A Theory of Totally Self-Checking System Design

JAMES E. SMITH, MEMBER, IEEE AND PAKLIN LAM

Abstract—**A totally self-checking digital system uses error detecting codes at subsystem interfaces to detect faults before they can lead to harmful undetected errors. This paper develops a formal model for studying totally self-checking systems.**

Totally self-checking systems are first defined, and, because the propagation of errors is of critical interest, properties characterizing error propagation are defined. For a given subsystem the "propagation graph" is used to represent the error propagation characteristics. The model is completed by defining a system's "interconnection graph" which is formed by connecting the propagation graphs of the subsystems. Then sufficient conditions for which a system is totally self-checking are stated in terms of the model. The paper concludes with applications of the model including system design, checker placement, data contamination analysis, and fault diagnosis.

Index Terms—**Checker placement, detecting codes, error propagation, fault secure, self-testing, totally self-checking systems.**

I. INTRODUCTION

SINCE the first computers, error detecting codes have been used to detect faults in logic circuits. These circuits are given the generic name "self-checking" because they are constantly being checked by normally occurring input sequences as they perform useful computation. When a fault occurs, its presence is indicated by the appearance of a noncode output at some system interface. This fault indication can be used in initiate automatic retry or reconfiguration, or to simply stop the system for human intervention and repair.

Originally, the use of self-checking logic circuits was motivated by low component reliability, and research was aimed at design techniques for particular types of circuits, e.g., adders, memories, and counters. With the advent of transistors and integrated circuits, component reliability increased to a point where interest in self-checking declined except for applications requiring ultrahigh reliability. By the late 1960's, however, hardware had become inexpensive enough and computer systems complex enough to again justify self-checking techniques, despite high component reliability. For a review of self-checking techniques prior to 1968, the reader should refer to [1].

In 1968, theoretical work in self-checking circuit design took a new turn, primarily as a consequence of work by Carter and Schneider [2]. Their work went beyond particular circuit designs and pointed toward the study of general models and design methods. It was also proposed that circuits used to check error detecting codes should themselves be self-checking. This work was followed by that of D. A. Anderson [3] which formally defined an important class of self-checking circuits that are known as "totally self-checking." The totally self-checking (TSC) model has led to many significant results in the areas of combinational circuit design [4], [5], sequential circuit design [6]–[8], and check circuit design [9]–[12]. Recently, several researchers have considered the use of TSC circuits in an LSI environment [13]–[15].

As for systems, many early computers had some self-checked parts, for example the Raydac [16], the Univac [17], and the IBM 650 [1]. More recently, the IBM 360 and 370 series computers [18], [19] as well as several other general purpose computers have used self-checking features. The Bell Telephone ESS systems [20] are tailored for telephone switching and achieve self-checked operation by relying heavily on duplication. Other work related to the design of self-checking systems includes the design of self-checking control units [21]–[23], micro- and minicomputers [23]–[26], and a paper design of an IBM 360 [27].

In the area of more general and theoretical research, a limited amount of effort has been directed at totally self-checking system design. In [3] D. A. Anderson proposed some general guidelines for building large combinational subsystems from smaller combinational subsystems. These guidelines were refined in [28]. In [23], many practical considerations of self-checking system design are covered, but there is only a brief discussion of system-level theory.

This paper is concerned with developing a theory of TSC system design. The main objective is to develop a general model for TSC systems that can be used for solving both synthesis and analysis problems.

First, the term TSC as it applies to systems is defined. Note that we often use the terms "system" and "subsystem" interchangeably. This reflects the hierarchical approach we are using where a system at one level may be used as a subsystem when constructing a more complex, higher level system. Our TSC system definition applies equally well to subsystems of any size. Other characteristics of TSC systems are also defined and discussed. These deal primarily with the propagation of errors through a system. A labeled bipartite graph, the "propagation graph" is used to represent this information.

Next a system made of interconnected subsystems is modelled by connecting the propagation graphs of the subsystems to form an "interconnection graph." Sufficient conditions for which a system is TSC are given in terms of the system's interconnection graph.

Several applications of the model are discussed. The first

Manuscript received January 4, 1981; revised February 15, 1983. This work was supported by the National Science Foundation under Grant ENG. 78-05778.

The authors are with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706.

of these is the hierarchical verification of the TSC property. The construction of a propagation graph from a system's interconnection graph is demonstrated. This allows one to use the system as a subsystem in a larger system so that the TSC sufficient conditions can be repeatedly checked in a hierarchical manner.

Then, the problem of the placement of check circuits is discussed. The model is used to minimize the number of check circuits required to make a system TSC (very seldom, if ever, will a system be TSC without some check circuits located at key points within it). Two other applications are then briefly covered, the first is the analysis of a system in order to determine which subsystems may contain contaminated data after an error has been detected. The second is the use of model to gather system diagnostic information following error detection.

II. Basic System Properties

A. Fundamental Definitions

The term "totally self-checking" was first defined for combinational networks [3]. We now give an informal definition extended to include systems. A digital system is *fault-secure* if during normal operation any modeled fault either does not affect the system's output or its presence is indicated no later than when the first erroneous output appears. A digital system is *self-testing* if any modeled fault eventually results in a failure indication during normal system operation. A digital system is *totally self-checking* if it is both self-testing and fault-secure. The fault-secure property is intended to guarantee that any results prior to a failure indication are correct, and the self-testing property is intended to expose all faults so that they do not build up and form a nonmodeled fault.

To more formally define TSC systems, we consider systems to behave as a sequential machine. Of course, system design and analysis methods developed later must depart from classical sequential machine methods in order to be practical, but a definition of TSC in classical terms provides a concise and unambiguous starting point.

Although asynchronous TSC sequential networks have been proposed [7], [8], we choose to consider synchronous ones. Aside from the usual advantages of synchronous logic, checking is easier since it is clear when interfaces should hold code words. In order to deal with clock failures, one can use check circuits for periodic signals [29].

Only the necessary notation and informal versions of definitions and theorems are given in this section. The main results in this section are most easily proved if additional formal notation is developed, and a more rigorous set of definitions are used. To make this section more accessible, however, the additional formalism and proofs of the first three theorems were moved to the Appendix.

We consider a system, S, where X is the set of all possible binary words that can be applied to the inputs of S, that is X is the *input space* of S. Similarly, Y is the *state space* and Z is the *output space* of S.

In fault-free operation, only a subset, A, of inputs in X are actually applied to S. A is the *input code space* of S. $B \subseteq Y$ is the *state code space*, and $C \subseteq Z$ is the *output code space*. In practice, a system or subsystem can receive sets of input lines from various sources and send sets of output lines to various destinations. Accordingly, we partition the input lines of S into r *input signal sets*. This also results in input spaces $X_1, X_2, \cdots, X_r$ and input code spaces $A_1, A_2, \cdots, A_r$; $A_i \subseteq X_i$ for each of the signal sets. Similarly, the outputs are partitioned into t *output signal sets* that result in output spaces $Y_1, Y_2, \cdots, Y_t$ and output code spaces $C_1, C_2, \cdots, C_t$. We do not decompose the state into signal sets, because we never explicitly handle the states. Fig. 1 illustrates the partitioning of inputs and outputs into signal sets.

We next define a model of the operating environment in which a system resides. The is necessary to give meaning to the term "normal operation" used in our informal definitions given earlier. When a system is in a particular state, only certain sequences of inputs can be applied by its environment. The *source model*, R, is a set of *sequence/state pairs*, $\langle \alpha, b \rangle$, where α is a finite sequence of input code words belonging to A that is potentially applicable by the environment when the system is in initial code state b.

Also needed is a description of the failure indication to be used. The use of noncode outputs, as in [3] is quite common. Accordingly, the *sink model*, Σ, identifies output sequences that will be accepted as correct, and includes all sequences where each output signal set contains only code words.

Definition 1: A system S, is *input-output consistent* if every sequence/state pair belonging to the source model causes S to produce an output sequence belonging to the sink model.

We define the *fault model*, F, to be a set of logical faults that modify the logical operation of S. A common fault model is stuck-at faults, although our definition would allow other logical fault models.

Definition 2: A system S is *fault secure* if for every fault in F, and for every sequence/state pair belonging to the source model, the faulty system either produces the correct output sequence, or a sequence not belonging to the sink model.

In terms of time sequences of inputs and outputs, an equivalent of Definition 2 is that the first erroneous output from a fault secure network due to a modeled fault is a noncode output. We will often use this equivalent definition.

The way we have defined a sink model, one can detect an error by checking for code words on each of the output signal sets.

We say $\langle \alpha', b \rangle$ is a *subsequence/state pair* of the sequence/state pair, $\langle \alpha, b \rangle$, if α' is an initial subsequence of α.

Definition 3: A system S is *self-testing* if for every fault in F, any sequence/state pair in the source model either produces an output not in the sink model, or is a subsequence/state pair of a sequence/state pair that does.

Definition 3 in essence, states that after a fault has occurred, any finite input sequence that fails to detect the fault can always be followed by one that does. It is up to the system designer to make sure that the system is exercized so that all faults are eventually detected—the definition is only intended to guarantee that this is possible. This is similar to the case with combinational circuits [3] where it is only required that each fault is detectable by a code word; it is not explicitly stated that the code word is actually applied when the circuit is being used.

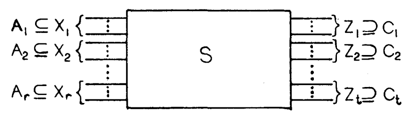


Fig. 1. A system S with r input signal sets and t output signal sets.

Definition 4: A system is *totally self-checking* if it is input-output consistent, fault-secure, and self-testing.

B. Error Propagation

An erroneous input at some signal set is either a noncode word or an erroneous code word. In the remainder of the paper, we use the term "error" or "erroneous input" to include both noncode and erroneous code words. The terms "noncode" and "erroneous code" are used when we choose to be more specific.

When it is part of a self-checking system, one faulty subsystem can cause another (fault-free) subsystem to receive erroneous input sequences. Consequently, it is important that we model the propagation of errors through a subsystem. First, we model the erroneous input sequences a subsystem may actually receive, and assume it begins in a code state.

