Design and Application of Self-Testing Comparators Implemented with MOS PLA's

YUVAL TAMIR, STUDENT MEMBER, IEEE, AND CARLO H. SÉQUIN, FELLOW, IEEE

Abstract — **A high probability of detecting errors caused by hardware faults is an essential property of any fault-tolerant system. VLSI technology makes the use of *duplication and matching* for error detection practical and attractive. A critical circuit in this context is a *self-testing comparator.* Faults in the comparator must be detected so that they do not mask discrepancies between the duplicated modules.**

This paper discusses the implementation of comparators which are self-testing with respect to faults caused by any single physical defect likely to occur in NMOS and CMOS integrated circuits. A new fault model for PLA's is presented. This model reflects several physical defects in VLSI circuits which are not accounted for in previously published models. It is shown that in a self-testing comparator, implemented as a single two-level NOR–NOR PLA, the number of required product terms grows *exponentially* with the number of input bits. A particular design of a comparator using a single two-level NOR–NOR PLA is discussed. The operation of this comparator under the faults in the fault model is analyzed in detail. The comparator is proven to be self-testing with respect to any likely single fault in the proposed fault model, provided that several guidelines about its physical layout are followed. The use of this comparator as a basic building block of fault-tolerant systems is discussed.

Index Terms — **Concurrent error detection, duplication and matching, faults in VLSI circuits, MOS PLA fault model, programmable logic array, self-testing comparator, two-rail code checker.**

I. Introduction

IDEALLY, a computer system must always generate the "correct" results. Since real systems suffer from design faults, fabrication defects, and other hardware faults, this ideal can never be reached. A minimum requirement from any computer system is that it must not mislead the "outside world" into accepting incorrect outputs as correct. Hence, the "outside world" must be notified (or be able to detect) when errors occur. *Fault-tolerant* systems attempt to achieve a higher probability of producing correct outputs by adapting to changes in the system caused by faults, and continuing correct operation. Explicit or implicit *error detection* is not only required for preventing acceptance of incorrect outputs but is also a necessary first step of any scheme for "recovering" from faults [2].

Checker circuits play a key role in systems with *on-line* error detection. They continuously verify that certain sets of lines in the system carry values that together conform to some *code*. Some of the typical codes used are: parity codes, m-out-of-n codes, arithmetic codes, and two-rail codes [28]. When such codes are used, it is assumed that faults will cause the values on the monitored lines to change in such a way that they will no longer conform to the code. If faults modify the values in unexpected ways so that the incorrect values still conform to the code, the error cannot be detected.

In modern VLSI chips the traditional fault model, that takes into account only single stuck-at faults, is no longer valid [13], [27]. For example, a single physical defect may result in erroneous values on several lines or may convert a combinational circuit into a finite state machine. Hence, simple error-detecting codes, such as a single parity bit, may fail to detect many of the possible errors. Furthermore, evaluating the percentage of faults that can be detected by a given scheme (fault-coverage) is very difficult since we cannot use the traditional method of simulating the effects of all possible faults [20].

No single type of error-detecting code is suitable for use throughout a complex chip such as a microprocessor. While Hamming codes may be used for detecting errors in registers or bus transfers [28], arithmetic codes [3] are needed for checking the operation of the ALU, and duplication and matching must be used for checking control lines and logical operations [8], [26]. The need to use different codes and implement different checkers in different parts of the chip exacerbates the already difficult problem of managing the complexity of VLSI chip design [21]. The increase in the complexity of the design and of its verification decreases the confidence that the design is correct, thereby reducing the overall reliability of the system.

Modern VLSI technology makes duplication and matching an attractive and economically feasible way of implementing error detection. Duplicate high-level functional modules, such as microprocessors or communication chips, can run in parallel, and errors can be detected by comparing module outputs [10], [22], [24]. Errors are detected as long as the first time one, or both, of the two modules fail(s), they produce different outputs. There is no need to predict exactly how defects will affect the modules and there is no increase in the complexity of designing, verifying the design, and testing the chips.

Duplication and matching may also be used to alleviate the problem of system failure due to chips with fabrication defects or undetected design faults. Separately designed functional modules can be used in order to detect latent design and fabrication faults during the operation of the system and pre-

Manuscript received August 5, 1983; revised December 8, 1983. This work was supported by the Defense Advance Research Projects Agency Order 3803, monitored by Naval Electronic System Command under Contract N00039-81-K-0251.

The authors are with the Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

0018-9340/84/0600-0493$01.00 © 1984 IEEE

vent undetected incorrect outputs [4]. Using different designs also virtually eliminates the possibility that the two modules fail in exactly the same way at exactly the same time.

The critical element in any duplication and matching scheme is the circuit that compares the outputs from the two functional modules. Undetected faults in this comparator can mask discrepancies between the outputs of the functional modules. Hence, the comparator must be *self-testing* [1], i.e., during normal operation physical defects in the comparator must result in an error indication.

This paper discusses the design and implementation of self-testing comparators in VLSI. It focuses on designs based on programmable logic arrays (PLA's) [18]. Large VLSI chips are far too complex to allow detailed analysis of all the possible physical defects that can occur and of the effects of these defects on the operation of the circuit. On the other hand, PLA's are characterized by a simple regular structure and are therefore more amenable to thorough analysis. Based on such analysis, a new fault model for PLA's is developed. The model reflects some physical defects that are likely to occur in integrated circuits but are not taken into account in previously published models.

Comparators implemented with two-level AND–OR or NOR–NOR circuits, which are claimed to be self-testing, have been presented in the literature [6], [29]. We show that these designs, which require that the number of product terms grow exponentially with the number of input bits, are *optimal* in terms of size. We present a formal proof that a comparator implemented as a NOR–NOR PLA, based on these designs, is self-testing with respect to most single faults in the new fault model. We propose a few simple layout guidelines that help ensure that the comparator is self-testing with respect to the remaining faults.

Finally, we discuss the application of the self-testing comparator as a basic building block for implementing fault-tolerant systems.

II. The Fault Model

In order to design and implement self-testing circuits, it is necessary to consider the physical defects that are likely to occur with the specific technology being used. However, the design is done at the level of Boolean logic ("ones and zeros") rather than at the level of voltages, currents, and charges. Hence, once the likely physical defects are known, it is desirable to determine the effects that these defects have on the operation of the circuit at the logical level. A description of these effects is called a *fault model*. In this section we present a fault model for general NMOS and CMOS VLSI circuits and use it to develop a detailed fault model for PLA's.

A. Faults in MOS VLSI Digital Circuits

The failure of a VLSI chip may be due to design or fabrication flaws, environmental factors, or a combination of the two [11], [14]. The resulting physical defects consist mainly of breaks in lines, shorts between lines at the same interconnection level (metallization, diffusion, and polysilicon), shorts through the insulator separating different levels, shorts to the substrate, and large imperfections such as scratches across the chip [9], [13]. Other possible defects are incorrect dosage of ion implants, contact windows that fail to open, and misplaced or defective bonds [11]. During the operation of the chip, faults may also be caused by power supply fluctuation, and ionizing or electromagnetic radiation [7], [11].

While the stuck-at fault model [12] can represent the effects of a significant percentage of the physical defects that occur in modern NMOS and CMOS VLSI circuits, it cannot represent the effects of several other possible defects and is therefore insufficient [9], [13], [27]. The effects of most defects can be represented, at the logical level, by a circuit model that consists of a network of switches, loads (for NMOS), and interconnection lines which directly correspond to the transistors and interconnections in the actual circuit [13]. Shorts and breaks in lines can be represented with this circuit model in an obvious way [9]. Shorts to "ground" and "power" are the traditional stuck-at faults. A "switch" may be permanently on or permanently off, corresponding to a gate input stuck-at-1 or -0, respectively. Shorted NMOS loads (pull-ups) are equivalent to an output line s-a-1. Disconnected gate inputs are usually equivalent to s-a-0 or s-a-1 faults. A single break in a line that fans out to many inputs is equivalent to multiple stuck-at faults (all of the same type).

Some physical defects have a more complex effect on the circuit. In NMOS, incorrect dosage of ion implants may cause a threshold shift in a load transistor. This can result in an output voltage that lies between the voltages assigned to logic 0 and logic 1. If the fan-out from the gate is greater than one, some of the attached gates may "interpret" its output as logic 1 while others will interpret it as logic 0. If, at some point in time (clock cycle), the line is supposed to be a logic 1 but is interpreted by at least one of the gates as logic 0, we call it a *weak* 1 fault. Conversely, if the line is supposed to be a logic 0 but is interpreted by at least one of the gates as logic 1, we call it a *weak* 0 fault [24]. It is clear that a line may exhibit both a weak 0 fault and a weak 1 fault, as a result of a single physical defect.

A stuck-at-1 fault is a degenerate case of a weak 0 fault while a stuck-at-0 fault is a degenerate case of a weak 1 fault. If a line is stuck-at-1, *all* the devices connected to it *always* interpret its value as logic 1. If a line has a weak 0 fault, *at least one* of the devices connected to it *always* interprets it as a logic 1.

Breaks in lines are another possible source of weak 0 and weak 1 faults. A break may result in a segment of the line that is only connected to gates of MOS transistors and is therefore essentially "floating." The gates connected to the floating segment of the line receive an incorrect value for the line in one of its states (0 or 1).

A single break in the line can result in the line being stuck-at-1 if all the pull-down devices are disconnected from the rest of the line, and in the line s-a-0 if all the pull-up (or load) devices are disconnected from the rest of the line. Furthermore, if only some of the pull-up or pull-down devices are disconnected from the line, the line may not be s-a-0 or s-a-1 but assume the wrong value for some inputs that only turn on the disconnected pull-ups or pull-downs. In the worst

case, in CMOS or dynamic logic circuits, a break in a line or a transistor that is permanently off can make the output of a supposedly combinational logic circuit dependent on the previous output rather than the current input alone [27]. Such a fault (called a *stuck-open* fault) may escape detection even if all possible input vectors are used to test the circuit [27].

