IP(\rho) = 0$, then it follows that $\sigma(\{l_n/1\}) \subseteq \sigma(\{l_c/1\})$ and $\sigma(\{l_n/0\}) \subseteq \sigma(\{l_c/0\})$. Alternatively, if $IP(\rho) = 1$ then $\sigma(\{l_n/1\}) \subseteq \sigma(\{l_c/0\})$ and $\sigma(\{l_n/0\}) \subseteq \sigma(\{l_c/1\})$. In either case, $\sigma(\{l_n/1\}) \cup \sigma(\{l_n/1\}) \subseteq \sigma(\{l_c/1\}) \cup \sigma(\{l_c/0\})$. Therefore, for all $x \in X$ $G(x,f) \in \{G(x, \varnothing) \circledcirc \Sigma(F_c)\}$.

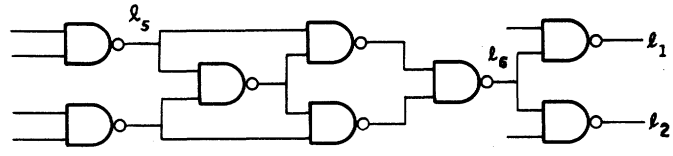


Fig. 5. A portion of an SFS network.

Since f is either in a member of S or not in a member of S, and we have considered both cases, the lemma follows. □

Theorem 7: If network G is PFS with respect to the critical fault set F_c, then G is SFS with respect to all single faults F_s.

Proof: The proof of this theorem is very similar to the proof of Theorem 2 except that Lemma 6 is used in place of Lemma 3. □

The network of Fig. 1 points out the utility of Theorem 7. That is, it is only necessary to show that the network is PFS with respect to six single faults in order to conclude that it is SFS with respect to all single faults, of which there are 84.

The theorem is also useful when it comes to constructing SFS networks. This is because it allows a wider variety networks than the PFS property. This is illustrated in Fig. 5 which shows a portion of a network used to realize the outputs l_1 and l_2 of a SFS network. There is a path from l_5 to l_1 with an even number of inverters, and a path from l_5 to l_2 with an odd number of inverters. Hence, vectors of the form $(1, 0, \cdots)$ can be members of $\sigma(\{l_5/1\})$. However, l_5 belongs to the maximal single-output subnetwork with output line l_6. Say this subnetwork is in S. Then

$$\sigma(\{l_6/1\}) \cup \sigma(\{l_6/0\}) = \{(1, 1, \cdots), (1, 1, \cdots), \cdots (0, 0, \cdots), (0, 0, \cdots)\},$$

and a vector $(1, 0, \cdots)$ cannot possibly be a member. Hence, $\Sigma(F_c) \subset \Sigma(F_s)$ and the network may fail to be PFS for F_s while it is PFS for F_c (and, therefore, SFS for F_s).

We now present a theorem that allows an SFS network to be modified while retaining the SFS property.

Theorem 8: Let G be a network that is PFS with respect to F_c, and let N be any single-output subnetwork in G. If any irredundant network realizing the same function as N is substituted for N, then G is still PFS with respect to F_c.

Proof: If the replaced subnetwork is included in a member of S the sensitization sets for the faults in F_c remain unchanged, and the theorem follows.

If the replaced subnetwork is not in a member of S, the fact that it is irredundant can be used to show that there cannot be a path from any of its inputs to its output with odd (even) inversion parity if the original N did not have such a path with odd (even) inversion parity. Consequently, all paths through the replacement have a counterpart in the original with respect to inversion parity. Therefore, the sensitization set for F_c in the modified network must be a subset of the sensitization set for F_c in the original network, and the theorem follows. □

Since $F_c \subseteq F_s \subseteq F_u$, it is easy to see that if a network is PFS with respect to F_u, it is PFS with respect to F_c. Therefore, the inverter-free networks discussed earlier are PFS with respect to F_c, and Theorem 8 can be applied to them. In particular, inverters can be introduced by substituting NAND or NOR equivalents for individual gates, or for larger single-output subnetworks. Various simplifications (e.g., converting two inverters on a line to a simple line) are also possible.

Another consequence of the theorem is that complex "gates," such as EXCLUSIVE-OR's and multiplexers, can be used in the design of PFS networks regardless of the way the "gates" are finally implemented; inversion parities are simply determined from the functional definitions of the "gates."