The *malfunctioning source*, R^M, represents those sequence/state pairs a subsystem observes due to other faulty subsystems. The initial state is a code state and the sequences have values from the input spaces, not just the code spaces. Note that any state after the first may be noncode. In most cases, it is difficult to exactly characterize the errors that can occur, so we typically let R^M be all pairs of all sequences of members of the input space and all code states. Example 4, to be discussed later, indicates some implications of choosing a more exact R^M.

We assume that the synchronous subsystems that make up a system are all clocked by a common system clock. The input and output sequences of all the subsystems consist of binary words that are generated during successive "ticks" of the system clock. The system clock can be used as a discrete time base for measuring the time needed to propagate errors.

Definition 5: For each input signal set i and output signal set j of system S, if two input sequences are applied to S that differ only in signal set i, then the *minimum response time*, δ_{ij}, is the minimum time it takes the change in input signal set i to cause a change in output signal set j. If input signal set i and output signal set j are unrelated, then $\delta_{ij} = \infty$.

The minimum response time can be bounded structurally by counting the minimum number of storage elements (flip-flops) on any path from input signal set i to output signal set j.

Example 1: For the simple buffer register shown in Fig. 2(a), $\delta_{11} = 1$. For the more complex structure in Fig. 2(b), $\delta_{11} = 3$; that is, any change in input signal set 1 take 3 time units to affect output signal set 1. In the same figure, $\delta_{21} = 2$.

We are also interested in whether noncode inputs produce noncode outputs, and in the length of time required for this to take place.

Definition 6: The pair of input and output signal sets i, j is *noncode transmitting* if a change from a code word to a noncode word in a sequence applied at input signal set i always results in a noncode word as the first error in the output sequence at output signal set j; all other input signal sets are assumed to be error-free. The *maximum noncode transmission*

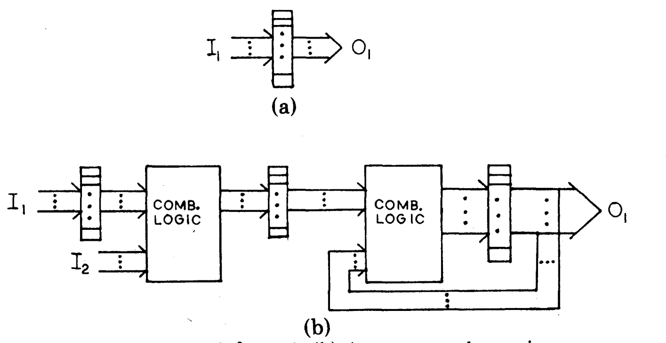


Fig. 2. (a) A register with $\delta_{11} = 1$. (b) A more complex register structure with $\delta_{11} = 3$ and $\delta_{21} = 2$.

time, T_{ij}, is the maximum time it takes for a noncode output to appear i response to a noncode input. If input signal set i and output signal set j are unrelated then $T_{ij} = \infty$.

Definitions 5 and 6 cover the situation where only one input signal set is in error and the others are unchanged. This simplifies determining the values of δ_{ij} and T_{ij}. Because of reconvergent fan out, however, it is possible for more than one input signal set to receive erroneous inputs, possibly at different times. We consider the interaction that occurs when one input signal set receives noncode inputs and others receive erroneous inputs.

First, we expand on Definition 5 by considering a system with two input signal sets receiving errors and one output signal set. The further generalization to arbitrary systems with multiple input and output signal sets is straightforward.

Theorem 1: Consider a system S with two input signal sets, i and k, and output signal set j. A change in the sequences applied at both input signal sets affects the output signal set j no sooner than the minimum time it would take if either input signal set i or k were changed alone.

Proof: The proof appears in the Appendix. □

Next, we expand on Definition 6.

Theorem 2: Let the pair of input and output signal sets i, j be noncode transmitting, and let k be any other input signal set. A change in the input sequence where a noncode word appears on i and any error appears on k results in a noncode word as the first error at j, provided that the time of occurrence of the error on k plus δ_{kj} exceeds the time of occurrence of the error on i plus T_{ij}.

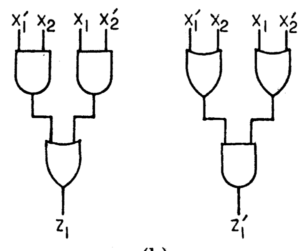
Proof: The proof appears in the Appendix.

Example 2: Consider a two-rail exclusive-OR network. It has two input signal sets, each with code space {01, 10} and one output signal set with code space {01, 10}. A truth table for the network is shown in Fig. 3(a). Fig. 3(b) shows a realization that is TSC. Each input-output pair is noncode transmitting, and since the network is combinational, $T_{11} = T_{21} = \delta_{11} = \delta_{21} = 0$. If a noncode input is applied to either input signal set, then a noncode output results. If both input signal sets are noncode, however, a code output may result (e.g., input 00, 11). The network of Fig. 3(c) also realizes the exclusive-OR, but it always produces a noncode output when either or both inputs are noncode. □

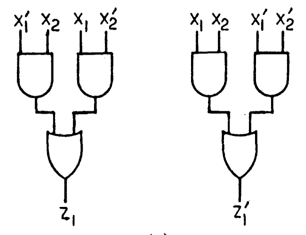
The situation that occurs in the above example is rather comon in combinational networks; that is, where more than one input signal set receives a noncode input at the same time. Unfortunately, if the individual input-output pairs are noncode transmitting, this alone is not enough to guarantee a noncode output as in Fig. 3(c). In order to include this situation (and

x_1	x_1'	x_2	x_2'	z_1	z_1'
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1

(a)



(b)



(c)

Fig. 3. (a) The truth table for a two-rail exclusive-OR network. (b) An implementation that is noncode transmitting but not strongly noncode transmitting. (c) A strongly noncode transmitting implementation.

a similar one for sequential networks) we need the following definition and its accompanying theorem.

Definition 7: The pair of signal sets i, j is *strongly noncode transmitting* if it is noncode transmitting and a noncode input at signal set i always results in a noncode output at signal set j within time T_{ij}; there are no constraints on the initial state of S or input signal sets other than i.

Theorem 3: Let the pair of input and output signal sets i, j be strongly noncode transmitting, and let k be any other input signal set. A change in the input sequence where a noncode word appears on i and any error appears on k results in a noncode word at j as the first error, provided that the time of occurrence of the error on k plus δ_{ij} exceeds, or is equal to, the time of occurrence of the error on i plus T_{ij}.

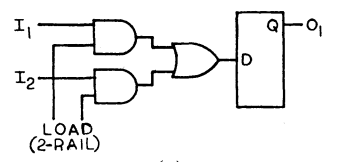
Proof: The proof appears in the Appendix. □

Theorem 3 differs from Theorem 2 only because the phrase "or is equal to" is added to the last sentence.

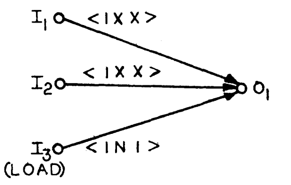
Both input-output pairs in the network of Fig. 3(c) are strongly noncode transmitting. In Fig. 3(b) they are noncode transmitting but not strongly noncode transmitting.

C. The Propagation Graph

The properties and parameters characterizing error propagation through a subsystem can be represented by a labeled bipartite graph called the *propagation graph.* One set of nodes, labeled $I_1, I_2, \cdots, I_r$, represent the r input signal sets, and the other set, labeled $O_1, O_2, \cdots, O_t$, represent the t output signal sets. There is an arc from each input node to each output node. We label the arc from input node I_i to output node O_j with the



(a)



(b)

Fig. 4. (a) A bit-slice of simple TSC subsystem; the inputs and outputs are 1-out-of-n encoded. (b) The subsystem's propagation graph.

triple $\langle \delta_{ij} \begin{pmatrix} N \\ S \\ X \end{pmatrix} T_{ij} \rangle$. δ_{ij} is the minimum response time for the input-output pair i, j. The second component is N if the pair is noncode transmitting but not strongly noncode transmitting, S if the pair is strongly noncode transmitting, and X if it is neither. T_{ij} is the maximum noncode transmission time; this component is an X if the second component is an X.

Example 3: Consider a register that always holds a 1-out-of-n code word and is loaded from one of two input signal sets depending on a two-rail encoded load signal. Such a register might appear as part of the control structure of a TSC system. A source assumption is that the two 1-out-of-n input signal sets are never the same code word at the same time. A typical bit slice of this structure is shown in Fig. 4(a). In the propagation graph, Fig. 4(b), there are three input nodes (two data and one control) and one output node. The 1-out-of-n inputs are not noncode transmitting because the first noncode word may be ignored if the LOAD signal selects the other input. The LOAD input is noncode transmitting since a 00 or 11 yields a noncode output during the next clock period (so $T_{31} = 1$). It is not strongly noncode transmitting because it is possible in the 11 case to have errors on other inputs that produce an error output that may be a code word. □

In larger subsystems some output signal set may be independent of some input signal set. In such a case, the triple $\langle \infty S \infty \rangle$ labels the corresponding arc in the internal propagation graph. For convenience, we simply delete the arc in this case. Hence, if the graph is incomplete, all deleted arcs are understood to be labeled $\langle \infty S \infty \rangle$.

III. A Structural Interconnection Model

Thus far we have defined a subsystem model along with parameters that characterize the propagation of errors through it. We now define a graph model for studying the propagation of errors through an interconnection of subsystems.