A short between adjacent or crossing lines forces both lines to have the same value. This value may lie between logic 0 and logic 1. Hence, if the two lines are supposed to carry complementary values, the line that is supposed to be at logic 1 may have a weak 1 fault and the line that is supposed to be at logic 0 may have a weak 0 fault. If the circuit is designed so that a short always forces both lines to a well-defined logic value, this value may be always the value of one of the lines that "dominates" because it is driven with larger devices, or it may always be logic 0 (AND operation) or always logic 1 (OR operation).

We assume that if the fault is *transient,* the circuit returns to its original physical structure after the fault has disappeared. It is, of course, possible for a transient fault to cause a permanent change in the *state* of a circuit with memory elements. We assume that, for the duration of the fault (defect), the effects of the defect are *deterministic* so that under identical conditions the effects of a particular defect are always the same. Thus, if a line has a weak 1 fault due to its driver, the devices connected to it which misinterpret the logic 1 as a logic 0 *always* misinterpret the logic 1 as a logic 0.

Traditionally, the term *single fault* has been used to denote an erroneous logic value on a single line in the circuit. From the above it is clear that a single physical defect may result in erroneous logic values on several lines in the circuit. Hence, we will use the term *single fault* to denote the effect, at the logical level, of a single physical defect.

B. A Fault Model for MOS PLA's

For any VLSI circuit, the effect of a single fault on the output is dependent on layout details such as which lines are adjacent, which lines cross each other, etc. One of the advantages of using PLA's is that their regular structure simplifies analysis of the effects of faults on its outputs and therefore facilitates test vector generation and determination of fault coverage. In this section we discuss how the faults discussed above affect the operation of a two-level NOR–NOR MOS PLA. To facilitate this discussion, a "typical" NMOS PLA is shown in Fig. 1.

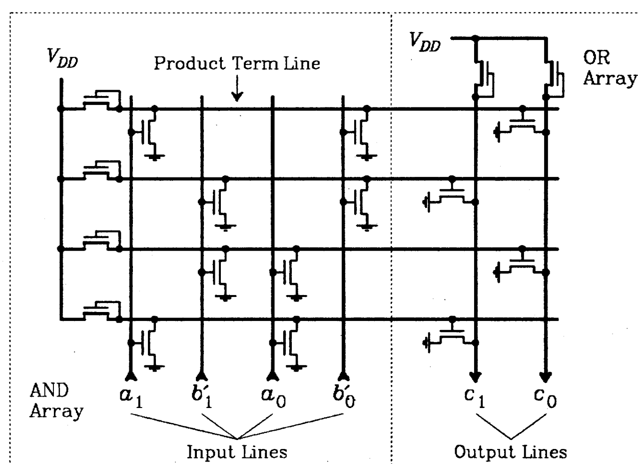

Fig. 1. A self-testing NMOS two-rail code checker.

The most elementary fault model used for PLA's includes three types of faults [16], [19], [29]:

1) A stuck-at fault on an input line, product term line, or output line.

2) A short between two adjacent or crossing lines that forces both of them to the same logic value.

3) A missing or extra crosspoint device in the AND array or in the OR array.

The first two types of faults were explained above and correspond directly to physical defects in the circuit. The third type refers to faults whose effect on the operation of the circuit is equivalent to the effect of a missing or extra crosspoint device. This may be the result of the gate of the crosspoint device stuck-at its "off" value (0 for NMOS, 1 for PMOS) when it should be connected to an input or product term line, or connected to an input or product term line when, by design, it should be permanently held at its "off" value.

A missing crosspoint device has the same effect as a device that always misinterprets the line that drives it as a logic 0 even when it is a logic 1. Thus, a missing crosspoint device fault in the AND array is equivalent to a weak 1 fault on the corresponding input line while a missing crosspoint device fault in the OR array is equivalent to a weak 1 fault on the corresponding product term line. Hence, if weak 1 faults on input lines and product term lines are considered, there is no need to consider missing crosspoint device faults separately.

The above three fault types do not include weak 0 and weak 1 faults or breaks in lines that are not equivalent to stuck faults. Since breaks in lines are one of the main causes of failures in VLSI circuits [9], [13], it is clear that the above simple fault model does not accurately reflect the possible physical defects in an MOS PLA.

Some of the effects of breaks on general MOS circuits cannot occur in PLA's due to their structure. This fact can be used to reduce the complexity of the fault model that must be considered in analyzing the operation of PLA's under faults. One such simplification relies on the fact that input lines are only connected to gates of devices in the PLA. A break in an input line causes a segment of that line to "float" and is therefore equivalent to a weak 0 and/or weak 1 fault. Hence, if weak 0/1 faults on input lines are taken into account, there is no need to consider breaks in input lines separately.

Another important simplification of the fault model is based on the fact that product term lines and output lines only have one pull-up (load) device and that this device is independent of the inputs to the circuit. Every point in a product term or output line is either connected to the single pull-up (load) or permanently disconnected from it (due to a break). For any input, segments of the line that are connected to the pull-up are either set to logic 1 or set to logic 0 by some pull-down device that is turned on by that particular input. A segment of the line that is disconnected from the pull-up is set to logic 0 by the first input that is supposed to set it to 0 and stays

stuck-at-0 for a long time thereafter. Hence, no state is preserved on lines between inputs (clock phases). The troublesome faults that can convert a general combinational circuit into a sequential circuit cannot occur.

Based on the above discussion, a realistic fault model for PLA's must include weak 0/1 faults as well as the possible effects of breaks in product term lines and output lines. Specifically, the following faults must be considered:

a) A weak 0 or weak 1 or both on one input line.
b) A short between two adjacent input lines.
c) A weak 0 or weak 1 or both on one product term line.
d) A short between two adjacent product term lines.
e) A weak 0 or weak 1 or both on one output line.
f) A short between two adjacent output lines.
g) A short between an input line and a crossing product term line.
h) A short between a product term line and a crossing output line.
i) An extra crosspoint device in the AND array.
j) An extra crosspoint device in the OR array.
k) A break in a product term line.
l) A break in an output line.

III. Background and Terminology

Since code checkers are key elements in computer systems with on-line error detection, the design and implementation of various self-testing checkers has been an active research area for many years. This section includes a discussion of some of that work that is relevant to self-testing comparators in VLSI. In addition, the terminology and notation that will be used in the rest of the paper are introduced.

We will assume that two n-bit vectors, $A = (a_{n-1}, a_{n-2}, \cdots, a_0)$ and $B = (b_{n-1}, b_{n-2}, \cdots, b_0)$, are to be compared. In much of the literature *two-rail code checkers* rather than *comparators* are discussed. Given two n-bit vectors $X = (x_{n-1}, x_{n-2}, \cdots, x_0)$ and $Y = (y_{n-1}, y_{n-2}, \cdots, y_0)$, the combined $2n$-bit vector $XY = (x_{n-1}, \cdots, x_0, y_{n-1}, \cdots, y_0)$ is a two-rail codeword if $x_i = y_i'$ for all i such that $0 \leq i \leq n-1$ (where y_i' means the complement of y_i). We will use B' to denote an n-bit vector whose elements are the complements of the elements of B, i.e., $B' = (b_{n-1}', b_{n-2}', \cdots, b_0')$. A two-rail code checker whose input is the bit vector AB' is, effectively, a comparator of vectors A and B. We will start out by making the assumption (that will later be shown to be unnecessary) that all the input bits are available in both complemented and uncomplemented form. Hence, there is no difference between the design of comparators and two-rail code checkers; and we will use the terms "comparator" and "two-rail code checker" interchangeably.

For a given *fault set F*, a checker is said to be *self-testing* if for every fault $f \in F$ there is a code input that results in a noncode output (error indication) [1]. When duplication and matching is used for on-line error detection, we assume that faults (both *permanent* and *transient*) do not occur simultaneously in the duplicated functional modules and in the comparator. If the first fault occurs in one of the functional modules and causes a permanent change in the circuit or in the state of the circuit, we assume that the outputs from the two modules "disagree" and result in a noncode output from the comparator before any faults occur in the comparator. If the first fault occurs in the comparator, we assume that, before additional faults can occur in the comparator or the functional modules, the set of codewords that will appear as inputs to the comparator will be sufficient to achieve a complete self-test of the comparator. Depending on the particular implementation of the comparator, this set of codewords may have to include all possible code inputs. If the fault persists for this duration and the comparator is self-testing, its output will indicate an error before a fault in one of the functional modules can lead to an undetected erroneous *functional* output from the system. Based on these assumptions, the comparator only needs to be self-testing with respect to a fault set F that includes all *single faults*, as defined in Section II.

Pioneering work in the field of self-testing checkers was reported by Carter and Schneider [6] whose design of a self-testing two-rail code checker serves as a basis for the comparator discussed in this paper. For the case $n = 2$, Carter and Schneider presented a design of a circuit that checks whether the input is a two-rail codeword and that is also self-testing with respect to any single stuck-at fault [6]. The circuit, shown in Fig. 2, has two output lines c_1 and c_0 where $(c_1, c_0) = (0, 1)$ or $(c_1, c_0) = (1, 0)$ for code input, and $(c_1, c_0) = (0, 0)$ or $(c_1, c_0) = (1, 1)$ for noncode input.

