SEL-73-039

Partially Self-Checking Circuits and Their Use in Performing Logical Operations

by

John F. Wakerly

August 1973

Technical Report No. 50; ISSUED IN JULY 1974AS
COMPUTER SCIENCE DEPARTMENT TECHNICAL REPORT NO. 420.

This research was performed while Mr. Wakerly was a Fannie and John Hertz Foundation Fellow; it was also partially supported by National Science Foundation Grant GJ-27527.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY · STANFORD, CALIFORNIA

SEL 73-039

PARTIALLY SELF-CHECKING CIRCUITS

AND THEIR USE IN PERFORMING LOGICAL OPERATIONS

by

John F. Wakerly

August 1973

Technical Report no. 50

DIGITAL SYSTEMS LABORATORY

Dept. of Electrical Engineering Dept. of Computer Science

Stanford University

Stanford, California

This research was performed while Mr. Wakerly was a Fannie and John Hertz Foundation Fellow; it was also partially supported by National Science Foundation Grant GJ-27527.

ABSTRACT

A new class of circuits called partially self-checking circuits is described. These circuits have one mode of operation called secure mode in which they have the properties of totally self-checking circuits; that is, every fault is tested during normal operation and no fault can cause an undetected error. They also have an insecure mode of operation with the property that any fault which affects a result in insecure mode is tested by some input in secure mode; however, undetected errors may occur in insecure mode. One application of these circuits is in the arithmetic and logic unit of a computer with data encoded in an error-detecting code. While there is no code simpler than duplication which detects single errors in logical operations such as AND and OR, it is shown that there exist partially self-checking networks to perform these operations. A commercially available MSI chip, the 74181 4-bit ALU, can be used in a partially self-checking network to perform arithmetic and logical operations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENT

The author expresses appreciation for the helpful suggestions and advice of Professor Edward J. McCluskey during the course of this work, and for the support of the Fannie and John Hertz Foundation.

1. INTRODUCTION

One approach to error detection in fault-tolerant computers is through the use of self-checking circuits, explored by Carter and Schneider [1] and also by Anderson [2]. As suggested by Fig. 1.1, the output of a self-checking circuit is encoded in some error-detecting code so that faults may be detected by a checker which monitors the output and signals the appearance of a non-code word. A self-checking circuit has properties of "self-testing" and "fault-secureness" introduced in [1] and formally defined by Anderson [2].

Definition A1: A circuit is self-testing if, for every fault from a prescribed set, the circuit produces a non-code space output for at least one code space input.

Definition A2: A circuit is fault-secure if, for every fault from a prescribed set, the circuit never produces an incorrect code space output for code space inputs.

Anderson's definitions imply the existence of a "code space" from which normal inputs are drawn, and for which the circuit is both self-testing and fault-secure. This facilitates his definition of a "totally self-checking circuit," a circuit which is both self-testing and fault-secure. Actually, a circuit may be self-testing for the set of normal code space inputs, but fault-secure for only a subset. In this report we formulate a theory of self-checking circuits that are self-testing for an input set N and fault-secure for a subset I of N. If I equals N, the

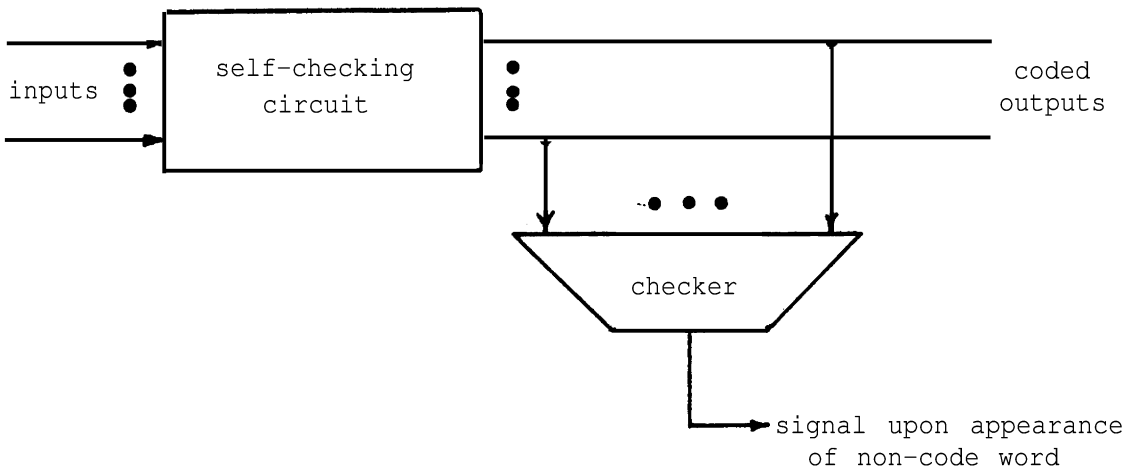


Fig. 1.1 Self-checking circuit

circuit is **totally self-checking** as described in [2]. If I is the null set, we have a circuit which is only self-testing and not at all **fault-secure**, such as the self-testing decoder described by Carter et. al. [3]. If I is a non-null proper subset of N, then we have a "partially **self-checking** circuit," as described in this report.

Due to the fact that no code short of duplication can be used to check the logical operations AND and OR [4], any totally self-checking circuit for these operations must use a form of duplication. For example, the JPL STAR computer uses duplicate logic units [5], while a processor designed by Monteiro and Rao duplicates the AND operation and uses a combination of AND and arithmetic operations to perform the other logical operations in a self-checking manner [6]. However, we will show how partially self-checking circuits using inexpensive codes may be used to perform logical operations. These circuits have one mode of operation in which they are fault-secure, and another mode, performing logical operations, in which they are not.

2. SELF-CHECKING CIRCUITS

Throughout this paper we will consider a combinational circuit to produce an output vector **Z(i,f)** which is a function of an input vector i and a fault f in the circuit. For our purposes a fault is a malfunction which is manifested as one or more lines in a circuit stuck at a logic value of 0 or 1. For example, we have the single fault <b/0> ("line b stuck-at-o") and the multiple fault **<a/1**, b/1, d/0>. The absence of a malfunction is called the null fault and denoted by **λ**. An error occurs when an incorrect value appears at the output of a circuit because of a fault. Associated with a circuit is an output code space S; a checker may monitor the output of the circuit and produce an indication when an output not in S appears. There is a set of normal inputs N, those inputs which occur periodically during fault-free operations of the system. The fault-free output function **Z(i,λ)** is a mapping from N into S. We will also associate with a circuit two fault sets, $\mathbf{F}_t$ and F_s, which are used in the definitions below.

Definition: A circuit is self-testing for a fault set $\mathbf{F}_t$ if for every f in $\mathbf{F}_t$ there is an input i in N such that Z(i,f) is not in S.

The definition of self-testing is illustrated in Fig. 2.1. In this definition, an input i for which **Z(i,f)** is not in S is called a test for f. The set $\mathbf{F}_t$ of faults which are tested during normal operation is called the tested fault set.

Definition: A circuit is fault-secure for an input set I and a fault set F_s if for any i in I and for any f in Fs either $Z(i,f) = Z(i,\lambda) \in S$ or $Z(i,f) \notin S$.