Fig. 5(a) shows the structure of an example system. The subsystems are labeled S_i and may be either combinational or sequential. In order to model the propagation of errors

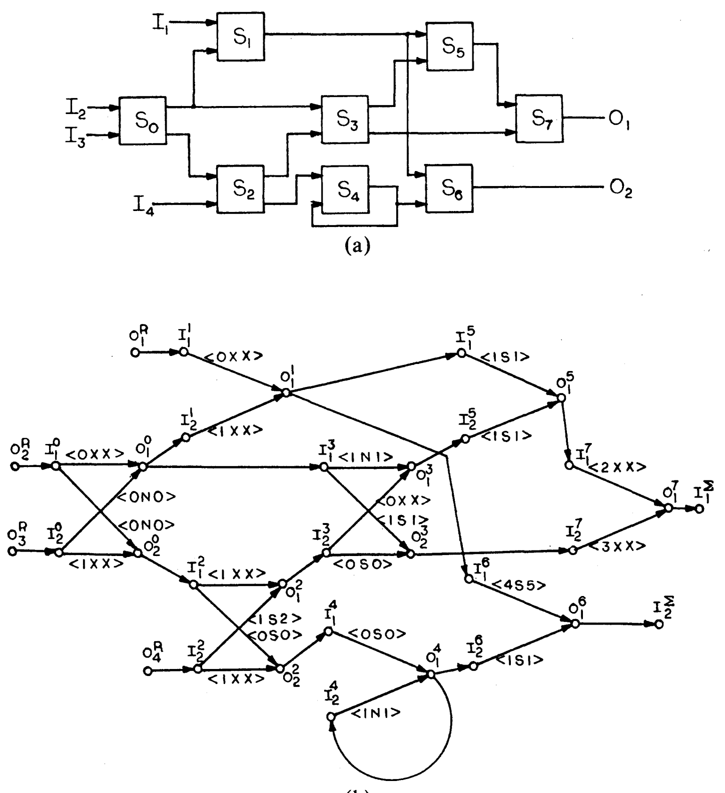


Fig. 5. (a) An example system formed as an interconnection of subsystems. (b) The system's interconnection graph.

through such a system, we begin with the propagation graphs of each of the subsystems. We add superscripts to the input and output nodes to distinguish them. That is, input signal set j of subsystem S_i is labeled I_j^i, and output signal set j is labeled O_j^i. Then we place a directed arc from O_j^i to I_l^k if output signal set j of subsystem i is connected to input signal set l of subsystem k. Such an arc is called a *connection arc*. The original labelled arcs of the internal propagation graphs are also present, and we refer to them as *internal arcs*.

To model the source and sink, we add two sets of nodes. The source has only output nodes that are labelled O_j^R and are connected to subsystem input nodes. The sink has only input nodes that are labeled I_j^{Σ}, and subsystem output nodes are connected to them. Fig. 5(b) shows the interconnection graph for the network of Fig. 5(a).

A. Paths and Path Properties

A *path* in an interconnection graph is denoted as a sequence of nodes where there is an arc connecting adjacent nodes in the sequence.

Definition 8: A path $(O_j^i I_l^k O_m^k, \cdots, I_q^p)$ is a *direct path* issued from output O_j^i if no node in the path except the first is in subsystem Si and no node is repeated.

In Fig. 5, a path is $(O_2^2 I_1^4 O_1^4 I_2^4 O_1^4 I_2^6 O_1^6 I_2^{\Sigma})$ but it is not a direct path. A direct path is $(O_2^2 I_1^4 O_1^4 I_2^6 O_1^6 I_2^{\Sigma})$.

In our discussion, a direct path will typically begin at an output of a faulty subsystem. We will be interested in three types of direct paths that can issue from a faulty subsystem:

1) N-paths (for noncode), that can potentially transmit noncode words to system outputs.

2) B-paths (for blocking), that can block the propagation of a noncode word along an N-path by producing an erroneous code word instead, and

3) E-paths (for error), that can transmit erroneous code words to system outputs.

Definition 9: A direct path is an *N-path* if the last node belongs to the sink (is I_q^{Σ} for some q) and every internal arc is noncode transmitting. A direct path is a *partial N-path* if every internal arc is noncode transmitting.

In Fig. 5(b), an N-path is $(O_2^0 I_1^2 O_2^2 I_1^4 O_1^4 I_2^6 O_1^6 I_2^{\Sigma})$ a partial N-path is $(O_2^0 I_1^2 O_2^2 I_1^4)$.

An N-path passes through a sequence of noncode transmitting subsystems and can potentially transmit a noncode word from the subsystem output to the system output. The time it requires to do this is critical because erroneous code words may also be propagating toward system outputs along different paths.

Definition 10: The *N-path propagation time*, N_p, of the N-path $P = (O_j^i, \cdots, I_l^{\Sigma})$ is the sum of the maximum noncode transmission times T_{mn} of all internal arcs $(I_m^k O_n^k)$ in P.

Definition 11: Given an N-path or partial N-path $P = (O_j^i \cdots I_n^m, \cdots, I_l^{\Sigma})$, a direct path $Q = (O_k^i, \cdots, I_n^m)$ is a *B-path with respect* to P if it intersects the N-path and begins at the same subsystem as the N-path, but not necessarily at the same output. If it does begin at the same output then the B-path never reenters the initial subsystem.

A B-path may intersect an N-path and errors on both paths (possibly both noncode) may conspire in such a way that an erroneous code word is produced.

Referring to Fig. 5(b), with respect to $P = (O_1^0 I_1^3 O_1^3 I_2^5 O_1^5$

I_1^7) a B-path according to 1) of Definition 11 is ($O_2^0 I_1^2 O_1^2 I_2^3 O_1^3 I_2^5$), and according to 2) a B-path is ($O_1^0 I_2^1 O_1^1 I_1^5 O_1^5 I_1^7$).

Definition 12: The *B-path propagation time*, B_p, of path $P = (O_j^i, \cdots, I_k^h)$ is the sum of the minimum response times δ_{mn} of all the arcs $(I_m^l O_n^l)$ in P.

Definition 13: Let $P = (O_j^i, \cdots, I_l^\Sigma)$ be an N-path. Let Q be any B-path with respect to P, and let P' be a partial N-path of P that terminates at the same node as Q. Then P is a *valid propagating path* if for every such Q, $N_{p'} \leq B_Q$, and if $N_{p'} = B_Q$ then the last internal arc in P' is strongly noncode transmitting.

Informally, at a node in a valid propagating path where an N-path and a B-path intersect, a noncode word must be produced before errors on both paths can possibly conspire to produce an erroneous code word. This is used in the proof of Theorem 4.

In Fig. 5(b), $P = (O_2^0 I_1^2 O_1^4 O_1^4 I_2^6 O_1^6 I_2^\Sigma)$ is an N-path with $N_p = 1$. The only B-path with respect to P is $Q = (O_1^0 I_2^1 O_1^1 I_1^6 O_1^6 I_2^\Sigma)$ and $B_Q = 5$. Therefore, P is a valid propagating path.

Definition 14: Given a valid propagating path $P = (O_j^i, \cdots, I_l^\Sigma)$, a direct path $Q = (O_k^i \cdots I_m^\Sigma)$, $m \neq l$, is an *E-path* with respect to P if either

1) $k \neq j$ or
2) $k = j$ and Q is not a valid propagating path.

Referring to Fig. 5(b), with respect to the valid propagating path P given above, $Q = (O_2^0 I_1^2 O_1^2 I_2^3 O_1^3 I_2^5 O_1^5 I_1^7 O_1^7 I_1^\Sigma)$ is an E-path according to 2) in Definition 14.

An E-path provides a means for propagating an incorrect code word to a system output, and the time required to do so is another critical parameter.

Definition 15: The *E-path propagation time*, E_Q, of the E-path Q is the sum of the minimum response times δ_{mn} of all internal arcs $(I_m^k O_n^k)$ in Q.

For the E-path Q given above, $E_Q = 4$.

B. Sufficient Conditions for TSC Systems

We are now ready to given sufficient conditions for which a system is TSC. We do this by examining the fault-secure and self-testing properties separately. First, we discuss the source a subsystem actually sees when it is embedded in a system.

Say subsystem S_i is designed to be TSC for source R_i, and S_i is embedded in a system. Then, subsystems separating S_i from the system source R result in an *observed source* R_i' that S_i actually observes during normal system operation. In general, $R_i \neq R_i'$, and we consider three cases in order to determine the effect on the consistency, self-testing, and fault-secure properties in S_i.

First, say $R_i' \supseteq R_i$. Then, consistency may not hold; that is, some noncode word could conceivably be produced by S_i during normal operation. Hence, consistency would need to be reverified. Also, the fault-secure property may not hold because there may be some sequence/state pair in $R_i' - R_i$ that produces an incorrect code word as the first incorrect output due to a fault. Hence, the fault-secure property would need to be verified. Finally, although it is less obvious, the self-testing property might also fail to hold. This is because a sequence/state pair in $R_i' - R_i$ might put S_i in a state from which a fault cannot be detected. Thus, the self-testing property, in general, also needs to be reverified; an exception is if S_i is combinational.

If $R_i' \not\supseteq R_i$ and $R_i \not\supseteq R_i'$ then there are state sequence pairs in R_i' that are not in R_i and, as before, all three TSC properties need to be reverified for R_i'.

If $R_i \supseteq R_i'$ then it is clear that the consistency property and the fault-secure property must hold with respect to R_i'. The self-testing may not hold, since S_i may not be exercised thoroughly enough; hence, the self-testing property must be reverified for R_i'. We define subsystem S_i in a system to be *sufficiently exercised* if it is self-testing with respect to its observed source R_i'.

Fortunately, in most cases a subsystem does not see input sequences it is not designed for, although it might not see all the sequences it is designed for. Consequently, in the remainder of the paper we assume that for all subsystems $R_i \supseteq R_i'$, and it is only necessary to show that each subsystem is sufficiently exercised to verify that the subsystem is self-testing in its particular environment. To do this one could, at least in theory, begin with the system source R and derive R_i'; via simulation. In practice, this process may be very time-consuming, and one may instead resort to ad hoc methods for demonstrating that subsystems are sufficiently exercised.

We now consider sufficient conditions for a system to be fault-secure.

Theorem 4: Given a system S composed of subsystems S_i, which are fault-secure with respect to fault sets F_i. S is fault secure with respect to $\cup_i F_i$ if for each subsystem output node O_j^i there is a valid propagating path, P, beginning at O_j^i such that $N_p \leq E_Q$ for every E-path, Q, with respect to P.

Proof: The proof is rather lengthy and can be found in [30]. We only outline the proof here.