Carter and Schneider's checker has the property that, with no faults, every line in the circuit is 0 for at least one code input and 1 for at least one code input. If any line is stuck-at-0 (s-a-0) or s-a-1, the code input for which the line is supposed to be at 1 or 0, respectively, results in the output $(0, 0)$ or $(1, 1)$.

Wang and Avizienis [29] extended Carter and Schneider's design to arbitrary size input vectors. For each one of the 2^n input codewords there is a single unique product term that is selected (set to 1) only by that codeword. Each product term line selects exactly one of the two output lines. Depending on the parity of the vector $A = (a_{n-1}, a_{n-2}, \cdots, a_0)$, half the code inputs select output c_0 and the other half select output c_1.

The checker proposed by Wang and Avizienis can be described by sum-of-products equations as follows. For any integer k, let I_k denote the set of the k integers between 0 and $k-1$, i.e., $I_k = \{0, 1, \cdots, k-2, k-1\}$. If Q is a set, let $|Q|$ denote the number of elements in Q.

$$c_0 = \sum_{\{Q \mid Q \subset I_n \text{ and } |Q| \text{ even}\}} \left\{ \left(\prod_{\{i \mid i \in Q\}} a_i \right) \left(\prod_{\{j \mid j \in (I_n - Q)\}} b_j' \right) \right\}$$

$$c_1 = \sum_{\{Q \mid Q \subset I_n \text{ and } |Q| \text{ odd}\}} \left\{ \left(\prod_{\{i \mid i \in Q\}} a_i \right) \left(\prod_{\{j \mid j \in (I_n - Q)\}} b_j' \right) \right\}. \quad (1)$$

In NOR–NOR form, similar functionality can be achieved based on (2). An NMOS PLA which implements these equations for the case $n = 2$ is shown in Fig. 1. It should be noted that there are a total of $2n$ input bits to this circuit: all the "a" bits uncomplemented and all the "b" bits complemented. Each "product term" contains exactly n literals.

$$c_0 = \underset{\{Q \mid Q \subset I_n \text{ and } |Q| \text{ odd}\}}{\mathrm{NOR}} \{\mathrm{NOR}(\{a_i \mid i \in Q\} \cup \{b_j' \mid j \in (I_n - Q)\})\}$$

$$c_1 = \underset{\{Q \mid Q \subset I_n \text{ and } |Q| \text{ even}\}}{\mathrm{NOR}} \{\mathrm{NOR}(\{a_i \mid i \in Q\} \cup \{b_j' \mid j \in (I_n - Q)\})\}. \quad (2)$$

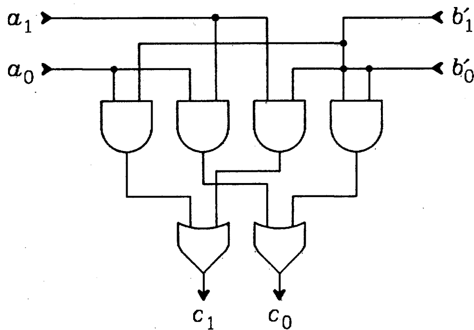

Fig. 2. A self-testing two-rail code checker [6].

In Section V it is shown that the circuit described by (2) is, in fact, a comparator. In Sections VI and VII it is shown that the circuit can be implemented so that it is self-testing with respect to the fault model presented in Section II.

In the literature, checkers which are *fault secure* as well as self-testing have often been discussed [1], [17]. A circuit is said to be fault secure if, for every fault in the prescribed fault set, the circuit never produces an incorrect code output for code inputs [1]. A checker only provides one bit of information to the rest of the system regarding its inputs; the checker's output is code or noncode depending on whether its input is code or noncode, respectively. As long as this binary distinction is performed correctly, it does not matter exactly which code output (or which noncode output) is produced by the checker. Thus, the concept of fault-secure checkers is meaningless [23] and the fault-secure property will not be considered further in this paper.

IV. Optimal Design of Self-Testing Comparators Using Two-Level Logic

Published work on self-testing checkers usually consists of a circuit design and a proof that the circuit is self-testing. There has been no attempt to show that the proposed designs are optimal in any respect. The self-testing comparator design proposed by Wang and Avizienis requires 2^n product terms for comparing n-bit vectors. However, it is possible to implement a comparator that has two outputs that are (0, 1) or (1, 0) for code inputs and (1, 1) for noncode inputs based on the equations

$$c_0 = a_0' + b_0' + \sum_{i=1}^{i=n-1} (a_i b_i' + a_i' b_i)$$

$$c_1 = a_0 + b_0 + \sum_{i=1}^{i=n-1} (a_i b_i' + a_i' b_i).$$

This comparator is self-testing with respect to faults in the input lines and output lines but requires only $4n$ product terms. However, this comparator is *not* self-testing with respect to stuck faults on the product term lines. The question thus arises whether 2^n is the minimum number of product terms necessary for a comparator that is self-testing with respect to a realistic fault model which also takes into account faults affecting the product term lines.

Since the comparator must be self-testing with respect to stuck-at faults on the output lines, it must have at least two output lines [6]. The use of more than two lines has been proposed [23]; however, since limited communication bandwidth is a severe problem in VLSI systems, it is preferable to minimize the bandwidth dedicated to transmitting self-testing information. Hence, we will only consider comparators with two output lines.

There are two possible ways to "code" the output from the comparator and still allow self-testing of the output lines: a) the code output is (0, 1) or (1, 0) and the noncode (error indication) output is (0, 0) or (1, 1); b) the code output is (0, 0) or (1, 1) and the noncode (error indication) output is (0, 1) or (1, 0). Option a) is preferable since it allows the comparator to be self-testing with respect to shorts between the output lines as well as any other fault that causes a *unidirectional error*. A unidirectional error means that, due to a fault, some lines that are supposed to be at logic 0 are at logic 1 *or* some of the lines that are supposed to be at logic 1 are at logic 0, *but not both*. It has been shown that the faults that are most likely to occur in PLA's (fault types 1), 2), and 3) in Section II), can result only in a unidirectional error [16]. Therefore, only comparators with the option a) encoding of the outputs will be considered.

Although the design of a self-testing comparator presented by Wang and Avizienis [29] uses 2^n product terms, one for each code input, this is never shown to be a necessary property of self-testing comparators implemented with PLA's. In several papers [15], [29] it is claimed that it is "desirable" to use PLA's that are *nonconcurrent,* i.e., where each code input selects only one product term. Wang and Avizienis propose a general approach to the design of self-testing PLA's that always results in a nonconcurrent circuit. They also give an example of a PLA where concurrency leads to a circuit which is not self-testing [29]. However, nonconcurrency is not presented as a *necessary* property nor is there any mention of a problem with product terms that are selected by more than one code input.

In this section it is shown that the exponential growth in the number of product term lines is indeed necessary for self-testing. For any two-level NOR–NOR implementation, it is shown that every code input must select exactly one product term line and that no two different code inputs can select the same product term line. This is necessary even if the only faults considered are single stuck-at faults on the input, output, and product lines. The proof that the same requirement also applies to two-level AND–OR implementations is almost identical and will not be presented here.

Lemma 1: Every product term must be selected (set to 1) by at least one codeword.

Proof: Assume that there is a product term that is not selected by any codeword. A stuck-at-0 fault on this product term line will not be detected during normal operation thus violating the self-testing requirement.

Lemma 2: Every codeword must select at least one product term.

Proof: If there is some codeword that does not select any product term, the comparator output for that codeword will be the noncode output (1, 1), which incorrectly signals an error.

Lemma 3: All the product terms selected by a single codeword must be connected to the same single output in the OR array.

Proof: If any of the product terms selected by a codeword are connected to both outputs in the OR array, then, for that codeword, the output will be the noncode output $(0, 0)$, which incorrectly signals an error. Similarly, the output will be $(0, 0)$ if the product terms are not all connected to the same output line.

Lemma 4: No product term can be selected by more than one codeword.

Proof: By contradiction. Assume that P_i is a product term which is selected by the two code inputs $AA = (a_{n-1}, \cdots, a_0, a_{n-1}, \cdots, a_0)$ and $BB = (b_{n-1}, \cdots, b_0, b_{n-1}, \cdots, b_0)$. Since the two codewords are different, there exists an integer k $(0 \leq k \leq n - 1)$ such that $a_k \neq b_k$. P_i is selected only if all the literals in the expression corresponding to P_i are 0. Since P_i is selected by both codewords, it must be independent of bit k from the two functional modules. Hence, P_i is also selected by the code input $WW = (a_{n-1}, \cdots, a_k', \cdots, a_0, a_{n-1}, \cdots, a_k', \cdots, a_0)$ and by the *noncode* input $Q = (a_{n-1}, \cdots, a_k', \cdots, a_0, a_{n-1}, \cdots, a_k, \cdots, a_0)$. Since Q is a noncode input, the corresponding output produced by the comparator must be noncode. When P_i is selected, it sets to 0 the one output line it is connected to. Hence, Q must select another product term P_j connected to the other output line, so that the noncode output $(0, 0)$ will be produced. By Lemma 1, P_j must also be selected by at least one codeword $CC = (c_{n-1}, \cdots, c_k, \cdots, c_0, c_{n-1}, \cdots, c_k, \cdots, c_0)$. Since in CC, bit k from both functional modules is the same, and in Q bit k from one unit is the complement of bit k from the other unit, P_j must not include the literal corresponding to bit k from at least one of the two functional modules. Hence, since Q selects P_j, either AA or WW must also select P_j. Without loss of generality, assume WW selects P_j. From the above, WW also selects P_i. But in the OR array P_j is connected to a different output line from P_i. Hence, Lemma 3 above is "violated" and the codeword WW results in the *noncode* output $(0, 0)$. Thus, the assumption that there exists a product term that is selected by more than one code input must be incorrect.

