Comput. Math. Applic. Vol. 13, No. 5/6, pp. 555–566, 1987
Printed in Great Britain. All rights reserved

0097-4943/87 $3.00 + 0.00

TECHNIQUES FOR EFFICIENTLY IMPLEMENTING TOTALLY SELF-CHECKING CHECKERS IN MOS TECHNOLOGY†

N. K. JHA[1] and J. A. ABRAHAM[2]
[1]Department of EECS, University of Michigan, Ann Arbor, MI 48109, U.S.A.
[2]CSL, University of Illinois, Urbana, IL 61801, U.S.A.

Abstract—This paper presents some new techniques for reducing the transistor count of MOS implementations of totally self-checking (TSC) checkers. The techniques are (1) transfer of fanouts, (2) removal of inverters and (3) use of multi-level realizations of functions. These techniques also increase the speed of the circuit and may reduce the number of required tests. Their effectiveness has been demonstrated by applying them to m-out-of-n and Berger code checkers. Impressive reductions of up to 90% in the transistor count in some cases have been obtained for the MOS implementation of these checkers. This directly translates into saving of chip area.

1. INTRODUCTION

MOS technology has found extensive use in the area of very large scale integration (VLSI). In order to pack as many circuits on a single chip as possible, it is essential to reduce the area requirement of each circuit. One way to do this is to implement them with as few transistors as possible. This has spurred interest in finding efficient techniques for reducing the transistor count of MOS circuits. In this paper we will be concerned with a special class of circuits, called totally self-checking (TSC) circuits.

TSC circuits are used to detect errors concurrently with normal operation. These circuits operate on encoded inputs to produce encoded outputs. TSC checkers are used to monitor the outputs to indicate error when a non-code word is detected. The concept of TSC circuits was first proposed in [4], and then generalised in [3], as follows:

Definition 1. A circuit is *fault-secure* for a set of faults F, if for every fault in F, the circuit never produces an incorrect code output for code inputs.

Definition 2. A circuit is *self-testing* for a set of faults F, if for every fault in F, the circuit produces a non-code output for at least one code input.

Definition 3. A circuit is *totally self-checking* if it is fault-secure and self-testing.

Definition 4. A circuit is *code-disjoint* if it maps non-code inputs to non-code ouputs.

Definition 5. A circuit is a *totally self-checking checker* if it is self-testing and code-disjoint.

The correct operation of TSC circuits rests on following two assumptions:

(1) Faults occur one at a time.

(2) Sufficient time elapses between any two faults so that all the required code inputs can be applied to the circuit.

With these assumptions, the first erroneous output due to a fault in the TSC circuit must be a non-code word. This is referred to as the TSC goal.

Most existing realizations of TSC circuits are at the logic gate level, and assume a stuck-at fault model. It has been pointed out in [5] that an MOS implementation cannot be TSC with respect to unidirectional stuck-at faults (multiple lines stuck-at-1 or stuck-at-0, but not both). This is owing to a theorem in [16], which says that a circuit is TSC with respect to unidirectional stuck-at faults only if its realization is inverter-free. This is not possible in any MOS technology since every MOS gate is inverting. However, MOS implementations of TSC circuits can be made TSC with respect to single stuck-at faults. So for the purpose of this paper the fault-set will consist of all *single*

†This research was supported by the Semiconductor Research Corporation under contract SRC RSCH 84-06-049.

stuck-at faults. Another paper [8] shows how the MOS implementations can be made TSC with respect to realistic physical failures observed in the field, if certain layout rules are followed.

A direct implementation of existing TSC designs in MOS technology requires a high transistor count and, therefore, a large silicon area. We will present some techniques in this paper to reduce the transistor count, with the added advantages of speed-up of the circuit and possible reduction in the number of tests required. Note that reducing transistor count reduces area even further because of fewer interconnections. In [5] the technological cost (in case of nMOS technology) of different coding and checking circuits has been evaluated. We will show that by using our techniques one can obtain marked improvements over the results given in [5]. Although our techniques can be used, wherever applicable, for any MOS circuit, we will show their usefulness by applying them to TSC m-out-of-n [3, 6, 7, 10, 12–14] and Berger code [1, 11] checkers. Both these codes detect unidirectional errors. It should be kept in mind that even single stuck-at faults can produce unidirectional errors.

It is possible to realize a function which requires a multi-level logic gate realization by a single complex MOS gate. The achievable integration under a single complex MOS gate is limited by the following constraints.

(1) AND fan-in: maximum number of control transistors in any series path between the output node and ground.

(2) OR fan-in: the number of all possible conduction paths between the output node and ground.

(3) Presence of inverting functions.

(4) Presence of fanouts.

For simplicity, we will not consider the constraints due to the AND fan-in and OR fan-in in this paper. The techniques that we will present below can be shown to be easily extensible, if these constraints are considered.

2. TECHNIQUES FOR REDUCING THE COST OF TSC CHECKERS

2.1. *Transfer of fanouts*

If a function is shared by more than one MOS gate (i.e. when there is fanout), it becomes necessary to implement the function with a complex MOS gate. So fanouts do not allow us to take full advantage of the integration capability of MOS technology. In Example 1 below we will show how transfer of fanouts can help achieve greater integration. The techniques we develop in this paper are first applied to existing gate-level designs of TSC checkers,.thereby obtaining modified gate-level designs. These modified designs can then be implemented in MOS technology, resulting in a considerable decrease in their technological cost. Hence, most of the examples we present here will be of gate-level circuits, rather than their MOS implementations. We assume that an AND (OR) gate in a gate-level realization is implemented as a series (parallel) connection of transistors in a complex MOS gate. This complex MOS gate is followed by an inverter to get the non-inverting function. This will be clear from Example 1.