Fig. 2.2 illustrates the above definition. The set I is called the secure input set. We will always assume that I is a subset of the normal input set N. Although the circuit may be fault-secure for some inputs outside of N, these inputs are not of interest since they do do occur in normal operation.

The set F_s above is called the secure fault set. We will always assume for convenience that F_s is a subset of the tested fault set F_t. For suppose there is a fault f in F_s which is not in F_t. Then there is no input among all the normal inputs for which an erroneous output is produced in the presence of f, and the fault is not an interesting one to consider. (However, multiple faults including f as a component may be of interest.)

The properties of self-testing and fault-secureness are illustrated in Fig. 2.3. This figure shows the set of all faults and its subsets F_t and F_s, the set of all input vectors and its subsets N and I, and the set of all output vectors and its subset S. In the absence of faults, inputs from N produce outputs in S, as shown by the behavior in i_1, i_2, and i_3. Self-testing is shown by noting that for each of the faults f_1, f_2, and f_3 in F_t there is a test in N (i_1, i_3, and i_1 respectively). Fault-secureness is illustrated by the behavior of $Z(i_2,f)$ for various f. In the presence of a fault from F_s, the output is either correct $(Z(i_2,f_2))$ or it is a non-code word $(Z(i_2,f_1))$. However, faults outside of F_s may produce erroneous code word outputs $(Z(i_2,f_3))$. Circuits which are self-testing and fault-secure for some sets of inputs and faults are self-checking.

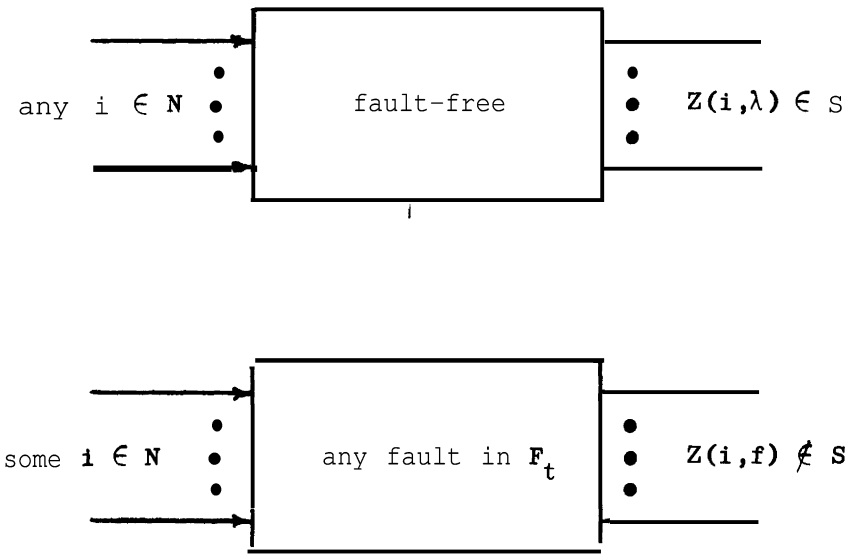


Fig. 2.1 Self-testing circuit

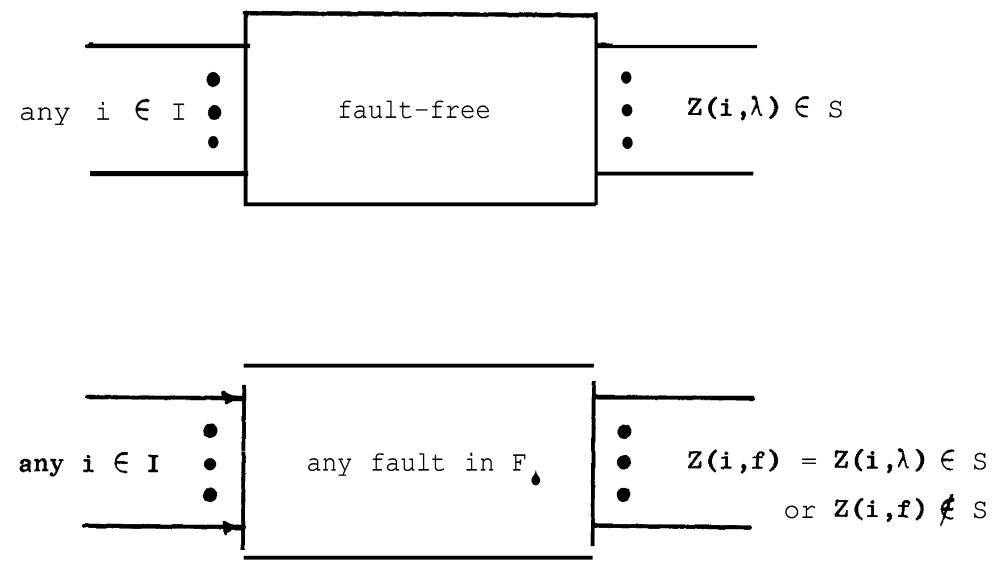
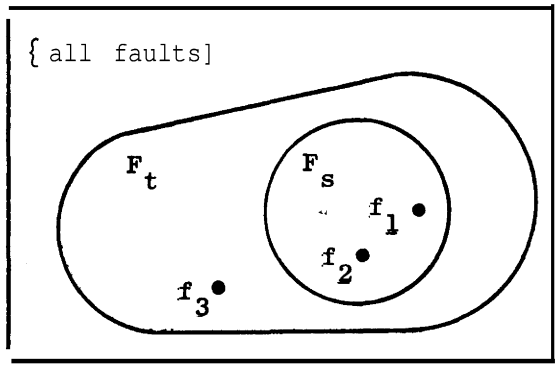