Lemma 5: Every codeword must select one, and only one, product term.

Proof: By Lemma 2, every codeword must select *at least* one product term. Assume that the codeword $AA = (a_{n-1}, \cdots, a_0, a_{n-1}, \cdots, a_0)$ selects the two product terms P_i and P_j. By Lemma 4, no other codeword except AA can select P_i or P_j. Hence, a stuck-at-0 fault on the P_i or P_j lines can only be detected by the input AA. By Lemma 3, both P_i and P_j must be connected to the same output line in the OR array. Hence, when the codeword AA is applied, the output from the PLA will be the same whether or not one of the product term lines P_i or P_j is stuck-at-0. Thus, a stuck-at-0 fault on one of the product term lines P_i or P_j will not be detectable by any codeword, thus violating the self-testing requirement.

Theorem 1: A self-testing comparator of two n-bit vectors that has two output lines and is implemented as a two-level NOR–NOR PLA, must have *exactly* 2^n product terms.

Proof: By Lemma 1, every product term line is selected by at least one codeword. By Lemma 5, every codeword selects one, and only one, product term line. Hence, the number of product term lines is equal to the number of codewords. Since there are n bits of output from each one of the two functional modules, there are 2^n codewords. Therefore, the number of product term lines is *exactly* 2^n. Q.E.D.

Any comparator of two n-bit vectors must have at least $2n$ input lines (two lines for every pair of bits being compared). As previously discussed, at least two output lines are necessary. Based on the proof presented in this section, exactly 2^n product term lines are necessary for any two-level NOR–NOR implementation. Hence, the design based on (2), which was discussed in the previous section, is optimal. In the next three sections a PLA implementation of a self-testing comparator based on this design is analyzed in detail.

V. Fault-Free Operation of the Comparator

In the previous section we proved that any self-testing comparator implemented as a single two-level NOR–NOR PLA must have 2^n product terms. In this section and the two subsequent sections we discuss a specific self-testing comparator based on (2) in Section III which satisfies this necessary property.

Although a comparator based on (2) has been discussed in the literature, we could find no rigorous proof that it indeed functions as a comparator. For completeness, we present such a proof in this section. To prove that, with no faults, the circuit described by (2) is a comparator, we first show that if $A = B$, the output is $(0, 1)$ or $(1, 0)$. Then we show that if $A \neq B$, the output is $(0, 0)$ or $(1, 1)$.

If $A = B$, there are exactly n ones and n zeros at the inputs. If U is a set of integers $U = \{i \mid a_i = 0\}$, then for every integer j such that $j \in (I_n - U)$, $a_j = b_j = 1$. Thus, $b_j' = 0$, and the one product term that corresponds to $Q = U$ in (2) is selected. Every other product term includes the literal a_j for some $j \in (I_n - U)$ or b_i' for some $i \in U$. Hence, all of the other product terms are set to 0. Thus, only the one output line connected to the single selected product term is set to 0, and the output is $(0, 1)$ or $(1, 0)$.

If $A \neq B$, the two bit-vectors differ by at least one bit. Consider the product term

$$\text{NOR}(\{a_i \mid i \in Q\} \cup \{b_j' \mid j \in (I_n - Q)\}) \tag{3}$$

for some $Q \subset I_n$. Assume that A and B differ in bit r, $r \in I_n$, so $a_r = 1$ and $b_r' = 1$ in the input AB. If $r \in Q$, the product term is set to 0 since it contains the literal a_r. If $r \notin Q$, the product term is set to 0 since it contains the literal b_r'. Hence, all of the product terms are set to 0 and the output is $(1, 1)$.

Since the two bit-vectors differ by at least one bit, if there does *not* exist any integer $r \in I_n$ such that $a_r = 1$ and $b_r' = 1$, there must exist an integer $s \in I_n$ such that $a_s = 0$ and $b_s' = 0$ in the input AB. If AB does not select any product term, the output is $(1, 1)$. Assume that the product term that corresponds to $Q = Q_1$ (3) is selected. If $s \in Q_1$, consider

the set $Q_2 = Q_1 - \{s\}$. Since $Q_2 \subset Q_1$, $I_n - Q_2 = I_n - Q_1 + \{s\}$, and $b_s' = 0$, the product term that corresponds to $Q = Q_2$ will also be selected. If $s \notin Q_1$, consider the set $Q_3 = Q_1 + \{s\}$. Since $a_s = 0$ and $I_n - Q_3 \subset I_n - Q_1$, the product term that corresponds to $Q = Q_3$ will also be selected. Thus, either the product terms corresponding to Q_1 and Q_2 will be selected, or the product terms corresponding to Q_1 and Q_3 will be selected. The number of elements in Q_1 is one greater than the number of elements in Q_2 and is one less than the number of elements in Q_3. Hence, either $|Q_2|$ and $|Q_3|$ are even while $|Q_1|$ is odd, or $|Q_2|$ and $|Q_3|$ are odd while $|Q_1|$ is even. Thus, the product terms corresponding to Q_2 and Q_3 are connected to the same output line, which is different from the output line to which the product term corresponding to Q_1 is connected. Therefore, product terms connected to both output lines are always selected and the output is (0, 0).

VI. Identification and Elimination of Undetectable Faults

Given that the circuit described by (2) functions as claimed when it is *fault free*, it remains to be shown that the circuit is self-testing with respect to any single fault in the fault model described in Section II. Specifically, it must be shown that for any such fault there exists a code input that results in a noncode output (0, 0) or (1, 1) from the comparator. In this section it is shown that there are a few faults in the fault model with respect to which the circuit is *not* self-testing. We refer to these problematic faults as *undetectable* faults. Layout guidelines that prevent these faults from occurring in the actual circuit are discussed.

A. A Short Between Adjacent Product Term Lines

One of the possible faults is a short between two adjacent product term lines that forces both of the lines to logic 1 when they are supposed to be carrying different values [fault type d)]. If the two product term lines are connected to the same output line, there is no code input that results in a noncode output. In fact, the circuit continues to function correctly despite this fault. The reason for this is that if one of the product term lines connected to an output line is selected, that output line is set to logic 0 regardless of the value of any other product term connected to it. It is undesirable to allow this fault to remain undetected since the situation may deteriorate in time and intermittently cause weak 0 or weak 1 faults that will not be detected and will later combine with an additional fault to cause more serious undetectable faults.

As indicated by Wang and Avizienis [29], the possibility that this undetectable fault will occur can be eliminated by ensuring that product term lines connected to the same output line are not adjacent. Since the same number of product term lines are connected to each output line, this guideline is easy to obey and incurs no penalty in terms of the size or performance of the circuit. The guideline is satisfied by simply alternating between product term lines connected to one output line and those connected to the other line.

B. A Short Between a Product Term Line and an Output Line

Another potentially undetectable fault is a short between a product term line P_i and an output line c_m where there is no device at the crosspoint of the two lines. This fault is undetectable if whenever the two lines are supposed to carry a different logic value, they are both forced to logic 1.

The short between P_i and c_m is not detectable since the faulty circuit will behave as follows. For the code input XX that is supposed to select P_i, P_i is supposed to be at logic 1 and c_m at logic 1 (since the other output line $c_{m'}$ is supposed to be at logic 0). Hence, there is no change in the output from the circuit. On the other hand, P_i is supposed to be at logic 0 and c_m is supposed to be at logic 1 for every code input YY, such that $YY \neq XX$ and the number of $a_i (i \in I_n)$ inputs that are at logic 0 in YY has the *same* parity as the number of a_i inputs that are at logic 0 in XX. For these code inputs, P_i is forced to logic 1 but this has no effect on c_m' which is supposed to be at logic 0. For the remaining 2^{n-1} code inputs, P_i is supposed to be at logic 0 and c_m is supposed to be at logic 0. Hence, there is no change in the output from the circuit.

The short between P_i and c_m can be made detectable if it is possible to ensure that when P_i is at logic 0, it forces c_m to logic 0 as well. In NMOS, this can be done by using large crosspoint devices in the AND array so that a single device can pull down two load devices: the output line pull-up as well as the product term line pull-up. In CMOS, this can be done by using large crosspoint devices in the AND array so that a single device can discharge the precharged output line and product term line together within the circuit's clock period. Unfortunately, larger AND array crosspoint devices lead to a larger PLA that is also slower due to larger capacitances.

It is possible that due to a short between a product term line P_i and an output line c_m both lines always assume the value at which c_m is supposed to be. Using arguments similar to the above, it can be shown that, in this case, the short is undetectable regardless of whether or not there is a device at the crosspoint of the two lines [25]. This short can also be made detectable by using large AND array crosspoint devices that ensure that when P_i is at logic 0, it forces c_m to logic 0 as well.

C. Shorts Resulting in Simultaneous Weak 0 and Weak 1 Faults

In this subsection we consider the possibility that, due to a short, two lines that are supposed to carry complementary values are both forced to a value between logic 0 and logic 1. The result is a weak 0 fault on one of the lines and a weak 1 fault on the other line. Such shorts may be undetectable by any code input. To show that the circuit is not self-testing with respect to such a short, it is sufficient to show that the fault is undetectable under the worst possible combination of devices that misinterpret the values on the lines.

1) A Short Between Adjacent Product Term Lines: As discussed in Section VI-A, adjacent product term lines should be connected to different output lines. If a short between two product term lines P_i and P_j forces both to a value between

logic 0 and logic 1 when they are supposed to be carrying different values, this fault may be undetectable. For a code input XX, the short can affect the output only if XX is supposed to select either P_i or P_j. Without loss of generality, assume that XX is supposed to select P_i. All other product term lines (including P_j) are not supposed to be selected by XX. However, a short between P_i and P_j can cause the OR array device connected to P_i to misinterpret it as logic 0 and the device connected to P_j to misinterpret it as logic 1. Hence, despite the fault, only one product term line (P_j) is interpreted as being selected and the output from the circuit is a code output. Thus, this short is not detected by any code input.