Example 1. The fanout in the circuit in Fig. 1a has been transferred in the circuit in Fig. 1b to the primary inputs. In this gate-level circuit the transfer of fanout increases the gate-count. But now let us look at the corresponding MOS implementations in Figs 2a and 2b. While the circuit in Fig. 2a requires 6 pull-up transistors and 9 control transistors for its MOS realization, the circuit

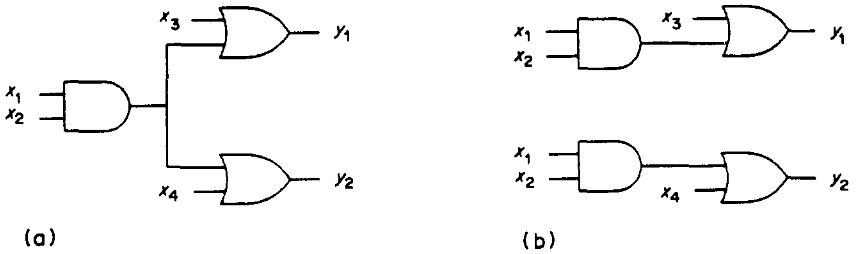

Fig. 1. (a) A gate-level circuit with fanout; (b) the circuit with fanout transferred to primary inputs.

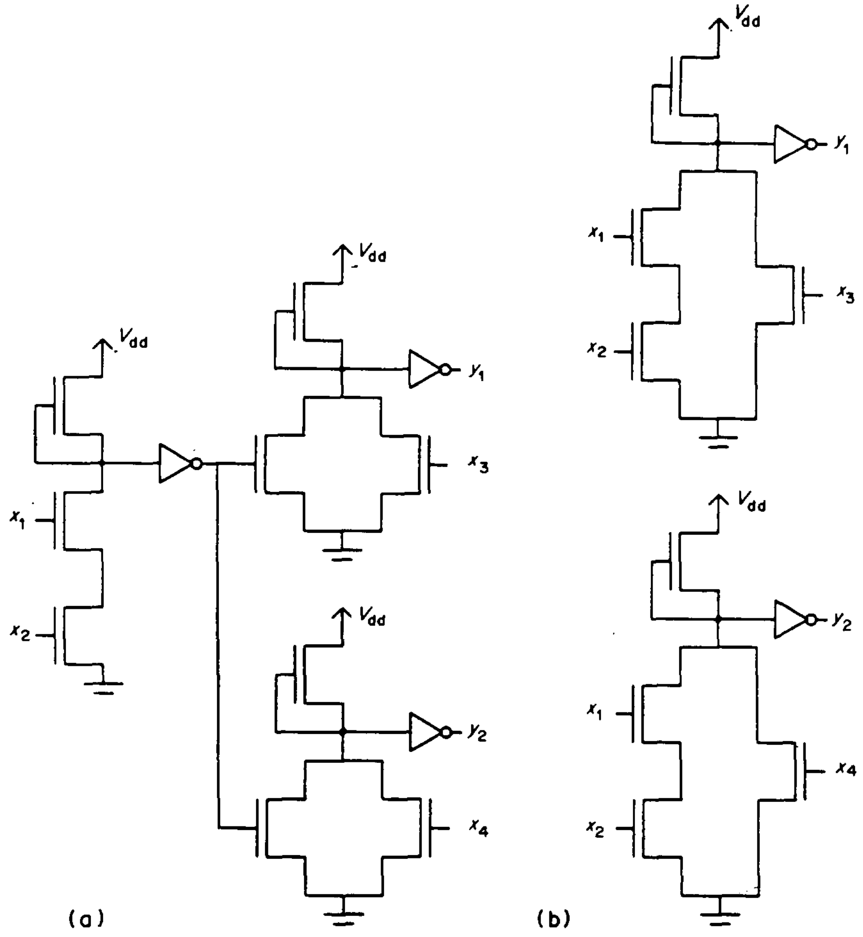

Fig. 2. (a) MOS implementation of the circuit in Fig. 1a; (b) MOS implementation of the circuit in Fig. 1b.

in Fig. 2b requires only 4 pull-up transistors and 8 control transistors. Thus transfer of this fanout helps reduce the transistor count of the circuit.

The transfer of fanouts can sometimes make the circuit redundant. Fortunately this is not true in general, and impressive reductions in transistor count can be achieved by judicious transfer of fanout. The transfer can also result in the loss of the self-testing property of a TSC circuit. So some conditions need to be developed to ensure that the self-testing property is maintained even after the transfer of fanouts. We will develop these conditions for m-out-of-n checkers. The same concept can be employed for other circuits as well. But before we do so, let us familiarize ourselves with some definitions and notation.

An r-input network is said to have 2^r vertices of the r-cube as possible input combinations. A particular vertex of the r-cube is written as

$$A = (a_1, a_2, \ldots, a_r) \quad \text{where} \quad a_i \in \{0, 1\}, \quad \text{for all} \quad i.$$

A vertex with exactly k ones is called a k-vertex.

Definition 6. The partial ordering on the vertices is defined as

$$A \leqslant B \quad \text{if} \quad a_i \leqslant b_i \quad \text{for all} \quad i.$$

For example,

$$(1, 1, 0, 0) \leqslant (1, 1, 0, 1)$$
$$(1, 0, 0, 1) \nleqslant (1, 1, 0, 0).$$

We will say that B covers A if $A \leqslant B$.

Definition 7. A universal test set is a set of tests which detects single stuck-at faults in any irredundant AND/OR logic gate-level realization of a given function [2].

The procedure for finding the universal test set is given in [2]. We will describe it in brief here.

Procedure 1

(1) Construct a truth table having one column for each literal that is present in the functional expression of f.