First, we assume an arbitrary subsystem S_i is faulty, and let t be the time of the first erroneous output on any of its output signal sets. If $t = \infty$, then the system is trivially fault secure for the fault in question.

Otherwise, the existence of P guarantees a noncode word will be propagated as the first error to a system output provided that all subsystems along P receive only correct code words on input signal sets not on P. This is the case unless there is some other path from S_i to another subsystem on P. The set of B-paths represents each of these reconverging paths that intersect the valid propagating path.

By considering only B-paths, we neglect reconverging paths that are not direct, i.e., those that contain cycles or pass through the subsystem that begins the path. This is because any nondirect path that ends at a node must be "longer" than some B-path that ends at the same node, where "length" is the sum of the δ_{ij} on the path.

The output nodes along P are then ordered and an inductive argument is used to show that if a noncode word is propagated as far as the nth output node, then it will be propagated as the first error to the $n + 1$st. This terminates at a system output.

In the process, it is also shown that the noncode propagation can take no longer than N_p. In order to prove the fault-secure property, it is then shown that no erroneous code output reaches a different system output signal set any sooner than N_p. The only direct paths that can produce an output error are E-paths and by repeatedly using Theorem 1 along an E-path, the earliest an error can be produced at a system output is E_Q. The condition that $N_p \leq E_Q$ guarantees that at the time the first error reaches a system output there is a noncode word at some output signal set. □

As for the self-testing property, it appears that the first noncode output of a faulty subsystem must be propagated as a noncode word to a system output if a valid propagating path exists. A point that is easily overlooked is that the error propagation properties (Definitions 5 and 6) are defined for the malfunctioning source R^M where the initial states are code states, i.e., they appear in normal (fault-free) operation.

By Definition 6, a noncode transmitting input-output pair is guaranteed to propagate a noncode word *only if* it is initially in a code state. Consequently, if an erroneous code input puts it in a noncode state it is no longer guaranteed to transmit later noncode words. It is easily shown that a valid propagating path is guaranteed to propagate a noncode word when the subsystem at the beginning of the valid propagating path is fault-secure. If the initial subsystem is just known to be self-testing, however, this may not be the case; because even if a noncode output can eventually be produced, any preceding erroneous code outputs can invalidate all propagating paths.

In order to verify the self-testing property in a system separately from the fault secure property, it would be necessary to begin with a malfunctioning source $R^{M'}$ where the initial states are members of Y and to reformulate Definitions 5 and 6. Since we are really interested in the TSC property, this is not a major obstacle, and it does not prevent us from deriving sufficient condition for the TSC property.

Theorem 5: A system S defined as in Theorem 4 is TSC with respect to $\cup_i F_i$ if

1) each subsystem S_i is TSC with respect to F_i;
2) each subsystem S_i is sufficiently exercised, and
3) the system S is fault secure by Theorem 4.

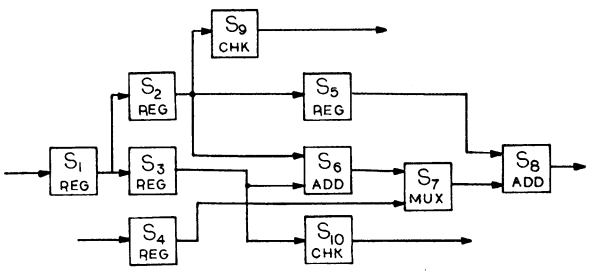
Proof: Input-output consistency follows from input-output consistency of the subsystems and the fact that $R_i' \subseteq R_i$.

The fault secure property is given.

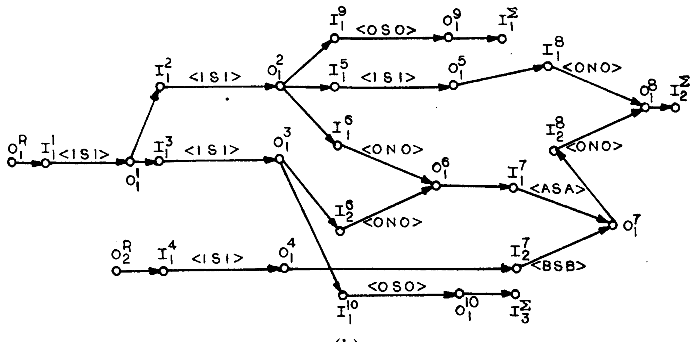
Because each subsystem S_i is sufficiently exercised each fault can be made to result in a noncode word as the first error at an output of S_i. Then the valid propagating path of Theorem 4 guarantees that a noncode word is propagated to an output of S. □

Example 4: Consider the system in Fig. 6 that is similar to one given in [23] with the checker placement slightly different. Control inputs are not shown, and are assumed to be checked elsewhere. The propagation graphs for the basic components are shown in Fig. 7. For the multiplexer, neither arc is noncode transmitting because the first noncode word at a multiplexer input may not be selected. Then, we hve no guarantee that an erroneous code input does not follow the noncode input and is selected. This was assumed in Example 3.

However, if we examine the environment of the multiplexer in Fig. 6, it happens that due to a fault "upstream" from the multiplexer all multiplexer inputs are either correct or noncode.



(a)



(b)

Fig. 6. (a) A system adapted from Fig. 4.1 of [23]. (b) The system's interconnection graph.

Using a more exact R^M based on this observation, we see that following a noncode input word the first erroneous output will be noncode (possibly caused by a different input) although we cannot say how long it will take. We do observe that the first error propagated and the first noncode word propagated require the same time, however. Consequently, we assume $T_{11} = \delta_{11} = A$ and $T_{21} = \delta_{21} = B$ where A and B are finite but unbounded. Then the multiplexer can be modeled by labeling its arcs $\langle A\,S\,A\rangle$ and $\langle B\,S\,B\rangle$.

We note that the "t-detection lossless" concept as used [23] always assumes that erroneous input streams contain only noncode and correct code words. In the general case, however, an incorrect code word can appear after a noncode word.

To continue our example, connecting the internal propagation graphs and adding source and sink nodes yields the system interconnection graph shown in Fig. 6(b). The subsystem output node O_1^1 has N-paths

$$P_1 = (O_1^1\, I_1^3\, O_1^3\, I_2^6\, O_1^6\, I_1^7\, O_1^7\, I_2^8\, O_1^8\, I_2^{\Sigma}),$$

$$P_2 = (O_1^1\, I_1^3\, O_1^3\, I_1^{10}\, O_1^{10}\, I_3^{\Sigma}),$$

$$P_3 = (O_1^1\, I_1^2\, O_1^2\, I_1^6\, O_1^6\, I_1^7\, O_1^7\, I_2^8\, O_1^8\, I_2^{\Sigma}),$$

$$P_4 = (O_1^1\, I_1^2\, O_1^2\, I_1^9\, O_1^9\, I_1^{\Sigma}), \text{ and}$$

$$P_5 = (O_1^1\, I_1^2\, O_1^2\, I_1^5\, O_1^5\, I_1^8\, O_1^8\, I_2^{\Sigma}).$$

$P_6 = (O_1^1\, I_1^2\, O_1^2\, I_1^6\, O_1^6\, I_1^7)$ is a B-path with respect to P_1 and $B_{P_6} = N_{P_1'}$ ($P_1' = (O_1^1\, I_1^3\, O_1^3\, I_2^6\, O_1^6\, I_1^7)$), but $(I_2^6\, O_1^6)$ is not strongly noncode transmitting, so P_1 is not a valid propagating path. P_2 and P_4 do not have any B-paths and, therefore, are valid propagating paths. The paths P_1 and P_3 are E-paths with E_{P_1} and $E_{P_3} \geq 1$ depending on A and B, but they may equal 1 in the worst case. $P_{P_4} = N_{P_2} = 1$ so the condition of Theorem 4 holds for O_1^1.

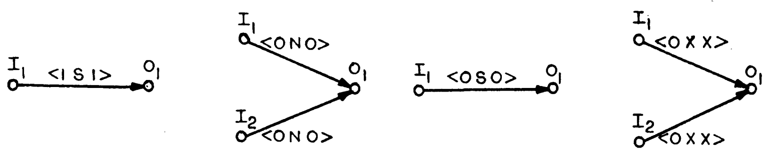


Fig. 7. Propagation graphs for some basic TSC components. (a) register. (b) adder. (c) checker. (d) multiplexer (data paths only).

Valid propagating paths that satisfy the theorem for the other I_j^i are

$$O_1^2 : (O_1^2\, I_1^9\, O_1^9\, I_1^\Sigma)$$
$$O_1^3 : (O_1^3\, I_1^{10}\, O_1^{10}\, I_3^\Sigma)$$
$$O_1^4 : (O_1^4\, I_2^7\, O_1^7\, I_2^8\, O_1^8\, I_2^\Sigma)$$
$$O_1^5 : (O_1^5\, I_1^8\, O_1^8\, I_2^\Sigma)$$
$$O_1^6 : (O_1^6\, I_1^7\, O_1^7\, I_2^8\, O_1^8\, I_2^\Sigma)$$
$$O_1^7 : (O_1^7\, I_2^8\, O_1^8\, I_2^\Sigma)$$
$$O_1^8 : (O_1^8\, I_2^\Sigma),\ O_1^9 : (O_1^9\, I_1^\Sigma),\ O_1^{10} : (O_1^{10}\, I_3^\Sigma).$$

All of them satisfy the condition of Theorem 4 with respect to E-paths, so the system is fault-secure. If each subsystem is self-testing and sufficiently exercised, then the system is TSC by Theorem 5. □

IV. Applications

A. System Design

A major application of the system model just developed has been suggested in the previous section—verification that a system is TSC. A possible difficulty is that if the sufficient conditions of Theorems 4 and 5 are applied to a large system made of many subsystems, the number of possible N-paths, E-paths, and B-paths that need to be checked may be excessive. A solution is to decompose the system into subsystems and to check each of the subsystems against the sufficient conditions. Then, after they are verified to be TSC, they can be composed for form larger subsystems that can be checked against the conditions. This process can continue in a hierarchical manner until the entire system is checked.