It can be shown that there exists a *noncode* input that, due to the short between product term lines, results in code output. Hence, this short, that is not detectable by code inputs, can mask noncode inputs. Thus, the PLA should be laid out in such a way that either this short cannot occur, or if it does occur, both lines are guaranteed to be forced to the same logic value instead of some value between logic 0 and logic 1.

We have already shown that the crosspoint devices in the AND array should be made large enough so that they can pull down both the product term line and an output line that it may be shorted to. Assuming that the same pull-ups are used for the product term lines and the output lines, each crosspoint device in the AND array is also able to pull down *two* product term lines. Hence, a short between two product term lines is guaranteed to force them both to logic 0 when they are supposed to be carrying complementary values. It will be shown in Section VII that this ensures that the short can be detected by some code input.

2) A Short Between Adjacent Input Lines: A short between adjacent input lines may also be undetectable by any code input if, whenever the lines are supposed to be carrying complementary values, both lines are forced to a value between logic 0 and logic 1. Consider a short between two adjacent input lines a_h and a_j ($h \neq j$). There exists a code input $XX = (x_{n-1}, \cdots, x_0, x'_{n-1}, \cdots, x'_0)$ for which a_h is supposed to be at logic 0 and a_j is supposed to be at logic 1. Clearly, $x_h = 0$ and $x_j = 1$, so

$$XX = (x_{n-1}, \cdots, x_{h+1}, 0, x_{h-1}, \cdots, x_{j+1}, 1, x_{j-1}, \cdots, x_0, \quad x'_{n-1}, \cdots, x'_{h+1}, 1, x'_{h-1}, \cdots, x'_{j+1}, 0, x'_{j-1}, \cdots, x'_0).$$

Assume that the single product term line that is supposed to be selected by XX is P_i. Since $x_h = 0$, there is a device CA_{hi} at the crosspoint of the input line a_h and the product term line P_i. Assume that, due to the short, the value of both a_h and a_j is forced to some value between logic 0 and logic 1 and that CA_{hi} misinterprets line a_h to be at logic 1. Hence, product term line P_i is *not* selected by code input XX. In the fault-free circuit, the code input

$$YY = (x_{n-1}, \cdots, x_{h+1}, 0, x_{h-1}, \cdots, x_{j+1}, 0, x_{j-1}, \cdots, x_0, \quad x'_{n-1}, \cdots, x'_{h+1}, 1, x'_{h-1}, \cdots, x'_{j+1}, 1, x'_{j-1}, \cdots, x'_0)$$

is supposed to select product term line P_k. Hence, there is a device CA_{jk} at the crosspoint of input line a_j and product term line P_k. Assume that, due to the short, when the input is XX, CA_{jk} misinterprets a_j to be a logic 0 although it is supposed to be at logic 1. In addition, we assume that CA_{hi} and CA_{jk} are the only AND array crosspoint devices that misinterpret the values of a_h and a_j (in particular CA_{hk} interprets a_h *correctly*). Under these assumptions, all the input lines that are supposed to be at logic 0 when the input is YY are interpreted as being at logic 0 by all the AND array crosspoint devices connected to P_k when the input is XX. Hence, P_k is selected by input XX while P_i is not selected by XX. Since no other crosspoint devices are affected by the short, no other product term line except P_k is selected by XX, and the output is a code output. This short does not affect the output from the circuit for any other code input since such input selects a product term other than P_k or P_i. Hence, the short is not detectable by any code input.

In the fault-free circuit, the *noncode* input

$$W = (x_{n-1}, \cdots, x_{h+1}, 0, x_{h-1}, \cdots, x_{j+1}, 1, x_{j-1}, \cdots, x_0, \quad x'_{n-1}, \cdots, x'_{h+1}, 1, x'_{h-1}, \cdots, x'_{j+1}, 1, x'_{j-1}, \cdots, x'_0)$$

does not select *any* product term and the output is noncode. However, due to the short described above between a_h and a_j, W selects P_k and the result is a *code* output from the circuit. Hence, this short, that is not detectable by code inputs, masks a noncode input.

It can be shown that if the adjacent input lines are a_h and b'_j a short between these lines may also be undetectable by code inputs and can mask noncode inputs [25]. Thus, in order to ensure that the comparator is self-testing, it is necessary to prevent shorts between input lines that can force both lines to a value between logic 0 and logic 1 from occurring. This can be done by laying out the PLA so that the separation between input lines is large enough that the probability of a short between them is negligible. Alternatively, the circuits that drive the inputs of the PLA can be designed so that a single pull-down device can overcome *two* pull-up devices so that a short between input lines when they are supposed to be carrying different values will always result in both being forced to logic 0. Unfortunately, these solutions lead to a larger PLA that is also slower due to larger capacitances.

3) A Short Between an Input Line and a Product Term Line: Using arguments similar to the above, it can be shown that if a short between an input line and a product term line is allowed to force both of them to a value between logic 0 and logic 1, an undetectable fault, that can mask noncode inputs, may result. Here again, one way to prevent this situation is to guarantee that when the lines are supposed to be at complementary values they are both always forced to logic 0. This can be done using large pull-down devices in the circuits that drive the inputs of the PLA and using large AND array crosspoint devices. A single AND array crosspoint device or a single pull-down in an input driver must be able to overcome both the pull-up device of the input driver and the pull-up device of the product term line.

D. Layout Guidelines for Eliminating Undetectable Faults

In the previous three subsections we identified several possible faults that are not detectable by any code inputs. All of these faults are shorts between adjacent or crossing lines.

In particular, any short that results in both lines being forced to a value between logic 0 and logic 1 when they are supposed to be carrying complementary values may lead to an undetectable fault. The layout guidelines for preventing these faults from occurring in the actual circuit are summarized below.

1) Adjacent product term lines must be connected to OR array crosspoint devices that control different output lines.

2) The AND array crosspoint devices must be large enough so that a single device can pull down two pull-ups: a product term line pull-up and an output line pull-up or two product term line pull-ups.

3) The circuits that drive the inputs of the PLA must be designed so that a single pull-down device can overcome *two* pull-up devices.

4) The separation between adjacent input lines and between adjacent product term lines should be larger than the minimum separation required by the design rules. This can help reduce the probability of a short between adjacent lines.

VII. The Self-Testing Property of the Comparator

In the previous section it was shown that the proposed comparator is *not* self-testing with respect to some of the possible faults unless certain guidelines about the layout of the circuit and the size of some of its devices are followed. In this section it is shown that the circuit is self-testing with respect to all the other faults in the fault model. It is assumed that some measures, such as those discussed in the previous section, are taken so that the undetectable faults cannot occur. In particular, it is assumed that if there is a short between two lines and the lines are supposed to be carrying complementary values, the value of one of the lines is modified so that they both carry the same logic value.

A. A Weak 0 and/or Weak 1 Fault on a Single Input Line

1) A Weak 0 Fault: Assume that the input line with a weak 0 fault is a_k for some $k \in I_n$. By definition, there is at least one AND array crosspoint device CA_{ki} connected to a_k that always misinterprets a logic 0 on a_k as a logic 1. Hence, the device CA_{ki} is always turned on. Thus, the product term line P_i that is connected to CA_{ki} can never be selected. Therefore, the code input that is supposed to select P_i results in no product term line being selected and the output is noncode $(1, 1)$. An identical argument can be made regarding a weak 0 fault on a b_j' $(j \in I_n)$ input line.

In the presence of a weak 0 fault on one of the input lines, for every crosspoint device which misinterprets the input line to be a logic 1 when it is supposed to be a logic 0, the code input that selects the corresponding product term line in the fault-free circuit results in a $(1, 1)$ output. Thus, the number of code inputs that detect this fault varies between 1 and 2^{n-1}, depending on the number of affected crosspoint devices.

2) A Weak 1 Fault: Assume that the line with a weak 1 fault is a_k for some $k \in I_n$. By definition, there is at least one AND array crosspoint device CA_{ki} connected to a_k that always misinterprets a logic 1 on a_k as a logic 0. We denote the product term line connected to that crosspoint device by P_i. In the fault-free circuit, P_i is selected by some code input $XX = (x_{n-1}, \cdots, x_0, x_{n-1}', \cdots, x_0')$. Since there is a device at the crosspoint of a_k and P_i, the literal a_k is in the product term that corresponds to P_i. Hence, $x_k = 0$. Thus,

$$XX = (x_{n-1}, \cdots, x_{k+1}, 0, x_{k-1}, \cdots, x_0, x_{n-1}', \cdots, x_{k+1}', 1, x_{k-1}', \cdots, x_0').$$

In the fault-free circuit, the code input

$$YY = (x_{n-1}, \cdots, x_{k+1}, 1, x_{k-1}, \cdots, x_0, x_{n-1}', \cdots, x_{k+1}', 0, x_{k-1}', \cdots, x_0')$$

selects some other product term line P_j. Since CA_{ki} misinterprets a logic 1 on a_k to be a logic 0, code input YY selects P_i. Since there is no device at the crosspoint of a_k and P_j, P_j is independent of a_k. Thus, YY also selects P_j despite the fault.