(2) Let X_j and X_l denote any two input vertices for which f is 1 and Y_k and Y_m denote any two input vertices for which f is 0. If

$$X_l \geqslant X_j$$

$$Y_k \geqslant Y_m$$

remove X_l and Y_m from the table.

(3) Repeat Step 2 until no more covering Xs or covered Ys remain. The set of input vertices left in the table is the universal test set for that function.

Theorem 1 below gives sufficient conditions for a set of m-out-of-n code words to be a universal test set. It should be kept in mind that an m-out-of-n codeword is nothing but an m-vertex.

THEOREM 1

In a mapping of an m-out-of-n code onto a 1-out-of-2 code (realized by functions y_1 and y_2) for a checker, if (1) each $(m+1)$-vertex covers some m-vertex mapping to $(1, 0)$ and some m-vertex mapping to $(0, 1)$ and (2) each $(m-1)$-vertex is covered by some m-vertex mapping to $(1, 0)$ and some m-vertex mapping to $(0, 1)$, then the set of m-out-of-n codewords is the universal test set for both y_1 and y_2.

Proof. From the monotone property of the circuit constructed with only AND/OR gates, and the code-disjoint property of the checker, all $(m+1)$ and higher vertices will map to $(1, 1)$ and all $(m-1)$ and lower vertices will map to $(0, 0)$. Since Condition (1) ensures that every $(m+1)$-vertex covers some m-vertex mapping to $(1, 0)$, from Procedure 1, we see that the $(m+1)$-vertices cannot belong to the universal test set for the function y_1; and since every $(m+1)$-vertex covers some m-vertex mapping to $(0, 1)$, the same is true for the function y_2. Similarly, from Condition (2), since every $(m-1)$-vertex is covered by some m-vertex mapping to $(1, 0)$, the $(m-1)$-vertices cannot belong to the universal test set for the function y_2; and since every $(m-1)$-vertex is also covered by some m-vertex mapping to $(0, 1)$ the same is true for the function y_1.

We have proved above that, given Conditions (1) and (2), any $(m+1)$-vertex or $(m-1)$-vertex cannot belong to the universal test set of either function y_1 or y_2. Now we will consider the $(m-2)$ and lower vertices. All such vertices map to $(0, 0)$. Due to the transitivity of the covering relation, for every $(m-2)$ or lower vertex X_i there exists an $(m-1)$-vertex X_j such that $X_i \leqslant X_j$. But from Condition (2), we have, $X_j \leqslant X_k$ and $X_j \leqslant X_l$, where X_k is some m-vertex mapping to $(1, 0)$ and X_l is some m-vertex mapping to $(0, 1)$. This implies that $X_i \leqslant X_k$ and $X_i \leqslant X_l$. So, from Procedure 1, any $(m-2)$ or lower vertex also cannot belong to the universal test set of either y_1 or y_2. A similar argument can be used to show that any $(m+2)$ or higher vertex cannot belong to the universal test set of either y_1 or y_2.

Since every m-vertex has the property that it does not cover any other m-vertex, only these m-vertices remain after executing Procedure 1 to find the universal test set for both y_1 and y_2. Q.E.D.

An alternative proof of Theorem 1 can be obtained by observing that the m-vertices are either minimal true-vertices or maximal false-vertices for y_1 and y_2, and hence, constitute the complete test set [15].

Theorem 1 is applicable to Smith's m-out-of-$2m$ checkers [16] since they are based precisely on the two conditions given in this theorem.

We will now focus our attention on Reddy's m-out-of-$2m$ checkers [14]. Reddy gave multi-level

Table 1. Classes of $(m + 1)$-vertices and m-vertices they cover

Possible classes of $(m + 1)$-vertices	Covered m-vertices belong to
$(m, 1)$	$(m, 0)$, $(m - 1, 1)$
$(m - 1, 2)$	$(m - 1, 1)$, $(m - 2, 2)$
$(m - 2, 3)$	$(m - 2, 2)$, $(m - 3, 3)$
—	—
—	—
—	—
$(2, m - 1)$	$(2, m - 2)$, $(1, m - 1)$
$(1, m)$	$(1, m - 1)$, $(0, m)$

Table 2. Classes of $(m - 1)$-vertices and the m-vertices that cover them

Possible classes of $(m - 1)$-vertices	Covering m-vertices belong to
$(m - 1, 0)$	$(m, 0)$, $(m - 1, 1)$
$(m - 2, 1)$	$(m - 1, 1)$, $(m - 2, 2)$
$(m - 3, 2)$	$(m - 2, 2)$, $(m - 3, 3)$
—	—
—	—
—	—
$(1, m - 2)$	$(2, m - 2)$, $(1, m - 1)$
$(0, m - 1)$	$(1, m - 1)$, $(0, m)$

cellular realizations for the checker functions y_1 and y_2 for an m-out-of-$2m$ checker as follows:

$$y_1 = \sum_{i=0}^{m} T(m_a \geqslant i) \cdot T(m_b \geqslant m - i), \quad i \text{ odd}$$

$$y_2 = \sum_{i=0}^{m} T(m_a \geqslant i) \cdot T(m_b \geqslant m - i), \quad i \text{ even.}$$

Here the input bits are divided into two groups A and B, each consisting of m bits. The number of ones occurring in the two groups is referred to as m_a and m_b respectively. For code inputs $m_a + m_b = m$.

We will say that a k-vertex belongs to a class represented by a 2-tuple (k_a, k_b) if it has k_a ones in group A and k_b ones in group B. Obviously, $k_a + k_b = k$.

Theorem 2

The set of m-out-of-$2m$ codewords is a universal test set for y_1 and y_2 functions of Reddy's checkers.