Fig. 2.2 Fault-secure circuit



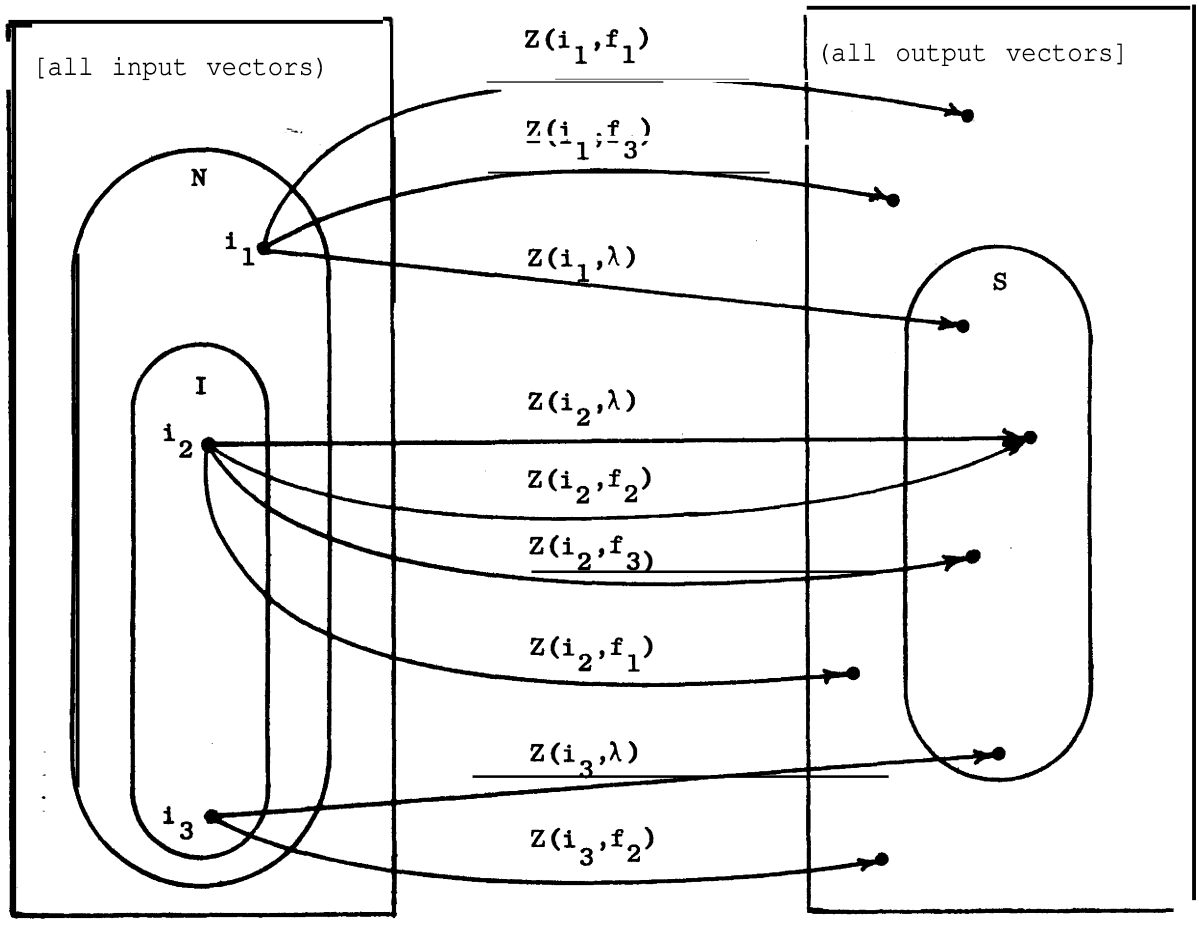


Fig. 2.3 Examples of self-testing and fault-secureness

Definition: A combinational circuit with normal input set N and output code space S is self-checking if it is self-testing for a fault set F_t and fault-secure for an input set I and fault set F_s.

For a self-checking circuit to be of any value, F_t and F_s should be reasonable fault sets, containing say all the single stuck-at faults.

During normal operation of a self-checking circuit, all reasonable faults are detected because of the self-testing property. In addition, fault-secureness guarantees there is no undetected erroneous output when inputs are from I. If I is equal to N, then the circuit is "totally self-checking."

Definition; A totally self-checking circuit is a self-checking circuit for which the set I of secure inputs equals the set N of normal inputs.

In a totally self-checking circuit, no fault in F_s can cause an undetected error for any normal input to the circuit. At the other extreme are circuits for which there is no non-null choice of I for which the circuit is fault-secure.

Definition: A self-testing circuit is a self-checking circuit for which the set I of secure inputs is the null set.*

An example of a self-testing circuit is the self-testing decoder of Carter et. al. [3]. For any input to this circuit there is a single

*Obviously self-testing circuits may also be defined without reference to self-checking circuits. However, this definition is included for consistency and completeness.

stuck-at fault which will cause an erroneous code word output, and thus I must be the null set.

Between the two extremes of self-testing and totally self-checking circuits are partially self-checking circuits.

Definition: A partially self-checking circuit is a self-checking circuit for which the set I of secure inputs is a non-null proper subset of the set N of normal inputs.

When inputs to a self-checking circuit are from I, the circuit is said to operate in secure mode. A totally self-checking circuit always operates in secure mode. When inputs are from the set I' = N - I, the circuit operates in insecure mode. A self-testing circuit always operates in insecure mode. A partially self-checking circuit operates sometimes in one mode, sometimes in the other.

The effectiveness of totally and of partially self-checking circuits may now be compared. With a totally self-checking circuit, any output which is in the code space is correct if no faults outside of F_s occur, and any fault in F_s is detected by the first error it produces. If only faults from F_s occur, no erroneous results may be transmitted. In secure mode, a partially self-checking circuit has these same desirable properties. But in insecure mode, erroneous results may be transmitted.

The likelihood of an undetected error in insecure mode is proportional to the frequency of operation in this mode. If this mode is infrequent, chances are that a fault will be detected in secure mode before any result in insecure mode is affected. Even when a solid fault produces an undetected error in insecure mode, it will soon be detected

in secure mode. At this point a software rollback scheme might be used to erase the effect of possible undetected errors.

Unfortunately, there is still..a chance in insecure mode of transmitting errors caused by short transient faults that are never detected. Although this possibility is very small, it may be sufficient to rule out the use of partially self-checking circuits in highly critical applications where ultra-reliability is required and the chance of transients is high. But for less critical applications, partially **self-**checking circuits can provide a good deal of low-cost error detection in areas where corresponding totally self-checking circuits are much more expensive. In particular, we will show networks for logical operations which are partially self-checking, but first we introduce a model of totally self-checking networks.

3. TOTALLY SELF-CHECKING CIRCUITS AND NETWORKS

In dealing with totally self-checking circuits we will mention only the set N of normal inputs because the set I of secure inputs is the same. A trivial example of a totally self-checking circuit is a bus driver for n-bit parity-encoded operands, illustrated in Fig. 3.1. The circuit consists simply of n identical bus driver gates (one-input AND gates), one for each output bit. The output code space S and the normal input set N both equal the set of all even-parity n-bit vectors. The circuit is fault-secure for all single faults, since a single fault causes either no error for a particular input, or a distance-one change in the output producing an odd-parity vector. The circuit is also **self-testing** for all stuck-at faults which affect less than n bits, since for any such fault there is an even-parity input vector which produces an odd-parity output in the presence of the fault. A checker which produces a signal when an odd-parity vector appears may be used to monitor the output of the circuit, as suggested by Fig. 1.1. Actually, we would like the checker also to be totally self-checking so that a fault in the checker also produces an error indication. This leads us to the concept of totally self-checking networks.

Anderson gives the model of Fig. 3.2 of a totally self-checking network consisting of a functional circuit and a checker which are both totally self-checking [2]. In terms of the notation presented here, the functional circuit has a fault-free output function which is a **surjection** from a normal input set $\mathbf{N_f}$ onto an output code space $S_{f'}$ while the checker