Networks that are PFS with respect to F_c also offer protection against a much larger variety of faults than single stuck-at faults. Let F_t be the set of all multiple stuck-at faults whose components are all located within the same single-output subnetwork of G.

Theorem 9: If G is PFS with respect to F_c then G is SFS with respect to F_t.

Proof: If a multiple fault is confined to a single-output subnetwork, all sensitized paths must pass through the output of the subnetwork. This observation with an argument similar to those used in Lemma 6 can be used to prove the theorem. □

We now consider a class of faults that cannot be modeled as sets of stuck lines. Given a single-output subnetwork N, a *unateness preserving fault* is a fault that changes the function realized by N to any other function, provided that the new function is positive unate (negative unate) in the same subnetwork inputs for which the original function was positive unate (negative unate).

Let F_p be the set of unateness preserving faults in G. If a single-output subnetwork is not unate in any variables (e.g., the EXCLUSIVE-OR), its failure to any other function is included in F_p. Since gates are single-output subnetworks, all bridging faults [16] involving inputs of the same gate are also in F_p.

Theorem 10: If G is PFS with respect to F_c (or F_s), then G is SFS with respect to F_p.

Proof: If a unateness preserving fault causes a primary output error, it must first cause an error on the output line of a single-output subnetwork to which the fault belongs. Hence, the erroneous output is the same as if the output line were simply stuck, and must therefore be a noncode word. If the fault causes no primary output errors, an argument similar to the one used in Theorem 8 can be used to show that the faulty network has a $\Sigma(F_c)$ that is a subset of $\Sigma(F_c)$ in the fault-free network. In Theorem 8, this followed from the fact that the new network was irredundant and realized the same function as the original. Here, it follows from the unateness preserving nature of faults.

By the very same argument, it follows that each subsequent member of F_p either yields only noncode outputs or correct outputs, or results in a network that is PFS with respect to F_c. Calling on the definition of a strongly fault secure network completes the proof. □

VI. Conclusions

In this paper we have presented a class of networks that are a superset of the class of totally self-checking networks, and which provide exactly the same protection against modeled faults under the same assumptions.

Key properties of these strongly fault secure networks are as follows.

1) They offer a complete characterization of all the networks that achieve the totally self-checking goal under the two assumptions that are fundamental to most self-checking methods.

2) Assuming stuck-at faults, any strongly fault secure network can be easily converted to form a totally self-checking network. Hence, any methods used to build strongly fault secure networks would apply equally well to the construction of totally self-checking networks.

3) When other than stuck-at faults are assumed, it may at times be impossible to build a totally self-checking network, while a strongly fault secure network can be built.

4) In order to have general methods for constructing self-checking networks of any kind, one must be able to characterize self-checking network structures. This has been an obstacle to the construction of totally self-checking networks. However, for strongly fault secure networks such characterizations can be rather simple as the sections on path fault secure networks show.

5) The class of strongly fault secure networks that we defined to be path fault secure have the property that they can be embedded in a system where they only receive some subset of their input code space during normal operation, and they are still path fault secure.

6) A large class of strongly fault secure networks are easily characterized under the single fault assumption. These networks can be shown to offer protection against a large variety of faults other than single faults.

A final property that some strongly fault secure networks may have which we have not discussed is the following:

7) Although one might infer that modeled faults that go undetected in a strongly fault secure network are accidental,

this is not necessarily the case. In particular, the class of strongly fault secure networks includes networks that contain intentional masking redundancy [7], and which have the property that when the redundancy "wears out," all errors are noncode words that indicate the unprotected condition of the system. Networks of this type may provide an interesting area of future research.

References

[1] W. C. Carter and P. R. Schneider, "Design of dynamically checked computers," in *IFIP 68*, vol. 2, Edinburgh, Scotland, pp. 878–883, Aug. 1968.

[2] D. A. Anderson, "Design of self-checking digital networks using coding techniques," Coordinated Science Laboratory Rep. R-527, University of Illinois, Urbana, IL, Sept. 1971.

[3] D. Reynolds and G. Metze, "Fault detection capabilities of alternating logic," in *Proc. 1976 Int. Symp. Fault-Tolerant Computing*, pp. 157–162, June 1976.