Proof. If we prove that Conditions (1) and (2) of Theorem 1 are satisfied by Reddy's checkers, then this theorem will follow.

We will first consider the $(m + 1)$-vertices. In the first column of Table 1 we give all the different classes of 2-tuples to whch any $(m + 1)$-vertex can possibly belong. The entries in the second column give the classes to which the m-vertices covered by that class of $(m + 1)$-vertices belong.

It can easily be verified that the m-vertices belong to the two different 2-tuples appearing in the second column of Table 1, for any given class of $(m + 1)$-vertices, map to the two different outputs $(0, 1)$ and $(1, 0)$. Hence, Condition (1) of Theorem 1 is satisfied.

Similarly, Table 2 can be formed for $(m - 1)$-vertices.

Following the same arguments as above, Condition (2) of Theorem 1 is seen to be satisfied. Hence, Theorem 2 follows from Theorem 1. Q.E.D.

The important implication of Theorem 2 is that any fanout in Reddy's m-out-of-$2m$ checkers can be transferred to some intermediate levels in the circuit. This does not make the circuit redundant, and allows us a lot of flexibility in the MOS implementation of these checkers. Since the set of m-out-of-$2m$ codewords forms a universal test set, we are assured that the circuit will remain self-testing even after the fanout transfer. It can be easily seen that the code-disjoint property is not affected by fanout transfer because the functions being implemented are still the same.

Example 2. If we transfer all the fanouts in Reddy's 5-out-of-10 checker to the primary inputs, its nMOS implementation requires 4 pull-up transistors and 116 control transistors. But another nMOS implementation, which leaves four of the fanouts untransferred, requires only 12 pull-up transistors and 80 control transistors. If we had implemented this checker without modification, it would require 40 pull-up transistors and 88 control transistors. So the savings in the transistor count is evident, as is the flexibility that this technique allows.

Another advantage of this technique is that it reduces the MOS gate levels (the number of complex or primitive MOS gates connected in series) from primary inputs to circuit outputs, and, hence, increases the speed of the circuit. On the other hand the disadvantage is that more tests are required to make the circuit self-testing. For example, for Reddy's checkers the number of required tests goes up from $2m$ to 2^m, if all fanouts are transferred to primary inputs. If all the fanouts are not transferred then the number of tests lies between these two limits. The 2^m tests, in case of full fanout, transfer, are the same as those required for Anderson's m-out-of-$2m$ checkers [3]. Anderson's m-out-of-$2m$ checkers do not have fanouts, hence this technique is not applicable to them.

Although this technique increases the number of required tests, when it is combined with the other two techniques given ahead, in most cases there is a considerable reduction in the number of tests. Now we go on to our second technique.

2.2. *Removal of inverters*

A function, which is shared by more than one MOS gate, is realised by a complex MOS gate followed by an inverter. We will show that these inverters can be eliminated in most cases, thereby reducing the transitor count and increasing the speed of the MOS implementation of the checker. In [5] it is mentioned that such inverters can be eliminated if the function realized at a fanout node can be replaced by its dual function without modifying the function realized by the circuit. We will not restrict ourselves to such functions.

Let $G(f)$ be an AND/OR realization of a function f and let $G_{\mathrm{dm}}(f)$ be the corresponding realization of f after using De Morgan's theorem. For example, if $G(f)$ realizes $f = x_1x_2 + x_3x_4$, then $G_{\mathrm{dm}}(f)$ realizes $f = \overline{(\bar{x}_1 + \bar{x}_2)(\bar{x}_3 + \bar{x}_4)}$.

Theorem 3

For any function f, $G(f)$ and $G_{\mathrm{dm}}(f)$ have the same single stuck-at fault test set.

Proof. Let the realization obtained by placing two inverters on every line of $G(f)$ be $G_i(f)$. The stuck-at fault test set for $G_i(f)$ is the same as that for $G(f)$, because the path sensitization of the stuck-at faults by their corresponding tests remains unaffected. Furthermore, we can see that the corresponding lines in the two gates in Figs 3a and 3b require the same test set for detecting the presence of stuck-at faults.

The two gates in Figs 3a and 3b can therefore be interchangeably used in the circuit $G_i(f)$ without changing its test set. It can easily be seen that $G_{\mathrm{dm}}(f)$ can be obtained from $G_i(f)$ by these interchanges of the two gates. But since $G_i(f)$ has the same test set as $G(f)$, $G_{\mathrm{dm}}(f)$ and $G(f)$ will also have the same test set. Q.E.D.

This theorem implies that by converting $G(f)$ to $G_{\mathrm{dm}}(f)$ we do not affect the self-testing property of the circuit.

Example 3. Consider Marouf–Friedman's design of a 2-out-of-5 checker [10] given in Fig. 4. This circuit can be converted to the circuit in Fig. 5 without affecting its TSC property with respect to single stuck-at faults, with the resultant saving of ten inverters in the MOS implementation. It is clear that the two inverters at the circuit outputs can also be done away with since the pair $(\bar{y}_1, \bar{y}_2)$ would also form a 1-out-of-2 code, if the pair (y_1, y_2) forms a 1-out-of-2 code.

This technique also has the added advantage that it speeds up the MOS implementation by reducing the MOS gate levels from circuit inputs to outputs. In Example 3, the number of MOS gate levels in the MOS implementation of the circuit in Fig. 4 is six, while for the MOS implementation of the circuit in Fig. 5 it is three. Hence, the speed is roughly doubled by using this technique for this example. This technique is applicable to designs in [6, 7, 10, 13, 16].

Now we continue on to our third technique.