In order to facilitate this hierarchical verification process, we give methods for constructing the propagation graph of a TSC system or subsystem, given its interconnection graph.

Say the given interconnection graph has a source with output nodes $O_1^R\, O_2^R, \cdots, O_t^R$ and a sink with input nodes $I_1^\Sigma\, I_2^\Sigma \cdots I_r^\Sigma$. These are connected to certain subsystem inputs and outputs, respectively. We form a propagation graph with input nodes $I_1, I_2, \cdots, I_t$ corresponding to the nodes $O_1^R, O_2^R, \cdots, O_t^R$ and with output nodes $O_1, O_2, \cdots, O_t$ corresponding to nodes $I_1^\Sigma, I_2^\Sigma, \cdots, I_r^\Sigma$ in the interconnection graph. Arcs are drawn connecting all the I_i to the O_j (unless there is no path from O_i^R to I_j^Σ in the interconnection graph; in this case we delete the arc following our earlier convention).

Next, the arcs need to be labeled with the triples

$$\langle \delta_{ij} \begin{pmatrix} S \\ N \\ X \end{pmatrix} T_{ij} \rangle.$$

The δ_{ij} are found by taking the least sum of the δ's along paths from O_i^R to I_j^Σ. It should be clear that this actually lower bounds the minimum response time, but as such it can still be used for δ_{ij} without affecting Theorem 4 or 5.

To determine if an input-output pair i, j is noncode transmitting, we determine whether there is a valid propagating path beginning at O_i^R and ending at I_j^Σ. Any E-paths or B-paths that begin at O_k^R, $k \neq i$, are neglected since the noncode transmitting property is determined with only the input signal set i being erroneous. If there is such a valid propagating path, the least sum of the noncode transmission times along all valid propagating paths is T_{ij}. To check the strongly noncode transmitting property, we must verify that the shortest valid propagating path (or one of them if there is more than one) propagates noncode words within time T_{ij}, regardless of input sequence and initial state. It can be shown that a sufficient condition for a valid propagating path to be strongly noncode transmitting is that each internal arc on the path is strongly noncode transmitting.

Example 5: We now design a simple microprogrammed control unit and verify it to be TSC in a hierarchical manner. For more detail, refer to [30].

A microinstruction has a format as shown below.

N	C	D	CONTROL SIGNALS

N: specifies source of next microinstruction address. It consists of 2 bits and indicates that the next address is derived from

1) incrementing the current address by 1 or by 2 depending on whether or not a skip condition is satisfied,

2) the opcode in the macroinstruction register (actually the opcode mapped through a PLA),

3) the D-field of the microinstruction (i.e., a jump).

C: specifies conditions for a conditional skip.

D: contains data and is used for unconditional jumps, among other things.

Control Signals: All the various control signals that go into the data structure and enable registers, control ALU's, etc. These are divided into subfields in some way which is not of particular interest here.

We assume the controller is initially designed in a top-down manner. Fig. 8 shows the controller broken into three subsystems and Fig. 9 shows subsystem S_1 decomposed into even simpler components. Diagrams of S_2 and S_3 are in [30]. Verification of the TSC property will be done in a bottom-up manner. We begin with the propagation graphs of the basic components as shown in Fig. 7. These are initially designed to be TSC, and their error propagation characteristics are determined. These propagation graphs are used to form the interconnection graph for the subsystem. The interconnection graph for S_1 is shown in Fig. 10, and it is necessary to add a TSC check circuit at the PLA output to make S_1 TSC (refer

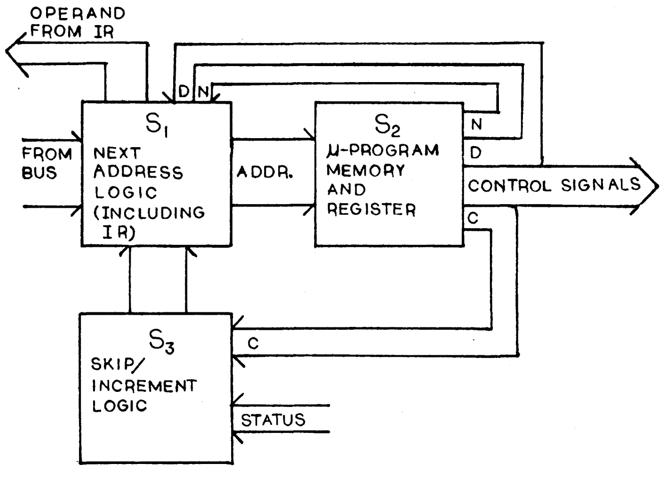


Fig. 8. The three major subsystems of a microprogrammed controller.

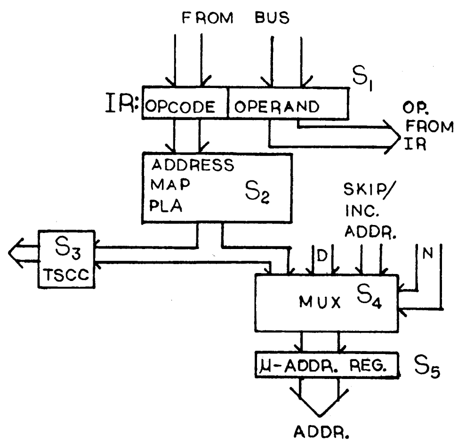


Fig. 9. The subsystem S_1 from Fig. 8 decomposed into smaller subsystems.

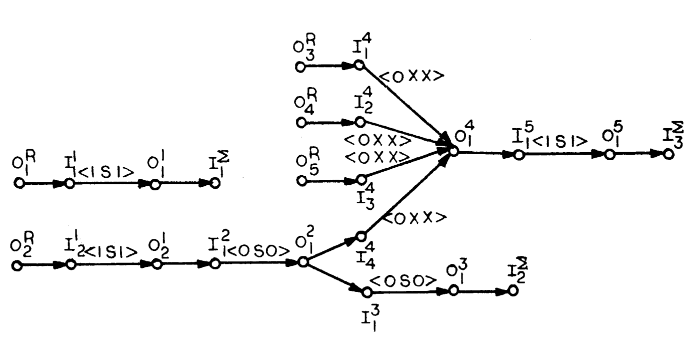


Fig. 10. The interconnection graph for the subsystem shown in Fig. 9 (the decomposition of S_1 from Fig. 8).

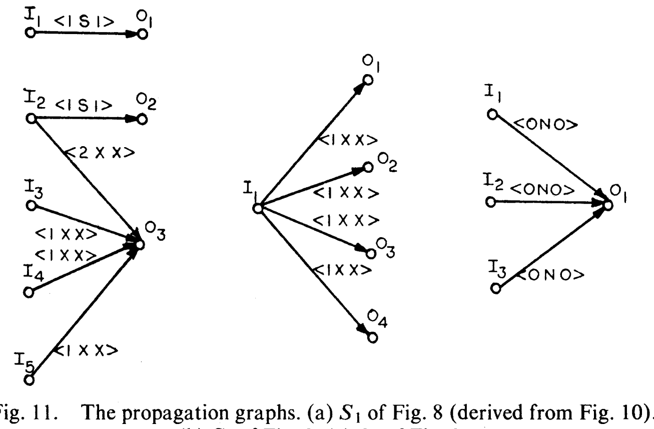


Fig. 11. The propagation graphs. (a) S_1 of Fig. 8 (derived from Fig. 10). (b) S_2 of Fig. 8. (c) S_3 of Fig. 8.

to the next section for checker placement). Then, the propagation graphs for the subsystems are generated. The graph for S_1 is in Fig. 11(a) with the graphs for S_2 and S_3 in Fig. 11(b) and 11(c). These propagation graphs are combined to form the interconnection graph for the entire controller [Fig. 12(a)]. At this point, it is determined that more TSC checkers are needed at some controller outputs (shown with dashed lines). Finally, the propagation graph for the entire controller can be generated [Fig. 12(b)]. This graph can now be used at still higher levels in the system of which the controller is a part.

B. Checker Placement

A TSC check circuit [9] is a combinational circuit that monitors an interface for noncode words, and when one appears its presence is immediately signaled by the checker. The propagation graph for a TSC checker is shown in Fig. 7(c).

By attaching a TSC checker to an output node and letting the checker output feed the sink (i.e., it is a system output), a valid propagating path from the output node satisfying the condition of Theorem 4 is guaranteed. In terms of the model, this is because there are no B-paths with respect to the N-path that passes through the checker, and the "length" (N_p) of the N-path is 0. Consequently, by adding TSC checkers any output node can be made to satisfy Theorem 4. Of course, adding a TSC checker might also create a valid propagating path for other output nodes besides the one to which the checker is attached.

In this section, we study the problem of placing checkers so that a system that does not satisfy Theorem 4 is converted to one that does. While this checker placement problem could be solved on a purely ad hoc basis, we derive a method that adds the least number of checkers needed. Not surprisingly, this turns out to be a covering problem and as such has NP-complexity. Nevertheless, it may be of practical use in situations where there is a small number of subsystems, and it can be used as a starting point for deriving near-minimal procedures of lower complexity.

We consider partial N-paths and their corresponding B-paths and E-paths. A single connecting arc is considered here to be a degenerate partial N-path.

Definition 16: A partial N-path $P = (O_j^i, \cdots, I_l^k)$ is a *partial valid propagating path* if it satisfies the condition of Definition 13.

It is easy to see that any initial portion $Q = (O_j^i \cdots I_n^m)$ of a partial valid propagating path $P = (O_j^i, \cdots, I_n^m, \cdots, I_l^k)$ is also a partial valid propagating path because all B-paths with respect to Q are also B-paths with respect to P.

We now come to what could be termed the "checker placement theorem."

Theorem 6: Given a partial valid propagating path $P = (O_j^i, \cdots, O_n^m I_l^k)$ in a system's interconnection graph, the nodes I_1^c, O_1^c, I_p^Σ and arcs $(O_n^m I_1^c)$, $(I_1^c O_1^c)$, $(O_1^c I_p^\Sigma)$ are added. The internal arc $(I_1^c O_1^c)$ is labeled $\langle O\ S\ O\rangle$ (i.e., we have added a TSC checker). Then $P' = (O_j^i \cdots O_n^m I_1^c O_1^c I_p^\Sigma)$ is a valid propagating path.