Since the number of a_i $(i \in I_n)$ inputs that are at logic 0 in XX has a different parity from the number of a_i inputs that are at logic 0 in YY, P_i and P_j are connected to different output lines (see (2) in Section III). Since in the faulty circuit the codeword YY selects both P_i and P_j, the output is $(0, 0)$. An identical argument can be made regarding a weak 1 fault on a b_j' $(j \in I_n)$ input line.

B. A Short Between Two Adjacent Input Lines

As previously mentioned, we assume that appropriate layout guidelines are followed so that a short between lines always forces both of the lines to the same logic value rather than to a value between logic 0 and logic 1. Since the inputs to the comparator are all the bits from one of the functional modules and the complements of all the bits from the other module, no two input lines are supposed to have the same value for all code inputs. If the two adjacent shorted lines are a_k and b_k' $(0 \leq k \leq n - 1)$, *every* code input is transformed to noncode input which, as previously shown, results in $(0, 0)$ or $(1, 1)$ output. Any other two input lines are supposed to transfer different values for half of the code inputs. For these code inputs, the short forces a change in value on one of the lines. Since we assume that there are no other faults, this is equivalent to noncode input which, as previously shown, results in $(0, 0)$ or $(1, 1)$ output.

C. A Weak 0 or Weak 1 Fault on a Single Product Term Line

Each product term line is connected to only one output line. Hence, a weak 0 fault on a product term line is simply a stuck-at-1 fault and a weak 1 fault is a stuck-at-0 fault.

1) A Weak 1 (s-a-0) Fault: If one of the product terms is s-a-0, for the code input that is supposed to select that product term, all product terms are set to 0 and the output is $(1, 1)$.

2) A Weak 0 (s-a-1) Fault: Assume that the product term line P_i that corresponds to set $Q = Q_1$ in (3) is s-a-1. For any code input that selects a product term corresponding to some $Q = Q_2 \in I_n$, where the parity of $|Q_1|$ and $|Q_2|$ are different, product terms connected to both output lines are selected, and the output is $(0, 0)$. Thus, half the code inputs will result in a $(0, 0)$ output due to the s-a-1 fault on P_i.

D. A Short Between Two Adjacent Product Term Lines

Since only one product term line is supposed to be selected by every code input, for any pair of adjacent product term lines P_i and P_j there is one code input that is supposed to select P_i but not P_j and there is another code input that is supposed to select P_j but not P_i. We consider the three possible effects of the short when the lines are supposed to carry complementary values.

1) Both lines are always forced to logic 0: In this case, for the two code inputs that correspond to the two product terms (i.e., that are supposed to select them), no product term line will be set to 1 and the output will be $(1, 1)$.

2) Both lines are always forced to logic 1: Since the two product term lines are connected to different output lines, for the two code inputs that correspond to these product term lines, the output will be $(0, 0)$.

3) Both product term lines always assume the value of one of the lines: Assume that the two lines are P_i and P_j, and that P_i always dominates. The code input YY that is supposed to select P_j does not select it since P_j is pulled to logic 0 by P_i, which is not selected by YY. Hence, YY does not select any product term and the output is $(1, 1)$. The code input XX that selects P_i also selects P_j which is pulled to logic 1 by P_i. Since adjacent product term lines are connected to different output lines, XX results in $(0, 0)$ output.

E. A Weak 0 or Weak 1 Fault on a Single Output Line

The output lines do not fan-out within the comparator circuit. Thus, we need only consider the value on the output line at the point of interface with the "outside world." Hence, a weak 0 fault on a product term line is equivalent to a stuck-at-1 fault and a weak 1 fault is equivalent to a stuck-at-0 fault.

Based on (2), any code input where the number of a_i ($i \in I_n$) bits that are at logic 0 is odd is supposed to result in the output $(c_1, c_0) = (1, 0)$. Thus, in the faulty circuit, if c_0 is s-a-1, the output is $(1, 1)$, and if c_1 is s-a-0, the output is $(0, 0)$. A similar argument can be made for any code input where the number of a_i bits that are at logic 0 is even and the output is supposed to be $(c_1, c_0) = (0, 1)$. Hence, 2^{n-1} code inputs will detect a s-a-1 on c_0 and a s-a-0 on c_1 while the other 2^{n-1} code inputs will detect a s-a-0 on c_0 and a s-a-1 on c_1.

F. A Short Between Two Adjacent Output Lines

There are only two output lines that are supposed to carry different values for every code input. Hence, a short will result in $(0, 0)$ or $(1, 1)$ output for every code input.

G. A Short Between an Input Line and a Crossing Product Term Line

Assume that the short is between input line a_k and product term line P_i. Let XX denote the code input that selects P_i in the fault-free circuit. We must consider the case where a_k is connected to a crosspoint device that is connected to P_i (CA_{ki} exists) as well as the case where CA_{ki} does not exist.

If CA_{ki} exists, every one of the 2^{n-1} code inputs for which a_k is supposed to be at logic 1 is supposed to result in P_i at logic 0. The code input XX is the only code input for which a_k is supposed to be at logic 0 while P_i is supposed to be at logic 1. For the rest of the $2^{n-1} - 1$ code inputs, both a_k and P_i are supposed to be at logic 0.

If CA_{ki} does not exist, the product term corresponding to P_i includes the literal b'_k. For the 2^{n-1} code inputs with b'_k at logic 1, both a_k and P_i are supposed to be at logic 0. For the code input XX, b'_k is supposed to be at logic 0, and both a_k and P_i are supposed to be at logic 1. For the rest of the $2^{n-1} - 1$ code inputs, b'_k is supposed to be at logic 0, a_k is supposed to be at logic 1, and P_i is supposed to be at logic 0. Thus, if CA_{ki} does not exist, there is *no* code input for which a_k is supposed to be at logic 0 and P_i is supposed to be at logic 1.

As in the proof for a short between product term lines, we consider the three possible effects of the short when a_k and P_i are supposed to carry complementary values.

1) Both lines are forced to logic 0: If CA_{ki} exists, for the code input XX, P_i is supposed to be the only selected product term line, while a_k is supposed to be at logic 0. We assume that, due to the short, P_i is forced to logic 0 by a_k. Hence, no product term line is selected and the output is $(1, 1)$.

On the other hand, if CA_{ki} does not exist, the literal a_k is not included in the product term that corresponds to P_i. Hence, the code input that selects P_i in the fault-free circuit is of the form

$$XX = (x_{n-1}, \cdots, x_{k+1}, 1, x_{k-1}, \cdots, x_0, x'_{n-1}, \cdots, x'_{k+1}, 0, x'_{k-1}, \cdots, x'_0).$$

Let YY be one of the $2^{n-1} - 1$ code inputs such that $YY \neq XX$ and YY is also of the form

$$YY = (y_{n-1}, \cdots, y_{k+1}, 1, y_{k-1}, \cdots, y_0, y'_{n-1}, \cdots, y'_{k+1}, 0, y'_{k-1}, \cdots, y'_0).$$

In the fault-free circuit YY selects some product term line P_j. Since there is no device at the crosspoint of a_k and P_j, P_j is independent of a_k so the short between a_k and P_i cannot affect P_j. Thus, P_j is selected by YY despite the fault. In the fault-free circuit, the code input

$$ZZ = (y_{n-1}, \cdots, y_{k+1}, 0, y_{k-1}, \cdots, y_0, y'_{n-1}, \cdots, y'_{k+1}, 1, y'_{k-1}, \cdots, y'_0)$$

selects some product term P_s. Since there is no device at the crosspoint of b'_k and P_s, P_s is independent of b'_k. For the code input YY, a_k is supposed to be at logic 1 and P_i at logic 0. Due to the fault, P_i forces a_k to logic 0. Therefore, YY selects P_s as well as P_j. Since the number of a_i ($i \in I_n$) inputs that are at logic 0 in YY has a different parity from the number of a_i inputs that are at logic 0 in ZZ, P_j and P_s are connected to different output lines (see (2) in Section III). Hence, for the code input YY the output is $(0, 0)$.

2) Both lines are forced to logic 1: Let YY be one of the 2^{n-2} code inputs for which a_k is supposed to be at logic 1 *and* the number of a_i inputs that are at logic 0 in YY has a different parity from the number of a_i inputs that are at logic 0 in XX. Due to the short, when the input is YY, a_k forces P_i (that is supposed to be at logic 0) to logic 1. In addition, as in the

fault-free circuit, YY selects another product term that controls a different output line from P_i. Hence, the output from the circuit is $(0, 0)$.

3) Both lines are always forced to the value of a_k *or* they are always forced to value of P_i:

a) Line a_k always dominates: The proof is identical to case 2) above.

b) Line P_i always dominates: There are at least $2^{n-1} - 1$ code inputs of the form

$$YY = (y_{n-1}, \cdots, y_{k+1}, 1, y_{k-1}, \cdots, y_0, y'_{n-1}, \cdots, y'_{k+1}, 0, y'_{k-1}, \cdots, y'_0)$$

that do not select P_i in the fault-free circuit. In the faulty circuit, if P_i always "dominates," YY selects two product term lines that are connected to different output lines. One is the product term line selected by YY in the fault-free circuit and the other is the product term line selected by

$$ZZ = (y_{n-1}, \cdots, y_{k+1}, 0, y_{k-1}, \cdots, y_0, y'_{n-1}, \cdots, y'_{k+1}, 1, y'_{k-1}, \cdots, y'_0)$$

in the fault-free circuit. Hence, the output is $(0, 0)$.

H. A Short Between a Product Term Line and a Crossing Output Line

Assume that the short is between product term line P_i and output line c_m, where $m \in \{0, 1\}$. Let m' denote 0 when m is 1 and denote 1 when m is 0. Let XX denote the code input that selects P_i in the fault-free circuit.

As in the proof for a short between product term lines, we consider the three possible effects of the short when P_i and c_m are supposed to carry complementary values.