2.3. *Increasing the logic gate levels of a circuit before its MOS implementation*

It is possible to reduce the transistor count of an MOS implementation by modifying the expression of the function it implements. This has been discussed in [8]. It was shown in that paper that it is desirable to first reduce the number of literals in the expression of a function before implementing it in an MOS technology. This basically corresponds to increasing the number of

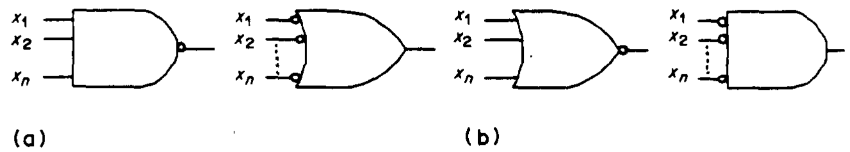

Fig. 3. (a) A NAND gate and its equivalent; (b) A NOR gate and its equivalent.

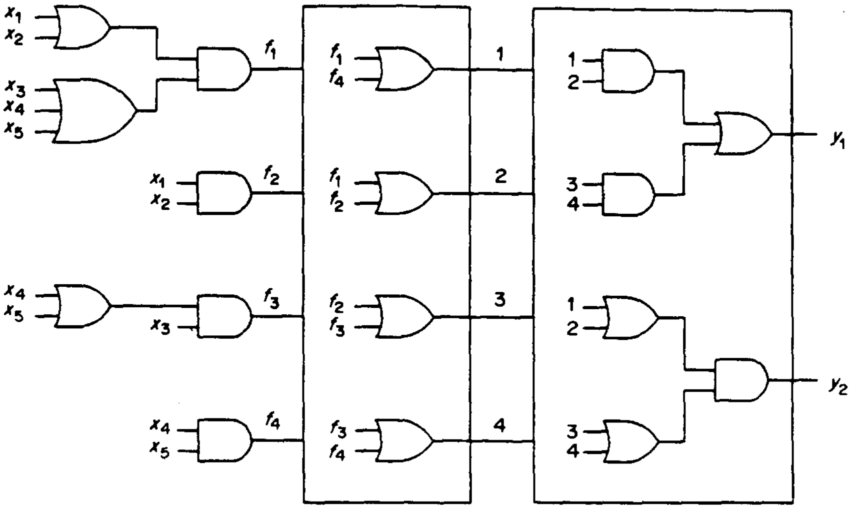

Fig. 4. Marouf–Friedman's 2-out-of-5 checker.

logic gate levels in the gate-level design of that function. In case of gate-level designs this would have amounted to reducing the speed of the circuit. But, as was explained in [8], the speed of the MOS implementation is not degraded by using this technique (if at all, the circuit becomes slightly faster). This is because the number of MOS gate levels from circuit inputs to outputs remains unchanged. Also, the diffusion area required on the chip for implementing the control transistors is reduced because fewer control transistors are required if this technique is used. Since the speed of the complex MOS gate primarily depends on the capacitive load generated at the output node by the diffusion conduction paths, we actually have a speed enhancement for the MOS implementations. Another advantage is that the number of tests required to test the circuit is considerably reduced.

We will show the usefulness of this technique by applying it to MOS implementations of majority functions. Majority functions (defined below) are extensively used in the design of m-out-of-n code checkers.

Maximum level realizations of majority functions. We will first define a majority function. Let A be the set of input bits, and let n_a and k_a represent the number of bits and the number of ones in the set respectively. The majority function $T(k_a \geqslant i)$, defined on set A, has a value 1 iff the condition inside the parentheses is true. For example, if $A = (a_1, a_2, a_3, a_4)$, then a functional

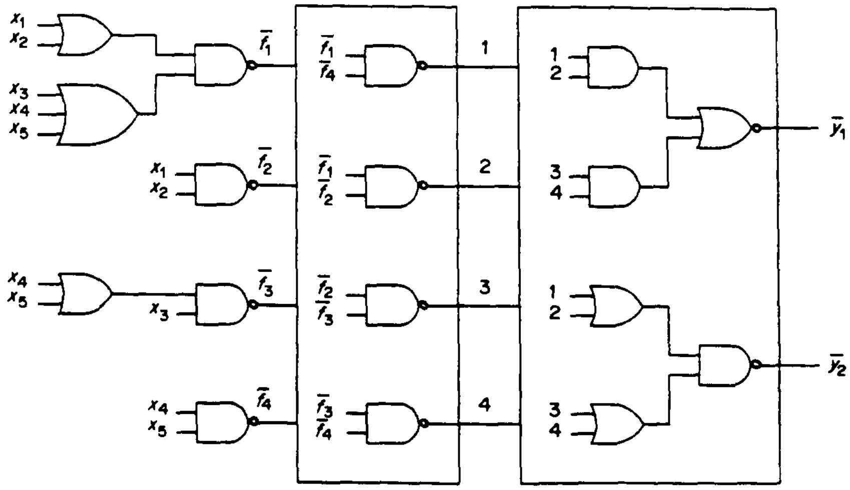

Fig. 5. Marouf–Friedman's 2-out-of-5 checker after using our technique.

expression of the majority function $T(k_a \geqslant 2)$ can be given as

$$T(k_a \geqslant 2) = a_1a_2 + a_1a_3 + a_1a_4 + a_2a_3 + a_2a_4 + a_3a_4. \qquad (1)$$

This is a two level realization of $T(k_a \geqslant 2)$. Usually the m-out-of-n code checker designs employ majority function realizations with minimum number of levels due to speed considerations. But as was mentioned above, this thinking is no longer valid for MOS circuits, and one should use minimum literal or maximum level realizations to reduce the transistor count. An efficient method of obtaining maximum level majority function realization is given in [9, 13]. Using this method an alternate functional expression can be obtained for $T(k_a \geqslant 2)$ as follows:

$$T(k_a \geqslant 2) = a_1a_2 + (a_1 + a_2)(a_3 + a_4) + a_3a_4. \qquad (2)$$

This equation gives the maximum level, minimum literal realization of $T(k_a \geqslant 2)$. Since the number of control transistors in the MOS implementation has one-to-one correspondence with the number of literals in the functional expression, we see that the transistor count, from (1) to (2), has been reduced by four.