Proof: P is an N-path and since I_1^c and I_p^Σ have in-degree 1, P' has only the B-paths of P. Since P satisfies the condition of Definition 13 so must P'. □

Theorem 6 says that adding a TSC checker to any output

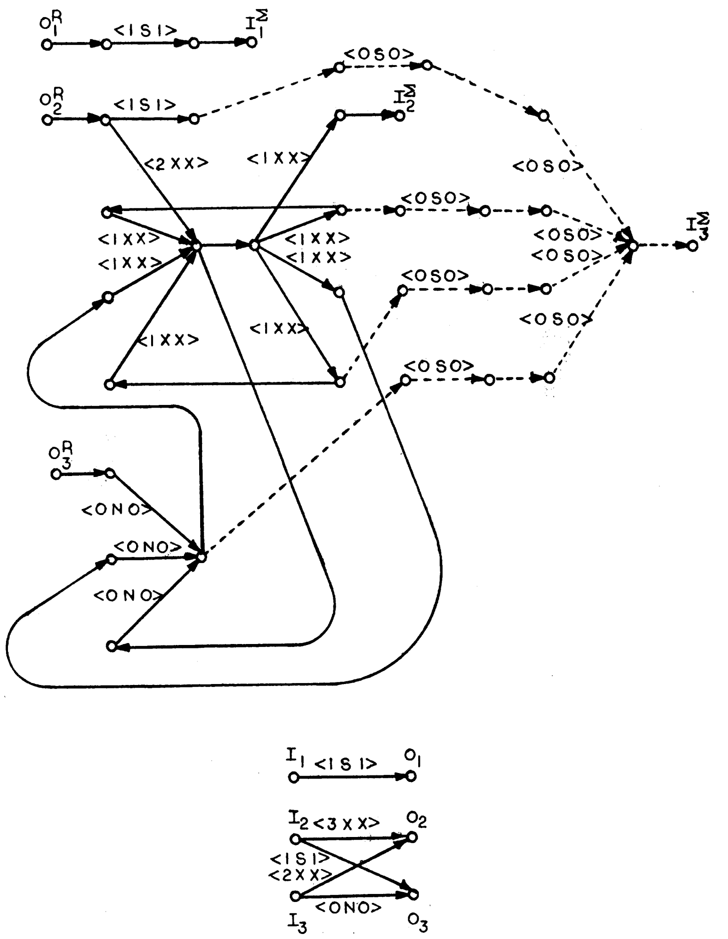


Fig. 12. (a) Interconnection graph for the system shown in Fig. 8; dashed lines indicate added checkers. (b) The system's propagation graph.

node in a partial valid propagating path from O_j^i yields a complete valid propagating path from O_j^i.

To optimally place checkers, we find every output node that does not have a valid propagating path satisfying Theorem 4. We then find all partial valid propagating paths, P, that begin at such an output node *and* that satisfy the condition of Theorem 4. That is, for every E-path, Q, with respect to P, $N_p \leq E_q$. Adding a checker to any of the output nodes in P yields a valid propagating path (Theorem 6) that also satisfies the condition of Theorem 4. Hence, for each output node O_j^i needing a valid propagating path we form a list L_{ij} containing all the output nodes in partial valid propagating paths from O_j^i and which satisfy the $N_p \leq E_Q$ condition. Then, we find a set of output nodes that covers all the L_{ij} lists. That is, the covering set contains at least one output node appearing in each list. Putting a checker at each output node in such a covering set satisfies Theorem 4 for the system, and finding a minimal covering set yields a minimum number of added checkers.

Example 5: Consider the system with the interconnection graph shown in Fig. 5. Output O_2^2 has the valid propagating path $(O_2^2\ I_1^4\ O_1^4\ I_2^6\ O_1^6\ I_2^\Sigma)$ and the path satisfies Theorem 4. Hence, no checker needs to be added to detect errors originating at O_2^2. On the other hand, O_1^0 has no valid propagating path. It does have the partial valid propagating $(O_1^0\ I_1^3\ O_2^3\ I_2^7)$. Then, the list of possible nodes for attaching a checker to detect errors originating at O_1^0 is $L_{01} = O_1^0, O_2^3$.

Similarly, the other lists L_{ij} that need to be covered are

$$L_{11} = O_1^1, O_1^5$$

$$L_{21} = O_1^2, O_2^3$$

$$L_{31} = O_1^3, O_1^5$$

$$L_{32} = O_2^3$$

$$L_{51} = O_1^5$$

A minimal cover is $\{O_2^3, O_1^5\}$ so adding two TSC checkers to output nodes O_2^3 and O_1^5 makes the system fault secure (Theorem 4) and TSC provided the self-testing and sufficiently exercized conditions are met (Theorem 5).

It is interesting to note that nowhere in the cycle $(O_1^4\ I_2^4\ O_1^4)$ is a TSC checker added. This is contrary to checker placement guidelines proposed in [23]. □

C. Data Contamination Analysis

Since the system interconnection graph models the propagation of errors, we can also use it to determine which subsystems may contain "contaminated" data, i.e., have received erroneous inputs. There are several data contamination analysis problems one could consider, and we choose one that is typical.

Say a noncode word has been detected at a system output, the system has been halted, and by using systems diagnosis

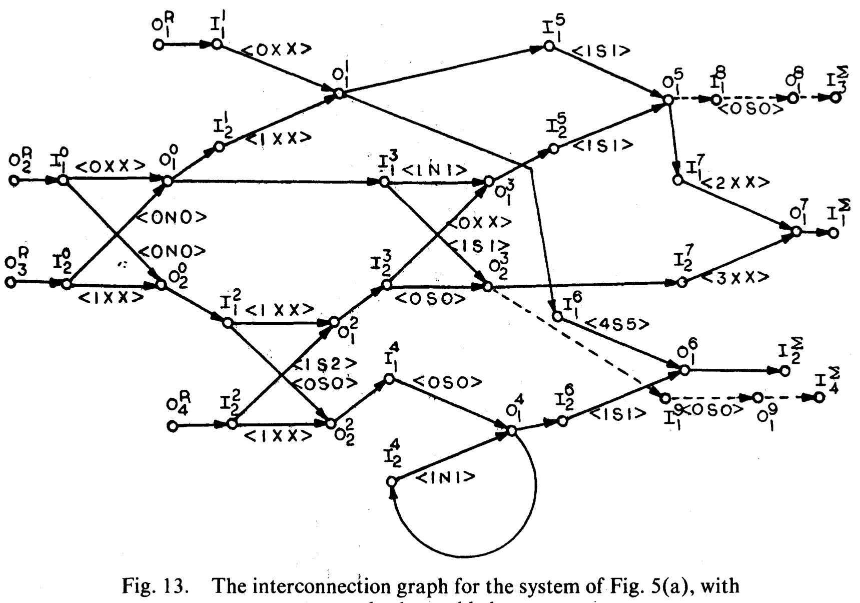


Fig. 13. The interconnection graph for the system of Fig. 5(a), with checkers added.

techniques (possibly including checker information) subsystem S_i is determined to be faulty. Then one might want to know what other subsystems have received erroneous inputs and are contaminated.

Say sink signal sets $I_i^\Sigma, I_j^\Sigma, \cdots, I_k^\Sigma$ all contained noncode words when the system was halted, i.e., these noncode words signaled the fault. Then, we find the valid propagating path P from S_i to any of $I_i^\Sigma, I_j^\Sigma, \cdots, I_k^\Sigma$ with the least N_p. This is the longest time errors could have to propagate from S_i. Then by using the δ's, any subsystem that could have received an error input can then be determined.

Example 6: Consider the system modeled in Fig. 13 (This is the system of Fig. 5 with added checkers). Say a noncode word at the checker output feeding I_4^Σ signals a fault. The system is stopped and S_0 is diagnosed as faulty. $P = (O_1^0\, I_1^3\, O_2^3\, I_1^9\, O_1^9\, I_4^\Sigma)$ is the only valid propagating path connecting O_1^0 and I_4^Σ, and $N_p = 1$. This means errors could have propagated for at most one clock period following the first error at the output of S_0. This means S_1, S_2, S_3 and S_4 could have become contaminated since errors can propagate to them in one clock period. □

D. Fault Diagnosis

As alluded to in the previous section, noncode outputs can provide some diagnostic information, i.e., which subsystem is faulty. Using noncode outputs alone, however, usually leads to poor diagnostic resolution if the system is designed with only fault detection in mind. Nevertheless, they can be used to provide some useful diagnostic information that may increase the overall resolution when used with other diagnostic techniques. The following example indicates a way to extract diagnostic information from the system interconnection graph.

Example 7: Again, consider the system of Fig. 13. Say a noncode output is received at I_3^Σ as the first error indication. By simply tracing paths to see what subsystems are on paths to I_3^Σ, we narrow down potentially faulty subsystems to S_0, S_1, S_2, S_3, S_5, and S_8. However, a noncode output from O_1^0 would reach I_4^Σ before I_3^Σ. Similarly a noncode output from O_2^0 would reach I_2^Σ before I_3^Σ. Therefore S_0 can be ruled out as the faulty subsystem. S_2 can also be ruled out since any noncode outputs would reach I_4^Σ before I_3^Σ. Therefore, only subsystems S_1, S_3, S_5 and S_8 are potentially faulty.

V. Summary and Conclusions

It was the purpose of this paper to develop a formal model for totally self-checking systems and to suggest some potential applications. Totally self-checking systems were first formally defined. This definition is rather straightforward, although the self-testing part of the definition differs from what is customarily used. Informally, it states that a sequential network or system is self-testing if after the fault occurs and before fault detection it is always in a state (normal or not) from which a normal input sequence can potentially detect the fault.