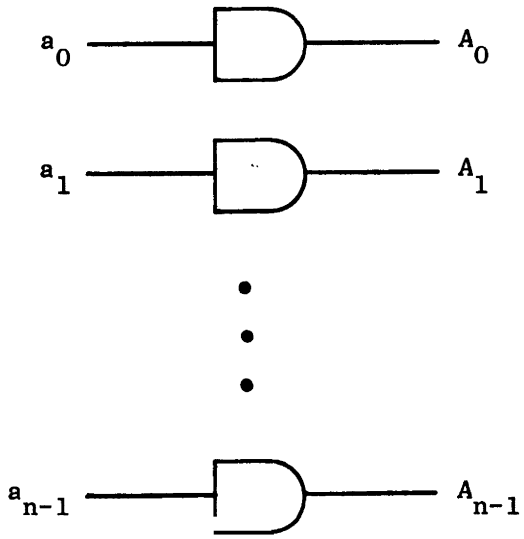


Fig. 3.1 Totally self-checking bus driver

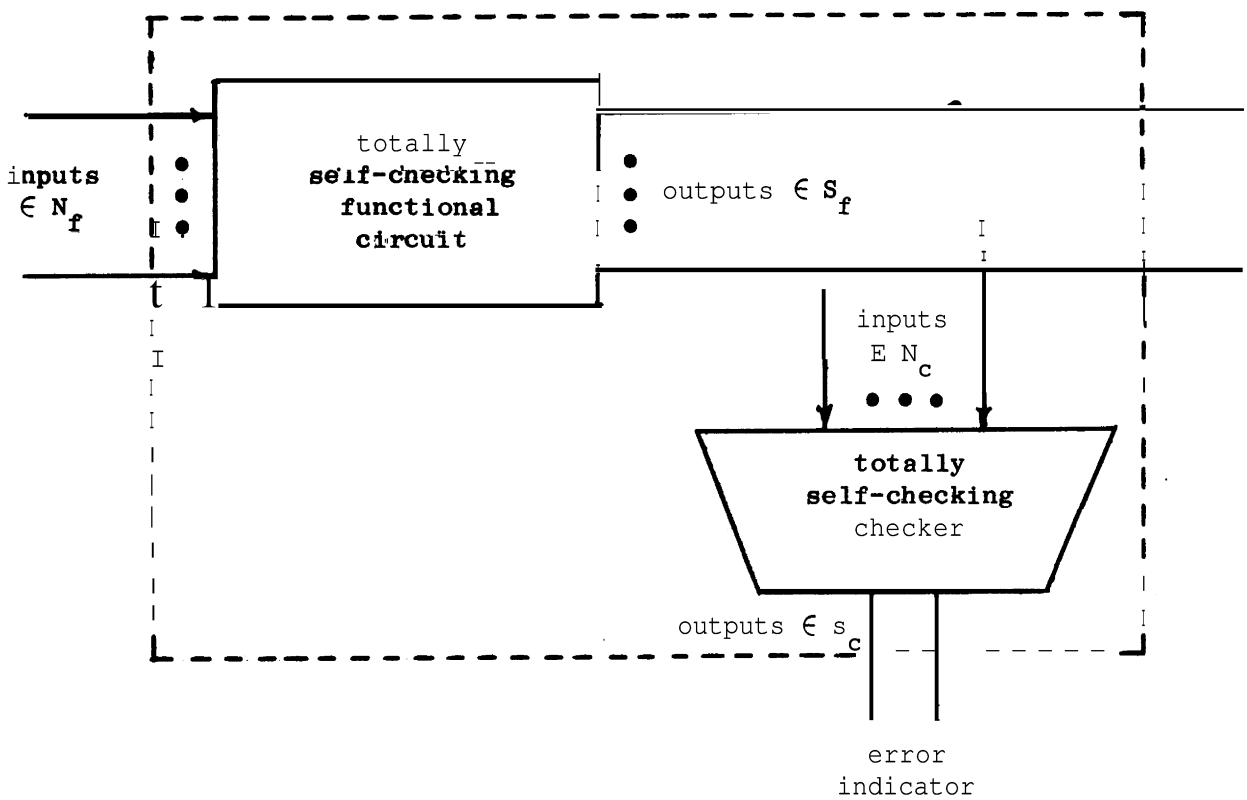


Fig. 3.2 Totally self-checking network

has a normal input set $N_c = S_f$ and an output code space $S_c = \{\langle 01\rangle, \langle 10\rangle\}$.* The fault-free output function of the checker is a code disjoint mapping, that is, it always maps non-code inputs into non-code outputs. With these constraints it is easy to show that the network itself is totally self-checking (for example, see Thm. 3.2 of [2]). The normal input set of the network is N_f, while its output code space is S_c. The secure and tested fault sets of the network are the unions of the corresponding fault sets of the functional circuit and the checker.

A simple example of a totally self-checking network employs the totally self-checking n-bit bus driver of Fig. 3.1 and an n-1-bit odd parity generator. The odd parity over n-1 bits together with a wire connected to the remaining bit comprise the required two-output totally self-checking parity checker.

*The checker must have two lines encoded in this manner, for a fault sticking a single error indicator line at the "good" value would never be detected.

4. PARTIALLY SELF-CHECKING NETWORKS

The use of and motivation for.partially self-checking circuits is best given by an example. Suppose we have a machine with buses A, B, and T that carry data encoded in a single error detecting code S. Fig. 4.1 shows one bit slice of a bus switch which can transfer either A or B to T. This circuit is replicated once for each bit to be switched. The lines $\langle s_1 s_0 \rangle$ are set to $\langle 01 \rangle$ to transfer A to T and to $\langle 10 \rangle$ to transfer B. A checker may then monitor the T bus with the appearance of a non-code word signaling an error. The reader can easily verify that the circuit is fault-secure for all stuck-at faults which affect only a single bit slice, and self-testing for all stuck-at faults which affect fewer than all the bit slices. Thus the circuit is totally self-checking when used as a bus switch in this manner.

Looking at the circuit of Fig. 4.1 we notice that it may also be used to compute the logical OR of A and B by setting $\langle s_1 s_0 \rangle$ to $\langle 11 \rangle$. Unfortunately, the result in general will not be valid because the encoding of the logical OR of two operands does not in general equal the

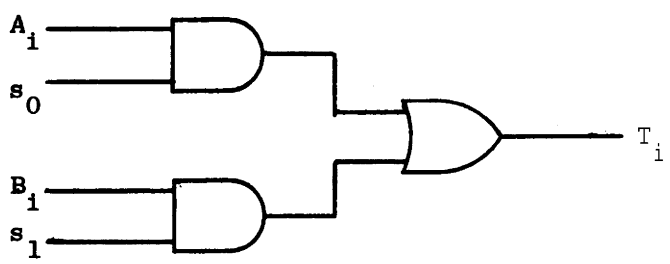


Fig. 4.1 Totally self-checking bus switch

logical OR of their encodings unless the encoding is at least complete duplication [4]. Suppose however that the encoding is a separate code, that is, a code with a separate data part and check symbol. Then the OR of the data parts will be correct; only the check symbol output will be wrong. We can then calculate a new check symbol based on the data output of the circuit and utilize the re-encoded output. This is a practical scheme only if it can be implemented in a self-checking manner at low cost. In the remainder of this section we show models of partially self-checking networks which fulfill that requirement.

4.1 Type 1 Networks

The simplest partially self-checking network is the type 1 model, shown in Fig. 4.2. It consists of a totally self-checking functional circuit with a fault-free output function which is a mapping from a normal input set N_f onto an output code space S_f; a totally self-checking checker with normal input set $N_c = S_f$ and output code space $S_c = \{<01>,<10>\}$; and two control gates and the control leads cl and c_0. The vector $<c_1 c_0>$ may be set to <01> to enable the output of the checker, or to <10> to force the error indicator output to <10> ("good").