One question still remains, however; that is, will the stuck-at fault test set of a two-level realization of any function (not just the majority function) still be valid for its multi-level realization. We given a theorem below to show that this indeed is the case.

Theorem 4

The stuck-at fault test set S of a two-level realization suffices as a test set for any multi-level realization of that function.

Proof. We will prove the theorem for the AND–OR realization. Dual arguments can be applied to the OR–AND realization.

Let $E_1(f)$ and $E_2(f)$ denote the functional expressions of the two-level AND–OR realization and the multi-level realization of a function f respectively. $E_1(f)$ is basically an irredundant sum of products expression of the function f.

If we expand $E_2(f)$ by removing the parentheses, we arrive at $E_1(f)$. Conversely, we can factor out common literals or sub-expressions from the terms of $E_1(f)$ and the other intermediate expressions to arrive at $E_2(f)$. The process of conversion repeatedly applied, going from $E_1(f)$ to $E_2(f)$, is illustrated by Figs 6a and 6b.

x_i, $i \in \{1, 2, 3\}$, in the circuits in Figs 6a and 6b can be a literal or a sub-expression. It can easily be seen that the stuck-at fault test set for the circuit in Fig. 6a suffices as a stuck-at fault test set for the circuit in Fig. 6b. Since $E_2(f)$ is obtained by repeatedly applying the above conversion at different levels, the theorem follows as a straightforward generalization. Q.E.D.

The test set S is sufficient, but not necessary, for making the multi-level realization self-testing. Usually a subset of S is required for this purpose. Hence, this technique reduces the number of tests as well as the transistor count when the circuit is implemented in MOS technology. Additionally, it also slightly increases the speed of the circuit, as explained before. We can apply this technique to designs in [3, 6, 7, 9, 10, 13, 16].

3. EFFICIENT MOS IMPLEMENTATION OF BERGER CODE CHECKERS

The Berger code is a separable code used to detect unidirectional errors. Designs of gate level Berger checkers are given in [1, 11]. These designs basically employ a combinational circuit which

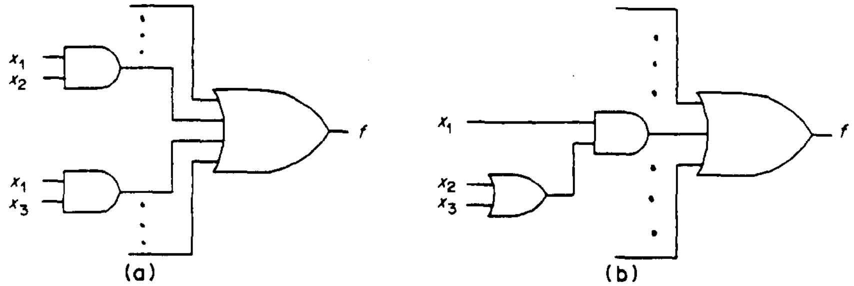

Fig. 6. (a) Two-level realization of a function f; (b) multi-level realization of f.

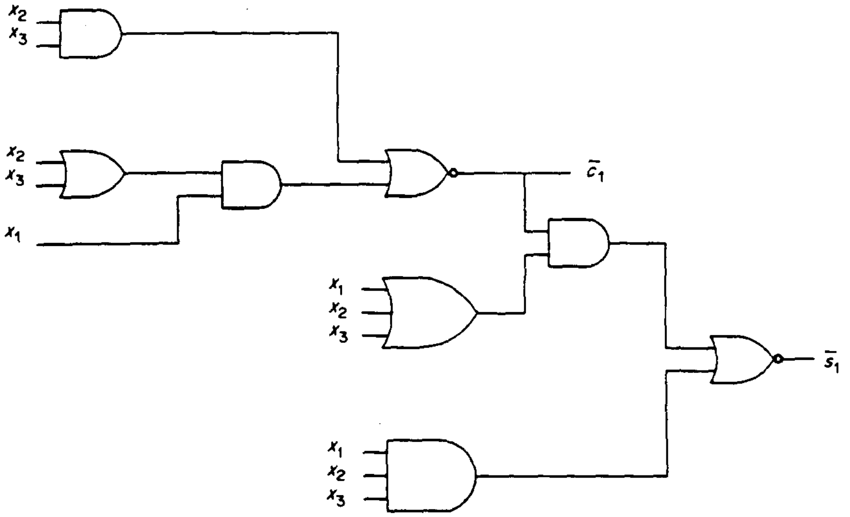

Fig. 7. An efficient realization of a two-bit adder.

generates the complement of the checkbits. This circuit is realized with full adders and half-adders. The outputs of this combinational circuit, together with the checkbit lines from the primary inputs, are fed to a two-rail code checker. The two outputs of the two-rail code checker indicate error when a fault occurs.

In [5] an efficient way of implementing full adders and half-adders in MOS technology is given. The transistor count n_T (for nMOS technology) for the Berger code checker, on the basis of these implementations, is given as

$$n_T = 23(n_I - n_k) + 10(n_k - 1) = 23n_I - 13n_k - 10 \tag{3}$$

where n_I = number of information bits and n_k = number of checkbits.

The number $23(n_I - n_k)$ refers to the transistor count of the checkbit complement generator and $10(n_k - 1)$ to the transistor count of the two-rail code checker.