When totally self-checking systems are being studied the propagation of errors, both code and noncode is of utmost concern, so following the definition of the totally self-checking property, other properties related to error propagation were discussed. These essentially characterize the minimum time an error needs to pass from an input to an output, and the maximum time a noncode word takes to be passed from an input to an output. These properties were defined by considering a single input signal set and a single output signal set, while leaving all other input signal sets correct. In this way, determination of the propagation parameters is simplified, although "interference" that might take place due to multiple errors is ignored. This difficulty was rectified, however, by Theorems 1 and 2 which place a limit on the effects that multiple input errors can have. Then the error propagation characteristics of a subsystem were modeled by the propagation graph.

The propagation graph essentially models the local behavior of individual subsystems that make up a system. Global behavior is modeled by connecting the propagation graphs of the subsystems to form the system's interconnection graph. Three types of paths in a system were then defined and discussed. An

N-path is one that can potentially propagate noncode words to a system output. A B-path for a given N-path begins at the same subsystem as the N-path and later reconverges with the N-path. It may under certain circumstances block the propagation along an N-path by introducing multiple errors at a subsystem input. If an N-path is not blocked, it is said to be a "valid propagating path." An E-path is a path that can potentially propagate an incorrect code word form a faulty subsystem to a system output.

Sufficient conditions under which a system is totally self-checking were given. The individual subsystems must each be totally self-checking and "sufficiently exercised." In addition, from each subsystem output there must be a valid propagating path that produces a noncode system output before any E-path can produce an erroneous code output.

Applications of the system model were then discussed. The first is verification of the totally self-checking property in a system. A hierarchical approach was suggested, where one begins with the propagation graphs of small subsystems, forms the interconnectin graph, and verifies the large subsystem interconnections to be totally self-checking. Then, a propagation graph can be formed from the interconnection graph of the large subsystem, and it can be combined with the propagation graphs of other large subsystems to form the interconnection graph of an even larger subsystem. This process can continue until the entire system is modeled and verified to be totally self-checking.

A second application discussed was the placement of check circuits in order to assure the totally self-checking property in a system. Finding a placement requiring a minimum number of checkers was discussed although the model can also be used as an aid to ad hoc methods or heuristics.

Finally, two other applications were briefly suggested. These are data contamination analysis and diagnosis. Examples were used as the principle means for demonstrating them, and their study in a more formal way is currently under investigation.

Appendix

This appendix includes a more formal development of the material in Section II. The definition and theorem numbering corresponds to that used earlier.

For a set of vectors V, V^* represents the set of all finite sequences. A subsystem or system has input, output and state spaces as defined in Section II. The source model, R, is a subset of $A^* \times B$. The malfunctioning source, R^m is $X^* \times B$. The sink model, Σ is C^*. A system S performs the mapping $S: X \times Y \times F \rightarrow Z$.

Definition 1: S is input-output consistent if

$$\forall \langle \alpha, b \rangle \in R \quad S(\alpha, \beta, \phi) \in \sum.$$

Definition 2: S is fault secure if

$$\forall f \in F \quad \forall \langle \alpha, b \rangle \in R \quad S(\alpha, b, f) = S(\alpha, b, \phi) \text{ or } S(\alpha, b, f) \notin \sum.$$

Definition 3: Let $\alpha_1 \circ \alpha_2$ represent the concatenation of two sequences. S is self-testing if

$$\forall f \in F \quad \forall \langle \alpha, \phi \rangle \in R \text{ such that } S(\alpha_1, b, f) = S(\alpha_1, b, \phi)$$

$$\exists \alpha_2 \in A^* \text{ such that } \langle \alpha_1 \circ \alpha_2, b \rangle \in R, \text{ and}$$

$$S(\alpha_1 \circ \alpha_2, b, f) \notin \sum.$$

Definition 4: Same as in Section II.

Let $\mathcal{V} = V_1 \times V_2 \times, \cdots, \times V_n$ represent an n-tuple of symbols, such as might be simultaneously applied to n input signal sets or observed at n output signal sets. Then $t_j^E: \mathcal{V}^* \times \mathcal{V}^* \rightarrow \mathcal{N}$ (the nonnegative integers) is defined as

$$t_j^E(\beta, \beta') = \text{the first position (time) in signal set } j \text{ where } \beta \text{ and } \beta' \text{ differ.}$$

If the sequences β and $\beta' \in \mathcal{V}^*$ do not differ in the signal set j then $t_j^E(\beta, \beta') = \infty$.

For a given set of code spaces $W_i \subseteq V_i$, we define $t_j^N: \mathcal{V}^* \rightarrow \mathcal{N}$ as $t_j^N(\beta)$ = the first position in signal set j of β that contains a noncode word. If β contains only code words in signal set j, then $t_j^N(\beta) = \infty$.

Definition 5: For input signal set i and output signal set j, the *minimum response time*, δ_{ij}, is the minimum $t_j^E(S(\alpha, b, \phi), S(\alpha', b, \phi)) - t_i^E(\alpha, \alpha')$ over all $\langle \alpha, b \rangle \in R$ and $\langle \alpha', b \rangle \in R^M$ for which $t_k^E(\alpha, \alpha') = \infty$ when $k \neq i$.

Definition 6: The pair of input and output signal sets i, j is *noncode transmitting* if

$$t_j^N(S\alpha', b, \phi)) = t_j^E(S(\alpha, b, \phi), S(\alpha', b, \phi))$$

for all $\langle \alpha, b \rangle \in R$, $\langle \alpha', b \rangle \in R^M$ such that

$$t_i^E(\alpha, \alpha') = t_i^N(\alpha') \text{ and}$$

$$t_k^E(\alpha, \alpha') = \infty \text{ when } k \neq i.$$

The *maximum noncode transmission time*, T_{ij}, is the maximum $t_j^N(S(\alpha', b, \phi)) - t_i^N(\alpha, \alpha')$ over all α, α' satisfying the above condition.

Theorem 1: For a system S with two input signal sets, i and k, and output signal set, j, let $\langle \alpha, b \rangle \in R$ and $\langle \alpha', b \rangle \in R^M$. Then $t_j^E(S(\alpha, b, \phi), S(\alpha', b, \phi)) \geq \min (t_i^E(\alpha, \alpha') + \delta_{ij}, t_k^E(\alpha, \alpha') + \delta_{kj})$.

Proof: Consider $\langle \alpha'', b \rangle \in R^M$ where α'' is identical to α in signal set i and identical to α' in signal set j; that is, $t_i^E(\alpha'', \alpha') = t_i^E(\alpha, \alpha')$, and $t_j^E(\alpha'', \alpha') = \infty$; $t_j^E(\alpha, \alpha'') = t_j^E(\alpha, \alpha')$, and $t_i^E(\alpha, \alpha'') = \infty$. Now, $t_k^E(S(\alpha', b, \phi), S(\alpha'', b, \phi)) \geq t_i^E(\alpha'', \alpha') + \delta_{ik}$ and $t_k^E(S(\alpha, b, \phi), S(\alpha'', b,)) \geq t_j^E(\alpha, \alpha'') + \delta_{jk}$ by Definition 5. This means that $S(\alpha', b, \phi)$ and $S(\alpha'', b, \phi)$ differ no sooner than $t_i^E(\alpha', \alpha'') + \delta_{ik}$ and $S(\alpha, b, \phi)$ and $S(\alpha'', b, \phi)$ differ no sooner than $t_j^E(\alpha, \alpha'') + \delta_{jk}$. Then it follows that $S(\alpha, b, \phi)$ and $S(\alpha', b, \phi)$ can differ no sooner than the minimum of $t_i^E(\alpha'', \alpha') + \delta_{ik}$ and $t_j^E(\alpha, \alpha'') + \delta_{jk}$. Or, in other words, $t_k^E(S(\alpha, b, \phi), S(\alpha', b, \phi)) \geq \min (t_i^E(\alpha'', \alpha') + \delta_{ik}, t_j^E(\alpha, \alpha'') + \delta_{jk})$ from the definition of t_i^E. Then, since $t_i^E(\alpha'', \alpha') = t_i^E(\alpha, \alpha')$ and $t_j^E(\alpha, \alpha'') = t_j^E(\alpha, \alpha')$ we have

$$t_k^E(s(\alpha, b, \phi), S(\alpha', b, \phi)) \geq \min (t_i^E(\alpha, \alpha') + \delta_{ik}, t_j^E(\alpha, \alpha') + \delta_{jk}). \quad \square$$

Theorem 2: Let the pair of input and output signal sets, i, j be noncode transmitting. Then

$$t_j^N(S(\alpha', b, \phi)) = t_j^E(S(\alpha, b, \phi), S(\alpha', b, \phi))$$

for any $\langle \alpha, b \rangle \in R$, $\langle \alpha', b \rangle \in R^M$ such that

a) $t_i^E(\alpha, \alpha') = t_i^N(\alpha')$
b) $t_k^E(\alpha, \alpha') + \delta_{kj} > t_i^E(\alpha, \alpha') + T_{ij}$ when $k \neq i$.

Proof: Consider $\langle \alpha'', b \rangle \in R^M$ where α'' is identical to α' in signal set i and identical to α in all the other signal sets. By Definitions 5 and 6 and the fact that $t_i^E(\alpha, \alpha'') = t_i^E(\alpha, \alpha')$

$$t_j^N(S(\alpha'', b, \phi)) = t_j^E(S(\alpha, b, \phi), S(\alpha'', b, \phi) \leq t_i^E(\alpha, \alpha') + T_{ij}.$$

From the generalization of Theorem 1

$$t_j^E(S(\alpha', b, \phi), S(\alpha'', b, \phi)) \geq \min_{k \neq i} (t_k^E(\alpha, \alpha') + \delta_{kj}).$$

By condition b) of the theorem

$$t_k^E(\alpha, \alpha') + \delta_{jk} > t_i^E(\alpha, \alpha') + T_{ij} \text{ for all } k \neq 1,$$

including the one with minimum $t_k^E(\alpha, \alpha') + \delta_{kj}$.