The output code space of the network is just S_c. However, the normal input set of the network consists of vectors of the form $<c_1 c_0 i>$ where cl and c_0 are the control gate inputs and i is the functional circuit input. When functional circuit inputs from N_f are expected,

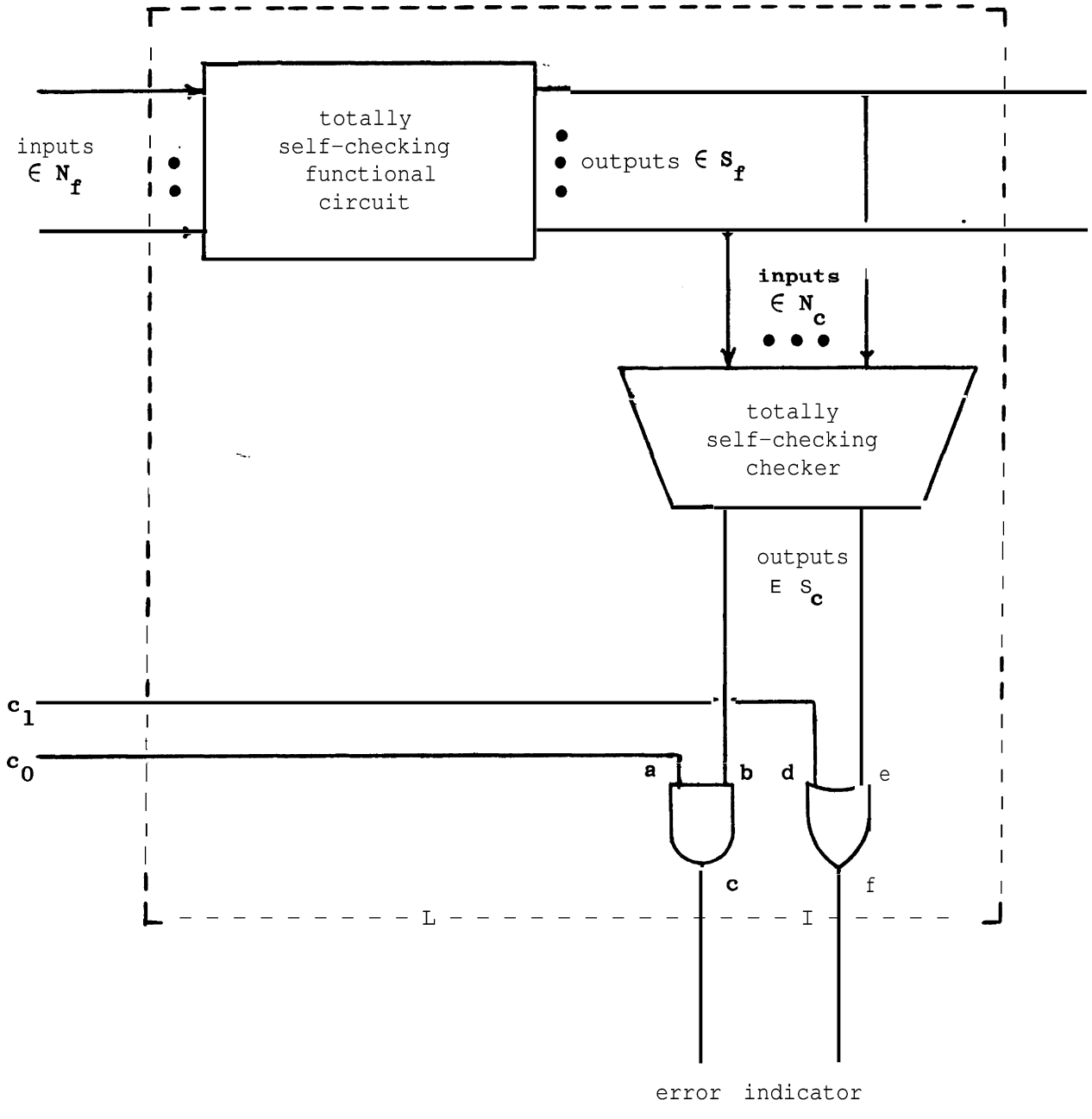


Fig. 4.2 Type 1 partially self-checking network

$\langle c_1 c_0 \rangle$ is set to $\langle 01 \rangle$ and the network is logically equivalent to the totally self-checking network of Fig. 3.2. However, when inputs not in N_f are expected $\langle c_1 c_0 \rangle$ may be set to $\langle 10 \rangle$ to disable the checker.

It is straightforward to show that the network of Fig. 4.2 is partially self-checking when used in the manner described above. Let F_a be the set of all single stuck-at faults on the control gates. That is,

$$F_a = \{a/0, a/1, b/0, b/1, c/0, c/1, d/0, d/1, e/0, e/1, f/0, f/1\}$$

Then the secure and tested fault sets of the network contain F_a as well as the corresponding fault sets of the functional circuit and checker. The secure input set of the network is I_n, where

$$I_n = \{\langle c_1 c_0 i \rangle \mid (\langle c_1 c_0 \rangle = \langle 01 \rangle) \wedge (i \in N_f)\}.$$

In insecure mode, the network has inputs from the set I'_n, where

$$I'_n = \{\langle c_1 c_0 i \rangle \mid \langle c_1 c_0 \rangle = \langle 10 \rangle\}.$$

Thus the normal input set of the network is $N_n = I_n \cup I'_n$.

Theorem 4.1: A type 1 network, described above and illustrated in Fig. 4.2, is partially self-checking.

Proof: In secure mode, that is, with inputs from I_n, the network is clearly self-testing and fault-secure for faults from the appropriate fault sets of the functional circuit and checker. It follows that the network is also self-testing with inputs from N_n since $N_n \supset I_n$. Thus we need only show self-testing and fault-secureness for faults from F_a.

(a) (self-testing) All faults except $\langle a/1 \rangle$ and $\langle d/0 \rangle$ are tested by some input from I_n, since a and d have the values 1 and 0 respectively during such operation, and both 0's and 1's must be transmitted through the paths $\langle bc \rangle$ and $\langle ef \rangle$. This is true because each checker output takes on both values 0 and 1. The faults $\langle a/1 \rangle$ and $\langle d/0 \rangle$ are each detected by some input from I'_n, since one of these faults changes the correct error indicator output of $\langle 10 \rangle$ to a non-code word. Thus all faults in F_a are tested by some input in $N_n = I_n \cup I'_n$.

(b) (fault-secureness) It is clear that a single fault from F_a causes at most a distance one change in the error indicator output, producing either the correct output or a non-code word. ∎

An example of a type 1 partially self-checking network is the n-bit parity checked bus driver shown in Fig. 4.3. The totally self-checking functional circuit here is the n-bit bus driver of Fig. 3.1, while the totally self-checking checker consists of an n-1-bit even-parity generator and an inverter connected to the remaining data bit. The control vector $\langle c_1 c_0 \rangle$ is set to $\langle 01 \rangle$ when even-parity operands are to be transmitted, and to $\langle 10 \rangle$ for vectors of unknown parity.

The usefulness of type 1 networks is limited since in insecure mode they do not re-encode the functional circuit output. We notice in the example of Fig. 4.3 that the correct parity output is always available from the parity generator at line p, and could be utilized at essentially zero cost. Type 2 networks are a formalization of this idea.