We will now present a more efficient way of implementing Berger code checkers. Figure 7 shows a gate level realization of the two-bit adder (with outputs complemented), whose MOS implementation requires only 14 transistors (12 control and 2 load transistors). Figure 8 shows a gate level realization of a two-bit adder for obtaining uncomplemented outputs when complemented inputs are available. This also requires 14 transistors for its MOS implementation. These are more efficient MOS implementations than the one given in [5], which requires 23 transistors for a two-bit adder with uncomplemented inputs and outputs.

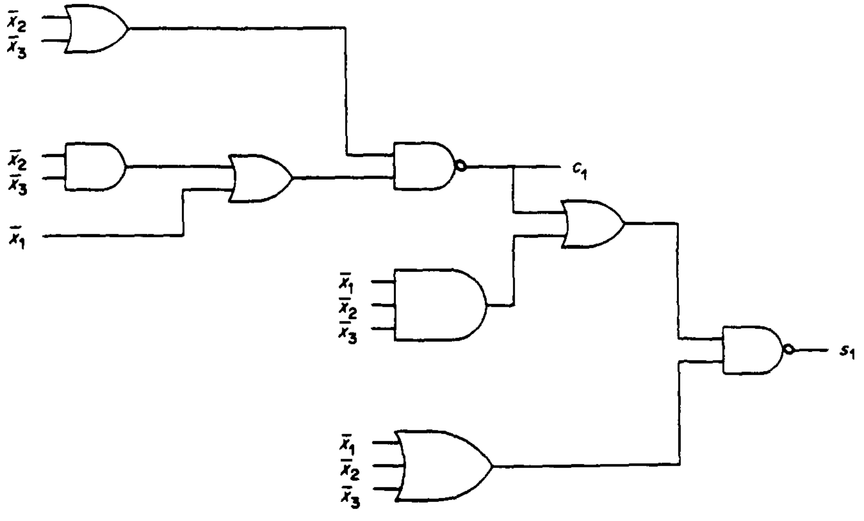

Fig. 8. An alternative realization of a two-bit adder.

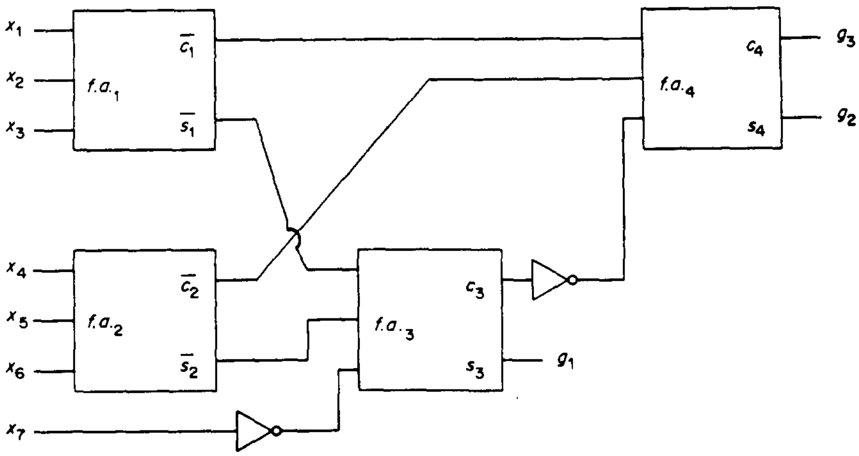

Fig. 9. An efficient checkbit complement generator for $n_I = 7$.

Let us consider a Berger code with $n_I = 7$. We know that for this code, $n_k = 3$. Figure 9 shows the combinational circuit for generating the complement of the three checkbits when the technique for removal of inverters is used, and efficient realizations given in Fig. 7 and 8 are used for the two-bit adders. This circuit is combined with a two-rail code checker to obtain a Berger code checker when $n_I = 7$. We get the following equation for the transistor count of the Berger code checker for the general case:

$$n_T = 14(n_I - n_k) + 10(n_k - 1) + m = 14n_I - 4n_k - 10 + m \tag{4}$$

where m is a constant required to account for some inverters that remain in the circuit. For the circuit in Fig. 9, $m = 4$, because two inverters are still present in it.

For the Berger code checker considered above, for $n_I = 7$ and $n_k = 3$, we get $n_T = 112$ from (3). If our techniques are used, however, we get $n_T = 80$, from (4). Hence, we have reduced the transistor count by about 29%. This is roughly the reduction to be expected for most Berger code checkers. Of course, the added advantage of speed-up of the circuit due to the removal of the inverters is also obtained by using our technique.

4. MOS CHECKER COST FOR m-OUT-OF-n CODES

In Section 2 we presented three techniques which are individually or collectively applicable to all the known designs of m-out-of-n code checkers. We present some tables to show the reduction in the transistor count of the nMOS realizations of some of these checkers. $L1$ and $C1$ refer to the number of load and control transistors respectively if the m-out-of-n code checker designs are implemented directly in nMOS technology. $L2$ and $C2$ refer to the number of load and control transistors respectively if the techniques in Section 2 are used.

$$\% \text{ reduction in transistor count} = \frac{(L1 + C1 - L2 - C2)}{(L1 + C1)} \times 100$$

For simplicity the AND fan-in and the OR fan-in restrictions have not been taken into account. Looking at Tables 3–8, we can conclude that for m-out-of-$2m$ codes for small m, Anderson's design [3] is the best from the point of view of minimum chip area requirement; while for

Table 3. Transistor counts for Smith's m-out-of-$2m$ code checkers

Code	$L1$	$C1$	$L2$	$C2$	% Reduction
3/6	16	32	2	24	46
4/8	36	66	10	48	43
5/10	64	112	12	80	48
10/20	324	522	82	360	48