[4] J. F. Wakerly, "Partially self-checking circuits and their use in performing logical operations," *IEEE Trans. Comput.*, vol. C-23, pp. 658–666, July 1974.

[5] J. E. Smith, "The design of totally self-checking combinational circuits," Coordinated Science Laboratory Rep. R-737, University of Illinois, Urbana, Aug. 1976.

[6] E. J. McCluskey and F. W. Clegg, "Fault equivalence in combinational logic networks," *IEEE Trans. Comput.*, vol. C-20, pp. 1286–1293, Nov. 1971.

[7] J. G. Tryon, *Redundancy Techniques for Computing Systems*, Wilcox and Mann, Eds. Washington, DC: Spartan Books, 1962, pp. 205–228.

[8] J. E. Smith, "The design of totally self-checking check circuits for a class of unordered codes," *J. Design Automation and Fault-Tolerant Comput.*, vol. 1, pp. 321–342, Oct. 1977.

[9] D. B. Armstrong, "On finding a nearly minimal set of fault detection tests for combinational logic networks," *IEEE Trans. Electron. Comput.*, vol. EC-15, pp. 66–73, Feb. 1966.

[10] M. Diaz, "Design of totally self-checking and fail-safe sequential machines," in *Dig. 4th Annu. Symp. on Fault-Tolerant Computing*, pp. 3–19 to 3–24, June 1974.

[11] F. Ozguner, "Design of totally self-checking asynchronous and synchronous sequential machines," in *Proc. 7th Annu. Int. Conf. on Fault-Tolerant Computing*, pp. 124–129, June 1977.

[12] G. Mago, "Monotone functions in sequential circuits," *IEEE Trans. Comput.*, vol. C-22, pp. 928–933, Oct. 1973.

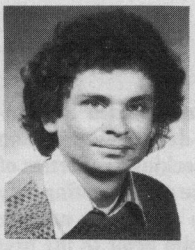
[13] J. F. Wakerly, "Checked binary addition with checksum codes," *J. Design Automation and Fault Tolerant Computing*, vol. 1, pp. 18–27, Oct. 1976.

[14] D. C. Bossen, "*B*-adjacent error correction," *IBM J. Res. Develop.*, vol. 14, pp. 402–408, 1970.

[15] C. E. Allen, "Design of digital memories that tolerant all classes of defects," Stanford University Rep. SU-SEL-66-031, Stanford Univ., Stanford, June 1966.

[16] K. C. Y. Mei, "Bridging and stuck-at faults," *IEEE Trans. Comput.*, vol. C-23, pp. 720–727, July 1974.

✣



James E. Smith (S'74–M'76) was born in Quincy, IL, on November 29, 1950. He received the B.S. degree in electrical engineering and computer science and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana, in 1972, 1974, and 1976, respectively.

From 1972 to 1976 he was a Research Assistant at the Coordinated Science Laboratory, University of Illinois, working in computer arithmetic and fault-tolerant computing. In 1976 he joined the faculty of the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, where he is an Assistant Professor. His current research interests are in fault-tolerant computing and parallel computation.

Dr. Smith is a member of the Association for Computing Machinery and Sigma Xi.

✣

Gernot Metze received the B.S. degree in electrical engineering from Iowa State University, Ames, in 1953, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana-Champaign in 1955 and 1958, respectively.

From 1957 to 1966 he was associated with the Digital Computer Laboratory at the University of Illinois where he participated in the design and construction of the Illiac II computer. He is presently a Professor of Electrical Engineering and a Research Professor in the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign, where he is concerned with the development of courses in information processing, switching and automata theory, and digital system architecture, and directs research projects on applications of multi-valued logics to switching theory and on methods for the automated design of fault-tolerant digital systems. He is the author of several papers in the areas of digital computer arithmetic, digital system design, and fault-tolerant computing, and is a coauthor of *Fault Diagnosis of Digital Systems* (New York: Wiley-Interscience, 1970).

Dr. Metze is a member of Eta Kappa Nu, Tau Beta Pi, Pi Mu Epsilon, Sigma Xi, the Society for Basic Irreproducible Research, and is listed in *Who's Who in the Midwest* and *American Men and Women of Science.*