1) Both lines are forced to logic 0: In the fault-free circuit there are at least $2^{n-1} - 1$ code inputs that do not select P_i and for which the output is $(c_m, c_{m'}) = (1, 0)$. For any one of these inputs, due to the short, P_i forces c_m to logic 0 and the output is $(0, 0)$.

2) Both lines are forced to logic 1: If there is a device at the crosspoint of P_i and c_m (CO_{im} exists), in the fault-free circuit, for the code input XX that selects P_i, the output is $(c_m, c_{m'}) = (0, 1)$. In the faulty circuit, due to the short, c_m is forced to logic 1. Since none of the product term lines is affected, the output is $(1, 1)$. If CO_{im} does not exist, then, as discussed in Section VI-B, the fault cannot be detected by any code input.

3) Both lines are always forced to the value of P_i *or* they are always forced to value of c_m:

a) If the value of P_i always "dominates," the proof is identical to case 1) above.

b) If the value of c_m always "dominates," then, as discussed in Section VI-C, the fault cannot be detected by any code input.

I. An Extra Crosspoint Device in the AND *Array*

In the fault-free circuit, every product term line P_i is connected to n crosspoint devices in the AND array. For every code input, n of the input lines are at logic 0 and n are at logic 1. If, due to a fault, there are $n + 1$ crosspoint devices connected to P_i, every code input turns on at least one of these devices and sets P_i to logic 0. Thus, the single code input that selects P_i in the fault-free circuit does not select P_i in the faulty circuit. Hence, for that input, no product term line is selected, and the output is $(1, 1)$.

J. An Extra Crosspoint Device in the OR *Array*

An extra crosspoint device in the OR array means that there is one product term line P_i that is connected to both output lines. Hence, for the single code input that selects P_i, the output is $(0, 0)$.

K. A Break in a Product Term Line

Each product term line controls one OR array crosspoint device and is controlled by n AND array pull-down devices and one pull-up (or precharge) device. All the pull-down devices are connected to the "middle" of the line. The pull-up device and the OR array crosspoint device are either connected on opposite ends of the product term line (as shown in Fig. 1) or on the same end of the line.

If the product term line pull-up device and the OR array crosspoint device are on opposite ends of the line, any break in the product term line prevents the segment of the line connected to the OR array device from being pulled up. As a result, the product term line is either floating or stuck-at-0. If the line is floating, its value is constant and independent of the input. Hence, in any case, the product term line segment that controls the output line is either stuck-at-0 or stuck-at-1. Earlier in this section it was shown that a stuck-at fault on a product term line is detectable by some code input.

If the product term line pull-up device and the OR array crosspoint device are on the same end of the line, a break in the product term line disconnects some of the AND array crosspoint devices from the segment of the line connected to the OR array device. As a result, the product term line is selected when it is not supposed to be selected. Let P_i denote the product term line that is selected by the code input $XX = (x_{n-1}, \cdots, x_h, \cdots, x_0, x'_{n-1}, \cdots, x'_h, \cdots, x'_0)$ in the fault-free circuit. A break in P_i disconnects some AND array crosspoint device CA_{hi} from the segment of P_i that controls the OR array device. Since CA_{hi} is controlled by input line a_h, in the fault-free circuit, P_i can only be selected if $a_h = 0$. Hence, $x_h = 0$. In the fault-free circuit, the code input $YY = (x_{n-1}, \cdots, x'_h, \cdots, x_0, x'_{n-1}, \cdots, x_h, \cdots, x'_0)$ selects the product term P_j where P_i and P_j are connected to different output lines. Since the crosspoint device CA_{hi} is disconnected from P_i in the faulty circuit, P_i is not affected by a_h and the code input YY selects *both* P_i and P_j. Hence, the output is a $(0, 0)$.

L. A Break in an Output Line

Each output line is controlled by 2^{n-1} OR array pull-down devices and one pull-up (or precharge) device. All the pull-down devices are connected to the "middle" of the line. The pull-up device and the output from the circuit are either on opposite ends of the output line (as shown in Fig. 1) or on the same end of the line.

If the output line pull-up device and the circuit output are on opposite ends of the line, any break in the output line

prevents the segment of the line that serves as the output from the circuit from being pulled up. As a result, the output line is either floating or stuck-at-0. If the line is floating, its value is constant and independent of the input. Hence, in any case, the segment of the line that serves as the circuit output is either stuck-at-0 or stuck-at-1. Earlier in this section it was shown that a stuck-at fault on an output line is detectable by some code input.

If the output line pull-up device and the circuit output are on the same end of the line, a break in the output line disconnects some of the OR array crosspoint devices from the segment of the line that is the output from the circuit. As a result, the output line is selected when it is not supposed to be selected. Let c_m denote the output line with a break. Let CO_{im} denote an OR array crosspoint device that is disconnected from the segment of c_m that serves as the circuit output. In the fault-free circuit, the product term line P_i that controls CO_{im} is selected by the code input XX. In the faulty circuit, due to the break, the crosspoint device CO_{im} cannot affect the output line c_m. For the code input XX, P_i is the only selected product term. Hence, CO_{im} is the only OR array crosspoint device that is turned on. Therefore, neither output line is pulled down and the circuit produces the noncode output (1, 1).

VIII. Implementation and Application Considerations

In the previous three sections it was shown that, using a single two-level NOR–NOR PLA, it is possible to implement a comparator that is self-testing with respect to any single fault that is likely to occur in MOS VLSI circuits. This result is a necessary prerequisite for the use of duplication and matching as the basic scheme for implementing error detection. However, two main problems remain to be discussed: 1) the size of the comparator, implemented as a single PLA, grows exponentially with the number of bits in the two vectors to be compared, and 2) it is necessary to ensure that all the code inputs will appear as inputs to the comparator often enough so that a complete self-test of the comparator will be performed before there is a chance for multiple faults to occur in the system.

In Section IV it was shown that a self-testing comparator implemented as a single two-level NOR–NOR PLA must have 2^n product term lines. If the output from each one of the duplicated functional modules is, say, 16 bits, this implementation is impractical since it requires $2^{16} = 65536$ product terms. Fortunately, efficient implementations of a self-testing two-rail code checker (comparator) for large input vectors can be achieved by using checkers for smaller input vectors as *cells* that are connected together in a tree structure (see Fig. 3) [15], [28].

Each cell is a self-testing comparator for relatively small bit vectors (2–6 bits wide) which is implemented with a single two-level NOR–NOR PLA as outlined in Section III. A complete tree with h levels of cells, where each cell is an m-bit comparator, can be used to compare m^h bits and contains $(m^h - 1)/(m - 1)$ cells. Hence, if the vectors to be compared are n bits wide, the number of levels in the tree is

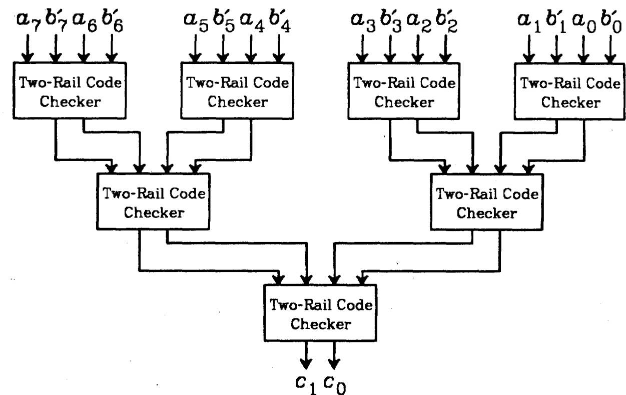

Fig. 3. A self-testing two-rail code checker tree.

$\lceil \log_m n \rceil$ while the total number of cells in the tree is at most $(n - 1)/(m - 1)$. Thus, the number of cells is (approximately) linearly related to n. Hence, tree-structured cellular implementations of self-testing comparators are practical for large input bit vectors.

In the cellular tree-structured implementation of the comparator, a noncode output from any one of the cells presents a noncode input to the cells at the next level. This forces the output from the entire tree to be noncode. Hence, the tree-structured implementation preserves the self-testing property of the cells.

If duplication and matching is used for error detection, the first fault that occurs in the comparator must be detected before additional faults can occur in the comparator or in the functional modules. Thus, a set of codewords that achieves a complete self-test of the comparator must appear as inputs to the comparator within a time interval that is significantly smaller than the mean time between failures for the two functional modules and the comparator together. Based on the results of Sections IV and VII, a complete self-test of a comparator implemented as a single NOR–NOR PLA requires all 2^n codewords to appear at the inputs. If n is large, this requirement may imply that the complete self-test takes so much time that there is an unacceptably high probability that additional faults may occur in the comparator or functional modules before the self-test is completed. Fortunately, for the tree-structured cellular implementation, the number of code inputs required for a complete self-test is only 2^m, where m is the size of the bit vectors compared by each cell [5], [15]. Thus, if the cells are 2-bit comparators, four code inputs are sufficient for a complete self-test of the entire tree.

Even with the relatively small number of code inputs needed for a complete self-test, it may still be difficult to satisfy the requirement that certain codewords appear as inputs to the comparator with some specified frequency, as implied by the assumptions in Section III. We assume that the system consists of subsystems that interact with each other. Each subsystem is implemented with duplicate functional modules and a self-testing comparator so that the failure of a particular subsystem is detected by the other subsystems by simply observing the output from the corresponding comparator [24]. The comparison of the outputs of the two modules that make up each subsystem is performed

at the interface between the subsystem and the rest of the system.

If the "subsystems" are low-level passive circuits such as an ALU or an instruction decoder within a processor, it may not be possible to ensure that the necessary outputs will be generated by the modules. Hence, duplication and matching is *inappropriate* at this level.