Table 4. Transistor counts for Reddy's m-out-of-$2m$ code checkers

Code	$L1$	$C1$	$L2$	$C2$	% Reduction
3/6	12	28	2	22	40
4/8	24	54	2	52	31
5/10	40	88	10	90	22
10/20	180	378	74	364	22

Table 5. Transistor counts for Anderson's m-out-of-$2m$ code checkers

Code	$L1$	$C1$	$L2$	$C2$	% Reduction
3/6	4	26	2	22	20
4/8	4	66	2	48	29
5/10	4	162	2	98	40
10/20	4	10242	2	1176	89

Table 6. Transistor counts for Marouf–Friedman's m-out-of-n code checkers

Code	$L1$	$C1$	$L2$	$C2$	% Reduction
2/5	20	38	10	28	34
3/7	20	63	10	48	30
3/8	24	84	12	64	29
3/10	24	116	12	86	30
4/9	20	126	10	84	31
5/11	20	240	10	137	43
10/21	20	16346	10	1464	91

Table 7. Transistor counts for Piestrak's m-out-of-n code checkers

Code	$L1$	$C1$	$L2$	$C2$	% Reduction
3/7	26	52	15	40	30
3/8	30	62	16	46	33
3/10	42	84	26	66	27
4/9	38	83	25	67	24
5/11	52	116	35	94	23
10/21	132	383	95	326	18

Table 8. Transistor counts for Gaitanis–Halatsis' m-out-of-n code checkers

Code	$L1$	$C1$	$L2$	$C2$	% Reduction
2/5	24	42	15	33	27
3/7	32	67	17	48	34
3/8	32	75	18	56	31
3/10	36	102	20	76	30
4/9	38	98	19	73	32
5/11	42	151	21	107	34
10/21	62	1020	31	625	40

m-out-of-$2m$ codes for large m, one could choose either Smith's [16] or Reddy's [14] design. It is difficult to say which design is better until the actual layout is done for each individual case.

For smaller m-out-of-n ($n \neq 2m$) codes there is not much of a difference among the three designs considered in Tables 6–8. For larger m-out-of-n codes Piestrak's [13] design seems to be the best from the chip area considerations. We have not given tables for the other two known designs for m-out-of-n codes, namely Nanya–Tohma's design [12] and Efstathiou–Halatsis' design [6], because these designs require much higher transistor counts compared to the three designs for which tables were presented here. Hence, they are not of practical interest from the point of view of MOS implementation.

5. CONCLUSION

In this paper we presented some techniques to reduce the transistor count of MOS implementations of TSC checkers. The reduction ranges from about 20 to over 90%. The resultant decrease in silicon area will be even more due to reduced routing. Other advantages of using these techniques are the speed-up of the circuit due to a reduction in the MOS gate levels and a possible reduction in the number of tests required to make the circuit self-testing. It should be noted that the application of these techniques is not limited to TSC checkers, but they can be applied to any general circuit whose MOS implementation is required. In this paper they were applied to TSC checkers to establish their effectiveness.

REFERENCES

1. M. J. Ashajee and S. M. Reddy, On totally self-checking checkers for separable codes. *IEEE Trans. Comput.* **C26,** 737–744 (1977).
2. S. B. Akers, Universal test sets for logic networks. *IEEE Trans. Comput.* **C22,** 835–839 (1973).
3. D. A. Anderson and G. Metze, Design of totally self-checking check circuits for *m*-out-of-*n* codes. *IEEE Trans. Comput.* **C22,** 263–269 (1973).
4. W. C. Carter and P. R. Schneider, Design of dynamically checked computers. *IFIP '68, Edinburgh, Scotland,* Vol. 2, pp. 878–883 (1968).
5. Y. Crouzet and C. Landrault, Design of self-checking MOS–LSI circuits: application to a four-bit microprocessor. *IEEE Trans. Comput.* **C29,** (1980).
6. C. Efstathiou and C. Halatsis, Modular realization of totally self-checking checkers for *m*-out-of-*n* codes. *13th Int. Symp. Fault-Tolerant Computing*, pp. 154–161 (1983).
7. N. Gaitanis and C. Halatsis, A new design method for *m*-out-of-*n* TSC checkers. *IEEE Trans. Comput.* **C32,** 273–283 (1983).
8. N. K. Jha and J. A. Abraham, Totally self-checking MOS circuits under realistic physical failures. *Int. Conf. Computer Design*, Port Chester, New York (1984).
9. G. P. Mak, The design of programmable logic arrays with concurrent error detection. Ph.D. Thesis, Univ. of Illinois, Urbana, Illinois (1982).
10. M. A. Marouf and A. D. Friedman, Efficient design of self-checking checker for any *m*-out-of-*n* code. *IEEE Trans. Comput.* **C27,** 482–490 (1978).
11. M. A. Marouf and A. D. Friedman, Design of self-checking checkers for Berger codes. *8th Int. Symp. Fault-Tolerant Computing*, pp. 179–184 (1978).
12. T. Nanya and Y. Thoma, A 3-level realization of totally self-checking checkers for *m*-out-of-*n* codes. *13th Int. Symp. Fault-Tolerant Computing*, pp. 173–176 (1983).
13. S. Piestrak, Design method of totally self-checking checkers for *m*-out-of-*n* codes. *13th Int. Symp. Fault-Tolerant Computing*, pp. 162–168 (1983).
14. S. M. Reddy, A note on self-checking checkers. *IEEE Trans. Comput.* **C23,** 1100–1102 (1974).
15. S. M. Reddy, Complete test sets for logic functions. *IEEE Trans. Comput.* **C22,** 1016–1020 (1973).
16. J. E. Smith, The design of totally self-checking check circuits for a class of unordered codes. *J. Des. Automn Fault-Tolerant Comput.* **2,** 321–342 (1977).