Combining the above equations and inequalities

$$t_j^E(S(\alpha', b, \phi), S(\alpha'', b, \phi)) > t_j^E(S(\alpha, b, \phi), S(\alpha'', b, \phi)) = t_j^N(S(\alpha'', b, \phi)).$$

This implies that $S(\alpha', b, \phi)$ and $S(\alpha'', b, \phi)$ are identical at $t_j^N(S(\alpha'', b, \phi))$ so $S(\alpha', b, \phi)$ must be noncode at that time. Furthermore, $S(\alpha', b, \phi)$ and $S(\alpha, b, \phi)$ must be identical prior to that time. This implies that

$$t_j^N(S(\alpha', b, \phi)) = t_j^E(S(\alpha', b, \phi), S(\alpha, b, \phi)). \quad \square$$

Definition 7: The pair of input and output signal sets i, j is *strongly noncode transmitting* if it is noncode transmitting, and for all $\langle \alpha', b \rangle \in R^m$

$$t_i^N(\alpha') \geq t_j^N(S(\alpha', b, \phi)) - T_{ij}.$$

Theorem 3: Let the pair of input and output signal sets i, j be strongly noncode transmitting. Then

$$t_j^N(S(\alpha', b, \phi)) = t_j^E(S(\alpha, b, \phi), S(\alpha', b, \phi)).$$

for any $\langle \alpha, b \rangle \in R$, $\langle \alpha', b \rangle \in R^M$ such that

a) $t_i^E(\alpha, \alpha') = t_i^N(\alpha')$
b) $t_k^E(\alpha, \alpha') + \delta_{jk} \geq t_i^E(\alpha, \alpha') + T_{ij}$ when $k \neq i$.

Proof: The proof is similar to the proof of Theorem 2. A difference is that

$$t_k^E(\alpha, \alpha') + \delta_{jk} \geq t_i^E(\alpha, \alpha') + T_{ij} \text{ for all } k \neq i.$$

Since $t_i^E(\alpha, \alpha') = t_i^N(\alpha')$, the strongly noncode transmitting property implies that

$$t_j^E(\alpha, \alpha') + T_{ij} \geq t_j^N(S(\alpha', b, \phi)).$$

Combining (1) and (2) from the proof of Theorem 2 and the two inequalities above, we have

$$t_j^E(S(\alpha', b, \phi), S(\alpha'', b, \phi)) \geq t_j^E(S(\alpha, b, \phi), S(\alpha'', b, \phi)) \geq t_j^N(S(\alpha', b, \phi)).$$

When $t_j^E(S(\alpha', b, \phi), S(\alpha'', b, \phi)) > t_j^E(S(\alpha, b, \phi), S(\alpha'', b, \phi))$ then the argument used in Theorem 2 suffices. This only leaves the case where

$$t_j^E(S(\alpha', b, \phi), S(\alpha'', b, \phi)) = t_j^E(S(\alpha, b, \phi), S(\alpha'', b, \phi)) \geq t_j^N(S(\alpha', b, \phi)).$$

Here, $S(\alpha', b, \phi)$ and $S(\alpha, b, \phi)$ must be identical up until $t_j^E(S(\alpha', b, \phi), S(\alpha'', b, \phi))$. Hence

$$t_j^E(S(\alpha, b, \phi), S(\alpha', b, \phi)) \geq t_j^E(S(\alpha', b, \phi), S(\alpha'', b, \phi)) \geq t_j^N(S(\alpha', b, \phi)).$$

Then it follows that

$$t_j^E(S(\alpha, b, \phi), S(\alpha', b, \phi)) = t_j^N(S(\alpha', b, \phi))$$

because $S(\alpha, b, \phi)$ is never noncode (consistency). $\square$

References

[1] F. F. Sellers, M. Y. Hsiao, and L. W. Bearnson, *Error Detecting Logic for Digital Computers.* New York: McGraw-Hill, 1968.
[2] W. C. Carter, and P. R. Schneider, "Design of dynamically checked computers," *IFIP 68*, vol. 2, Edinburg, Scotland, pp. 878-883, Aug. 1968.
[3] D. A. Anderson, "Design of self-checking digital networks using coding techniques," Coord. Sci. Lab., Tech. Rep. R-527, Univ. Illinois, Urbana, Sept. 1971.
[4] J. E. Smith and G. Metze, "Strongly fault secure logic networks," *IEEE Trans. Comput.*, vol. C-27, pp. 491-499, June 1978.
[5] D. C. Ko and M. A. Breuer, "The design of self-checking multi-output combinational circuits," in *Proc. Nat. Comput. Conf.*, 1977, pp. 711-721.
[6] M. Diaz, P. Azema, and J. M. Ayache, "Unified design of self-checking and fail-safe combinational circuits and sequential machines," *IEEE Trans. Comput.*, vol. C-28, pp. 276-281, Mar. 1979.
[7] F. Ozguner, "Design of totally self-checking asychronous and synchronous sequential machines," in *Proc. 7th Int. Symp. Fault-Tolerant Comput.*, June, 1977, pp. 124-129.
[8] R. David and P. Thevenod/Fosse, "Design of totally self-checking asynchronous modular circuits," *J. Des. Automat. Fault-Tolerant Comput.*, vol. 2, pp. 271-287, Oct. 1978.
[9] D. A. Anderson and G. Metze, "Design of totally self-checking check circuits for *m*-out-of-*n* codes," *IEEE Trans. Comput.*, vol. C-22, pp. 263-269, Mar. 1973.
[10] J. E. Smith, "The design of totally self-checking check circuits for a class of unordered codes," *J. Des. Automat. Fault-Tolerant Comput.*, vol. 1, pp. 321-342, Oct. 1977.
[11] M. J. Ashjaee and S. M. Reddy, "On totally self-checking checkers for separable codes," *IEEE Trans. Comput.*, vol. C-26, pp. 737-744, Aug. 1977.
[12] M. A. Marouf and A. D. Friedman, "Efficient design of self-checking checkers for any *m*-out-of-*n* code," *IEEE Trans. Comput.*, vol. C-27, pp. 482-490, June 1978.
[13] R. M. Sedmak, and H. L. Liebergot, "Fault-tolerance of a general purpose computer implemented by very large scale integration," in *Proc. 8th Int. Symp. Fault-Tolerant Comput.*, June 1978, pp. 137-141.
[14] Y. Crouzet and C. Landrault, "Design of self-checking MOS-LSI circuits, application to a four-bit microprocessor," in *Proc. 9th Int. Symp. Fault-Tolerant Comput.*, June 1979, pp. 189-192.
[15] D. A. Rennels, "Architectures for fault-tolerant spacecraft computers," *Proc. IEEE*, vol. 66, pp. 1255-1268, Oct. 1978.
[16] *Proc. 2nd Symp. Large Scale Dig. Calculat. Mach. in Annals of the Computation Laboratory* XXVI. Cambridge, MA: Harvard Univ. Press, 1949.
[17] J. P. Eckert *et al.*, "The UNIVAC system," in *Proc. AIEE-IRE Conf.*, 1961, pp. 6-16.
[18] W. C. Carter *et al.*, "Design of serviceability features for the IBM system/360," *IBM J. Res. Develop.*, vol. 8, pp. 115-126, Apr. 1964.
[19] J. L. Fox, "Availability design of system 370 model 168 multiprocessor," in *Proc. 3rd USA-Japan Comput. Conf.*, 1975.
[20] W. N. Toy, "Fault-tolerant design of local ESS processors," *Proc. IEEE*, vol. 66, pp. 1126-1145, Oct. 1978.

[21] M. Diaz and J. M. DeSouza, "Design of self-checking microprogrammed controls," in *Proc. 5th Int. Symp. Fault-Tolerant Comput.*, June 1965, pp. 137–142.

[22] R. W. Cook *et al.*, "Design of a self-checking microprogram control," *IEEE Trans. Comput.*, vol. C-22, pp. 255–262, Mar. 1973.

[23] J. Wakerly, *Error Detecting Codes, Self-Checking Circuits, and Applications.* New York: North-Holland, 1978.

[24] D. Ho, "Design of totally self-checking digital systems," Coord. Sci. Lab., Tech. Rep. R-723, Univ. Illinois, Oct. 1975.

[25] M. J. Ashjaee, "Totally self-checking check circuits for separable codes," Ph.D. dissertation, Univ. Iowa, July 1976.

[26] G. Maki, "A self-checking microprocessor design," *J. Des. Automat. Fault-Tolerant Comput.*, vol. 2, pp. 15–27, Jan. 1978.

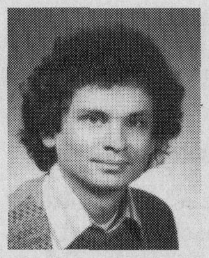
[27] W. C. Carter *et al.*, "Cost effectiveness of self-checking computer design," in *Proc. 7th Int. Symp. Fault-Tolerant Comput.*, June 1977, pp. 117–123.

[28] J. E. Smith, "The design of totally self-checking combinational circuits," Coord. Sci. Lab., Tech. Rep. R-737, Univ. Illinois, Urbana, Aug. 1976.

[29] A. M. Usas, "A totally self-checker design for the detection of errors in periodic signals," *IEEE Trans. Comput.*, vol. C-24, pp. 483–488, May 1975.

[30] J. E. Smith and P. L. Lam, "A model for totally self-checking systems," Dep. Elect. Comput. Eng., Tech. Rep. ECE-80-5, Univ. Wisconsin-Madison, Mar. 1980.

James E. Smith (S'74–M'76) 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-Champaign, in 1972, 1974, and 1976, respectively.



From 1972 to 1976 he was a Research Assistant at the Coordinated Science Laboratory, University of Illinois. In 1976 he joined the faculty of the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, where he is an Associate Professor. He spent the summer of 1978 at the IBM T. J. Watson Research Center, Yorktown Heights, NY. From September 1979 to June 1981 he took a leave of absence from his position at the University of Wisconsin to work for the Control Data Corporation at Arden Hills, MN. His current research interests are in high performance computer architectures and fault-tolerant computing.

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

Paklin Lam was born August 20, 1953 in Hong Kong. He received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan in 1975. From September 1976 to December 1977 he attended the University of Wisconsin-Madison where he received the M.S. degree in electrical engineering. In January 1978 he returned to Taiwan to take an industrial position.