Duplication and matching is an attractive scheme for implementing error detection if the subsystems are high-level "intelligent" modules that interact with similar modules. Examples of such high-level subsystems are the computation nodes or the communication nodes in a multicomputer system [24]. In this case, the subsystem may periodically initiate action that causes it to generate all the necessary patterns at its interface with the other subsystems. The subsystem initiating the self-test of its comparator can inform the other subsystems that the next "message" is simply a test and should not be interpreted as "real work."

IX. Summary and Conclusion

We have presented a new fault model for MOS PLA's. This model incorporates several types of faults that are likely to occur in VLSI circuits but are not taken into account in previously published models. Using this more realistic model, it was shown that the widely accepted design of a "self-testing" comparator implemented as a NOR–NOR PLA results in a comparator that is self-testing with respect to any single fault *only* if certain guidelines regarding the physical layout of the circuit are followed. Following these guidelines requires additional silicon area and reduces performance.

The use of duplication and matching of high-level modules for error detection appears especially attractive in view of the difficulties in analyzing and verifying the self-testing properties of the comparator. Despite the simple structure of the proposed self-testing comparator, the analysis and verification of its self-testing properties are surprisingly lengthy and complex. It is therefore doubtful that the effects of faults on large VLSI circuits, such as microprocessors, can be predicted reliably. Furthermore, while restrictions on the layout of the comparator are acceptable in order to enhance its testability, such restrictions cannot be tolerated in the layout of large chips, where one of the major goals is to implement as much functionality as possible per unit area.

Since the area taken up by the comparator may be of concern, it was shown that the proposed comparator is optimal with respect to the area it occupies. Specifically, it is shown that if a single two-level NOR–NOR PLA is used to implement a self-testing comparator of two n-bit vectors, an optimal design must include $2n$ input lines, 2^n product term lines, and 2 output lines. Furthermore, 2^n code inputs are necessary for a complete self-test of any such circuit.

The effectiveness of self-testing comparators as critical elements in duplication and matching schemes for error detection is dependent on the system within which they are employed. If the duplicated functional modules are simple low-level passive circuits, it may not be possible to ensure that the comparator will go through a complete self-test often enough and the scheme may eventually fail due to an undetected fault in the comparator. However, if the duplicated modules are high-level subsystems, the use of self-testing comparators in a duplication and matching scheme is an effective way to implement error detection in VLSI systems.

Acknowledgment

We would like to thank R. Fujimoto and C. Thompson for reviewing an early draft of this paper.

References

[1] 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.

[2] T. Anderson and P. A. Lee, "Fault tolerance terminology proposals," in *Proc. 12th Fault-Tolerant Computing Symp.*, Santa Monica, CA, June 1982, pp. 29–33.

[3] A. Avizienis, "Arithmetic error codes: Cost and effectiveness studies for application in digital system design," *IEEE Trans. Comput.*, vol. C-20, pp. 1322–1330, Nov. 1971.

[4] ——, "Design diversity—The challenge of the Eighties," in *Proc. 12th Fault-Tolerant Computing Symp.*, Santa Monica, CA, June 1982, pp. 44–45.

[5] D. C. Bossen, D. L. Ostapko, and A. M. Patel, "Optimum test patterns for parity networks," in *Proc. AFIPS*, vol. 37, 1970, pp. 63–68.

[6] W. C. Carter and P. R. Schneider, "Design of dynamically checked computers," in *Proc. IFIPS*, Edinburgh, Scotland, Aug. 1968, pp. 878–883.

[7] X. Castillo, S. R. McConnel, and D. P. Siewiorek, "Derivation and calibration of a transient error reliability model," *IEEE Trans. Comput.*, vol. C-31, pp. 658–671, July 1982.

[8] M. L. Ciacelli, "Fault handling on the IBM 4341 processor," in *Proc. 11th Fault-Tolerant Computing Symp.*, Portland, ME, June 1981, pp. 9–12.

[9] B. Courtois, "Failure mechanisms, fault hypotheses and analytical testing of LSI-NMOS (HMOS) circuits," in *VLSI 81*, J. P. Gray, Ed. New York: Academic, 1981, pp. 341–350.

[10] R. P. Davidson, M. L. Harrison, and R. L. Wadsack, "BELLMAC-32: A testable 32 bit microprocessor," in *Proc. 1981 Int. Test Conf.*, Philadelphia, PA, Oct. 1981, pp. 15–20.

[11] E. A. Doyle, "How parts fail," *IEEE Spectrum*, vol. 18, pp. 36–43, Oct. 1981.

[12] A. D. Friedman and P. R. Memon, *Fault Detection in Digital Circuits*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[13] J. Galiay, Y. Crouzet, and M. Vergniault, "Physical versus logical fault models MOS LSI circuits: Impact on their testability," *IEEE Trans. Comput.*, vol. C-29, pp. 527–531, June 1980.

[14] R. T. Howard, "Packaging reliability: How to define and measure it," in *Proc. 32nd Electron. Components Conf.*, San Diego, CA, May 1982, pp. 376–384.

[15] J. Khakbaz and E. J. McCluskey, "Concurrent error detection and testing for large PLA's," *IEEE J. Solid-State Circuits*, vol. SC-17, pp. 386–394, Apr. 1982.

[16] G. P. Mak, J. A. Abraham, and E. S. Davidson, "The design of PLAs with concurrent error detection," in *Proc. 12th Fault-Tolerant Computing Symp.*, Santa Monica, CA, June 1982, pp. 303–310.

[17] M. A. Marouf and A. D. Friedman, "Efficient design of self-checking checker for any *m*-out-of-*n* code," *IEEE Trans. Comput.*, vol. C-27, pp. 482–490, June 1978.

[18] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.

[19] D. L. Ostapko and S. J. Hong, "Fault analysis and test generation for programmable logic arrays," *IEEE Trans. Comput.*, vol. C-28, pp. 617–627, Sept. 1979.

[20] R. A. Rasmussen, "Automated testing of LSI," *Computer*, vol. 15, pp. 69–78, Mar. 1982.

[21] C. H. Séquin, "Managing VLSI complexity: An outlook," *Proc. IEEE*, vol. 71, pp. 149–166, Jan. 1983.

[22] D. P. Siewiorek and D. Johnson, "Design methodology for high reliability systems: The Intel 432," in *The Theory and Practice of Reliable System Design*, D. P. Siewiorek and R. S. Swarz, Eds. Bedford, MA: Digital, 1982.

[23] K. Son and D.K. Pradhan, "Completely self-checking checkers in PLAs," in *Proc. 1981 Int. Test Conf.*, Philadelphia, PA, Oct. 1981, pp. 231–237.

[24] Y. Tamir and C.H. Séquin, "Self-checking VLSI building blocks for fault-tolerant multicomputers," *IEEE Int. Conf. Comput. Design*, Port Chester, NY, Nov. 1983, pp. 561–564.

[25] Y. Tamir, "Fault tolerance for VLSI multicomputers," Ph.D. dissertation, Electron. Res. Lab., Univ. Calif., Berkeley, in preparation.

[26] M. M. Tsao, A. W. Wilson, R. C. McGarity, C. Tseng, and D. P. Siewiorek, "The design of C.fast: A single chip fault tolerant microprocessor," in *Proc. 12th Fault-Tolerant Computing Symp.*, Santa Monica, CA, June 1982, pp. 63–69.

[27] R.L. Wadsack, "Fault modeling and logic simulation of CMOS and MOS integrated circuits," *Bell Syst. Tech. J.*, vol. 57, no. 5, pp. 1449–1474, May–June 1978.

[28] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*. Amsterdam: Elsevier North-Holland, 1978.

[29] S. L. Wang and A. Avizienis, "The design of totally self checking circuits using programmable logic arrays," in *Proc. 9th Fault-Tolerant Computing Symp.*, Madison, WI, June 1979, pp. 173–180.

Yuval Tamir (S'78) received the B.S.E.E. degree ("with highest distinction") from the University of Iowa, Iowa City, in 1979 and the M.S. degree in electrical engineering and computer science from the University of California, Berkeley, in 1981.

Since 1979 he has been a Research Assistant in the Electronics Research Laboratory, University of California, Berkeley, where he is currently working on his Ph.D. dissertation. His research interests are fault-tolerant computing, computer architecture, and distributed systems.

Mr. Tamir is a student member of the IEEE Computer Society and the Association for Computing Machinery.

Carlo H. Séquin (M'71–SM'80–F'82) received the Ph.D. degree in experimental physics from the University of Basel, Switzerland, in 1969.

His subsequent work at the Institute of Applied Physics, University of Basel concerned interface physics of MOS transistors and problems of applied electronics in the field of cybernetic models. From 1970 till 1976 he worked at the MOS Integrated Circuit Laboratory, Bell Telephone Laboratories, Murray Hill, NJ, on the design and investigation of charge-coupled devices for imaging and signal processing applications. He has written many papers in that field and is an author of the first book on charge-transfer devices. He spent the academic year 1976–1977 on a leave of absence with the University of California, Berkeley, where he lectured on integrated circuits, logic design, and microprocessors. In 1977 he joined the faculty of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, where he is now a Professor of Computer Science. His research interests include computer architecture and design tools for very large scale integrated systems. In particular, his research concerns multimicroprocessor computer networks, the influence of VLSI technology on advanced computer architecture, and the implementation of special functions in silicon. His most recent interests lie in the area of computer graphics and solids modeling. From 1980 till 1983, he was Head of the Computer Science Division of the Department of Electrical Engineering and Computer Science.

Dr. Séquin is a member of ACM and the Swiss Physical Society.