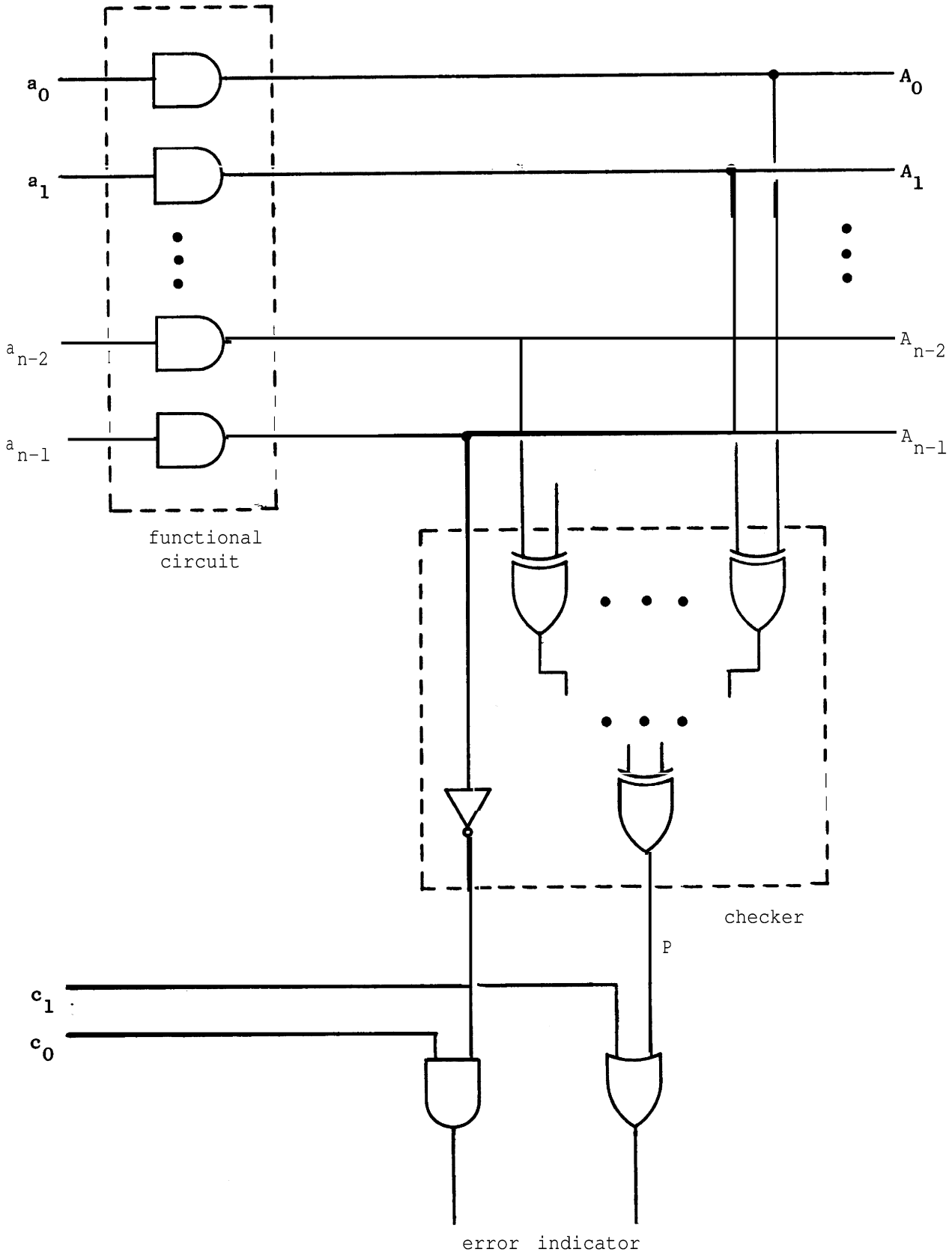


Fig. 4.3 Partially self-checking parity-checked bus driver

4.2 Type 2 Networks

If the output code space of a self-checking circuit is a separate code, a checker can consist of an equality checker which compares the check symbol output of the circuit with a new check symbol generated on the basis of the data output of the circuit, as suggested by Fig. 4.4. The following lemma shows that such a checker is totally self-checking if the equality checker is.

Lemma: Let the code words <cd> in a separate error-detecting code S consist of a data part d and a check symbol c such that c = C(d). Then a network consisting of a check generator G which computes C(d) and a totally self-checking equality checker which compares the output of G and check symbols **c** is a totally self-checking checker for code words **<cd>** in S.

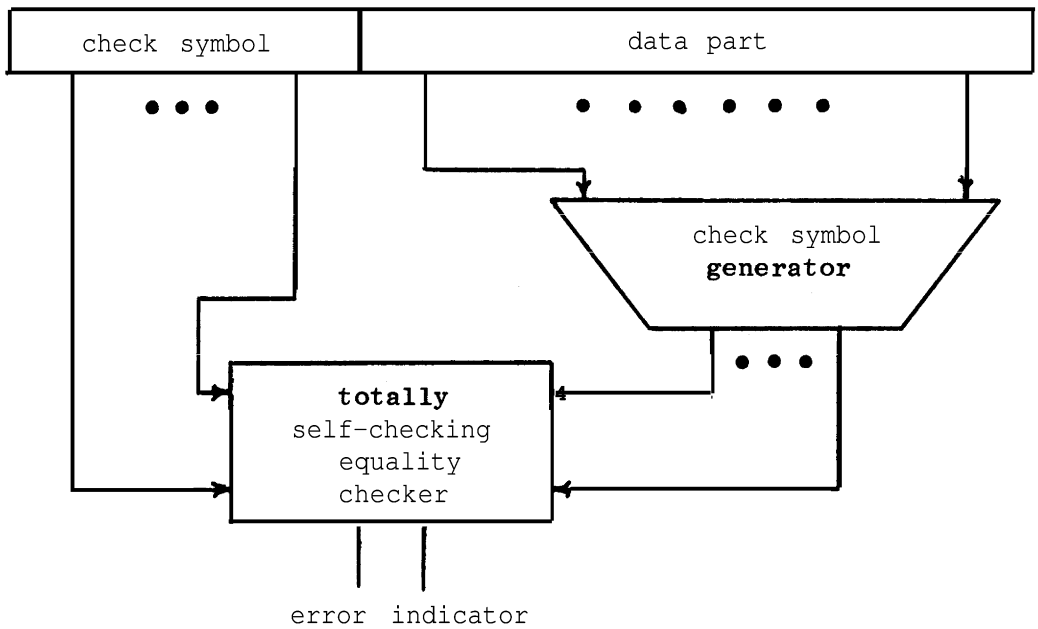


Fig. 4.4 Totally self-checking checker for separate codes

Proof: The normal input set of the network is S, while the output code space of the network is the output code space of the equality checker. Let F_g be the set of all check generator faults which produce an incorrect generator output for at least one network input in S. Clearly faults outside of F_g have no effect on the network. The reader can easily verify that the network is self-testing and fault-secure for faults in $F_{g'}$ as well as for faults in the tested and secure fault sets of the equality checker. The tested and secure fault sets of the network are the appropriate unions of the above sets. •

The proof of the above lemma depends primarily on the existence of a totally self-checking equality checker for k-bit check symbols c. If the k-bit vectors do not take on all 2^k possible values then a checker might not exist. However, if the k-bit vectors do take on all values then we are assured of the existence of a totally self-checking equality checker regardless of the value of k [2].

A type 2 network, shown in Fig. 4.5, is a type 1 partially self-checking network which uses the totally self-checking checker for separate codes described above, and which has a re-encoded functional circuit output-derived from the check generator. The input sets, fault sets, and output code space of a type 2 network are the same as those of the corresponding type 1 network. Thus ignoring the re-encoded functional circuit output, a type 2 network is merely a type 1 network with more detail specified, and hence is partially self-checking. However, it does have a re-encoded functional circuit output available, and the

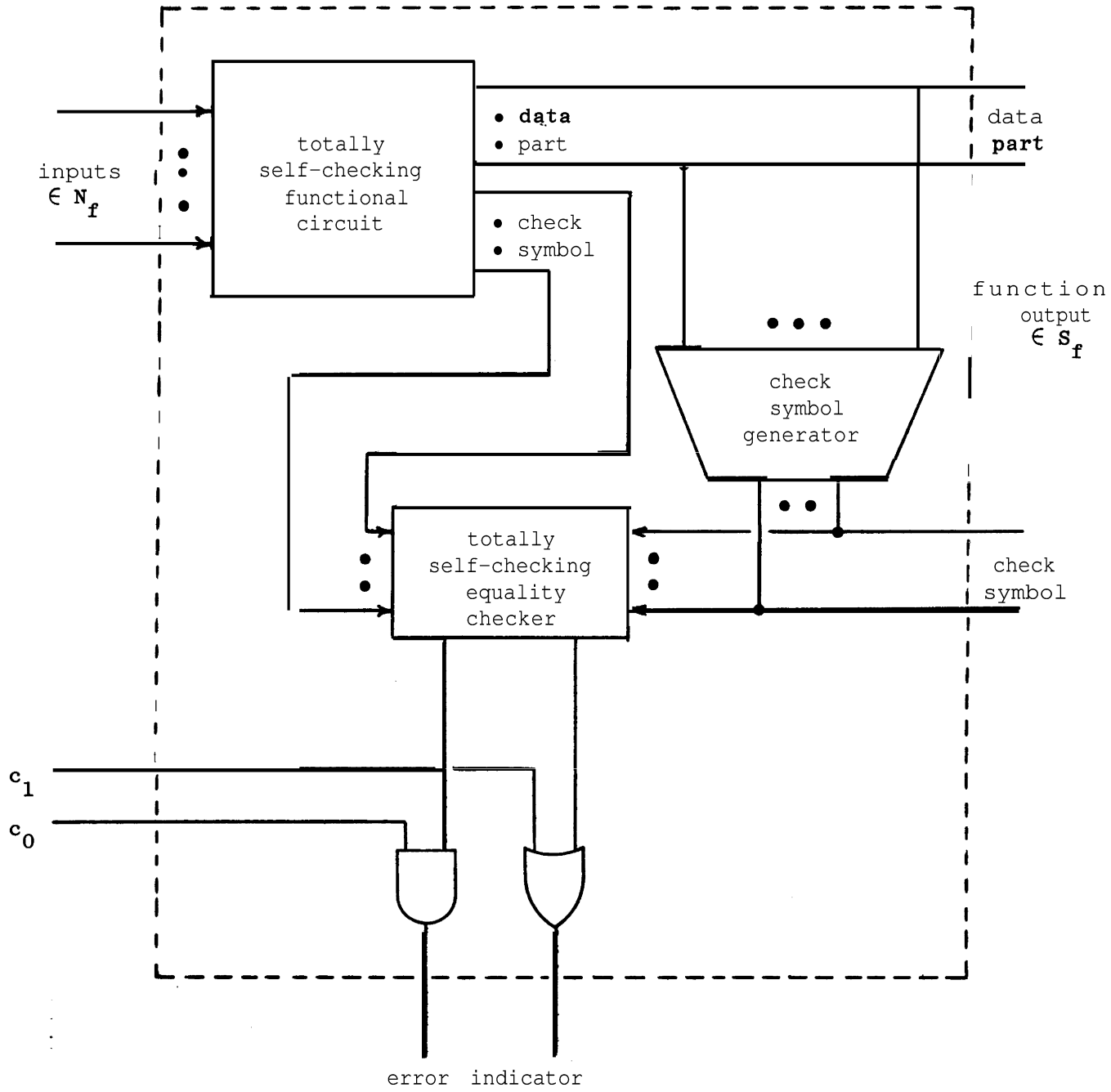


Fig. 4.5 Type 2 partially self-checking network

appearance of a non-code word here is reflected by the checker output, since the checker function is a code disjoint mapping. These results are summarized in the following theorem...

Theorem 4.2: A type 2 network, described above and illustrated in Fig. 4.5, is partially self-checking. Furthermore, in the absence of faults, the re-encoded functional circuit output is always a code word; the appearance of a non-code word because of a fault is reflected by a non-code output of the checker.

4.3 Type 3 Networks

A noticeable disadvantage of type 2 networks is that the functional circuit output is delayed by the re-encoding process using the check generator. In a totally self-checking or type 1 partially self-checking network the total delay is that of the functional circuit alone, while in a type 2 network it is the sum of the functional circuit and check generator delays. In insecure mode the re-encoding process will always introduce some delay, but a type 3 network reduces the delay in secure mode to two gate delays.

A type 3 network, illustrated in Fig. 4.6, consists of a totally self-checking functional circuit and equality checker, a check generator, and control gates to switch either the functional circuit check symbol output or check generator output to the network output. The equality checker compares the network check symbol output with the generated

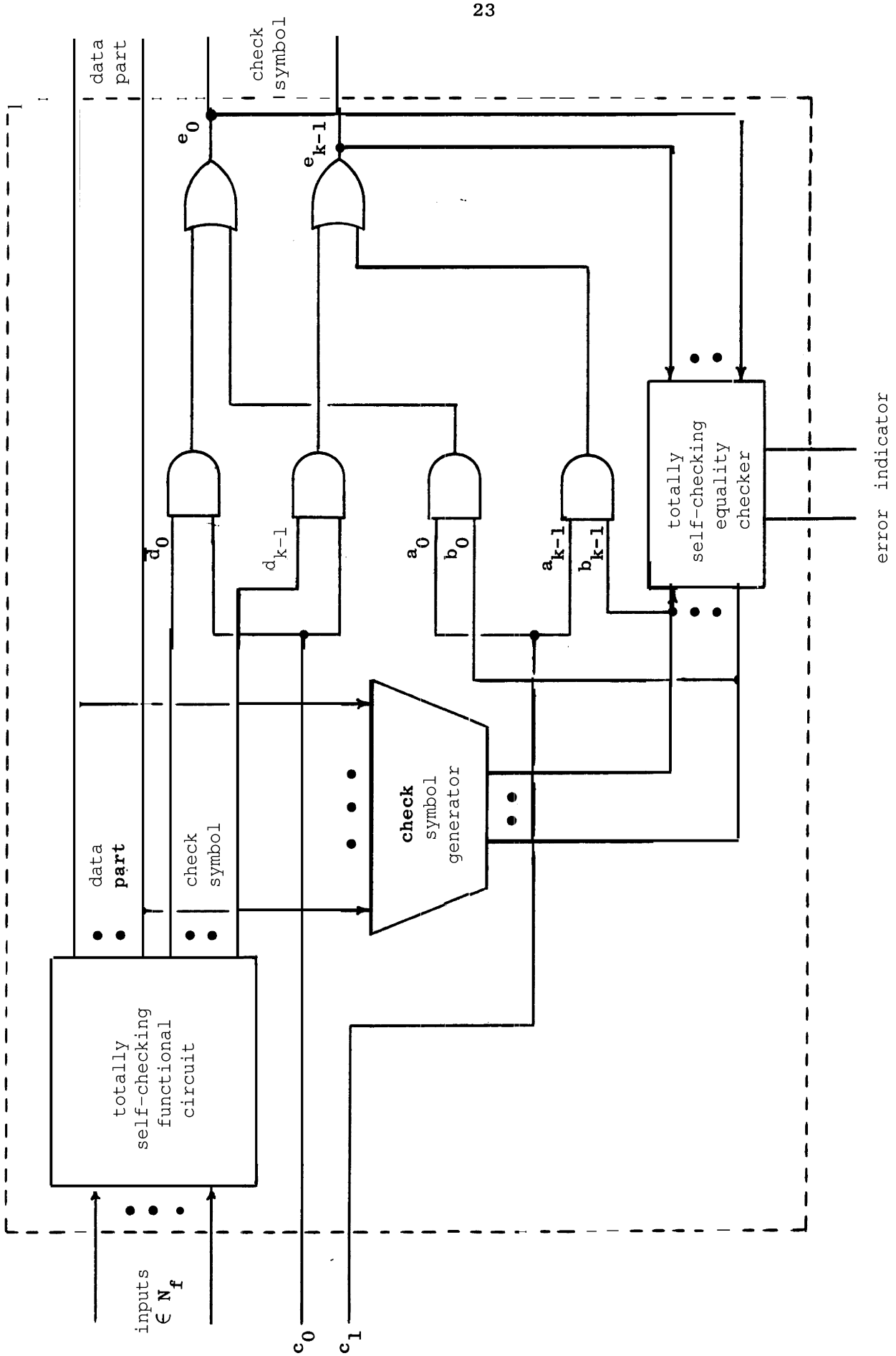


Fig. 4.6 Type 3 partially self-checking network

check symbol. When $\langle c_1 c_0 \rangle$ equals $\langle 01 \rangle$ (secure mode), the network is logically equivalent to a totally self-checking network; when $\langle c_1 c_0 \rangle$ equals $\langle 10 \rangle$ (insecure mode), the functional circuit is re-encoded and the equality checker compares the generated check symbol with itself, producing a "good" output.

The normal input set, secure input set, and output code space of a type 3 network are similar to those of type 1 and 2 partially self-checking networks. If F_a is the set of all single faults on the control gates, except the faults $\langle a_i/1 \rangle$, then the secure and tested fault sets of the network are the union of F_a and the appropriate fault sets of the functional circuit and checker.*

-Theorem 4.3: A type 3 network, described above and illustrated in Fig. 4.6, is partially self-checking.

Proof: The problem is similar to Thm 4.1, and reduces to showing that the network is self-testing and fault-secure for faults in F_a. As in Thm 4.1, self-testing is proved by showing that there is a test for every fault in F_a in either secure or insecure mode. Fault-secureness follows from the observation that a fault in F_a either has no effect on the check symbol output, or changes the check symbol output causing an error indication by the equality checker. ∎

*Here the "checker" is the combination of the check generator and totally self-checking equality checker, as in type 2 networks.

Although type 3 networks avoid the delay of re-encoding the functional circuit output in secure mode, they have some disadvantages. First, they require more control gates than a type 2 network, with a corresponding increase in cost. Second, they have a set of single stuck-at faults for which the network is not generally self-testing or fault-secure, namely the faults $\langle a_i/1 \rangle$. If the network is not self-testing for faults $\langle a_i/1 \rangle$, then these faults must be tested periodically by some manual, software, or firmware method.

In a specific implementation of a type 3 network, self-testing and fault-secureness for $\langle a_i/1 \rangle$ will depend on timing in the network and in the circuits following it. For example, suppose the type 3 network performs an operation which sets lines b_j, d_j, and e_j to 1. Suppose that the next operation sets line d_j to 0. Depending on the timing and control sequence used, line b_j may become 0 some time after line d_j does. Thus line e_j is erroneously held at logic value 1 until the check generator "catches up." To the circuit receiving the output of the type 3 network, the effect is similar to that of intermittent stuck-at-1 fault on line e_j. On the other hand, if the output of the check generator always has the value 0 between operations, then the problem outlined above does not occur.

A simple example of a partially self-checking network uses the bus switch circuit of Fig. 4.1 in a type 2 or type 3 configuration modeled after Fig. 4.2 or Fig. 4.6. This network could be used in a CPU as a bus switch and also to perform the logical OR operation. In a machine in which data was encoded in an arithmetic code, the other logical

operations could be performed using a combination of the OR operation and totally self-checking arithmetic operations [6]. However, we will later show a totally self-checking functional circuit which can be used in a partially self-checking network to perform all logical operations. But first we must indicate how to verify the self-checking properties of non-trivial circuits.

5. VERIFICATION OF SELF-CHECKING PROPERTIES

In this section we will show how to verify the self-checking properties of a class of circuits defined below.

Definition: A bit-sliced circuit is a multiple-output combinational circuit in which each output bit is computed by an independent subcircuit, called a bit slice.

The bus switch discussed earlier is a bit-sliced circuit, with a bit slice shown in Fig. 4.1.

To show that a circuit is self-checking, we must show that it is self-testing for a fault set F_t and fault-secure for a set F_s.

5.1 Fault-secureness

Fault-secureness of bit-sliced circuits is particularly easy to show, as is evidenced by the following theorem.

Theorem 5.1: Let S be an error-detecting code of distance two or more. Let a bit-sliced circuit have a fault-free output function $Z(i,\lambda)$ which is a mapping from an input set I into S. Let F_s be the set of all faults that affect only a single bit slice. Then the circuit is fault-secure for inputs in I and faults in F_s.

Proof: Any fault f in F_s affects only a single bit slice, and therefore only a single output bit. For a particular input vector $\underline{i}$ if

the fault does not change this output bit then $Z(i,f) = Z(i,\lambda) \in S$; if it does change it then the output is distance one away from a code word in S and Z(i,f) is not in S because S is a distance-two code. ∎

In practice, the normal input set N of a totally self-checking functional circuit may be chosen as the largest set for which the output function is a mapping from N onto a distance-two code S; due to Thm. 5.1 the circuit will be fault-secure for these inputs. If there are inputs outside of N which will be used in normal operation, but which produce outputs outside of S, these are the inputs for which the checker is disabled in a partially self-checking network.

5.2 Self-testing

While fault-secureness is easy to show, self-testing for all single stuck-at faults is not a general property of bit-sliced circuits and depends on the design of the circuit and the exact composition of N. However, we shall see in the following development that we can determine self-testing for an entire circuit by considering only individual bit slices.

Definition: The <u>set of active input combinations</u> to a bit slice B_i in a bit-sliced circuit is the set $C_i = \{c \mid c$ is the input of B_i for some circuit input in $N\}$.

Definition: Let a bit slice B_i realize the single output function $Z_i(c,f)$. Then the set of testable faults of the bit slice is the set

$$F_i = (f \mid (f \text{ affects only } B_i) \wedge (\exists c \in C_i \text{ s.t. } Z_i(c,f) = \overline{Z_i(c,\lambda)})\}.$$

Theorem 5.2: A bit-sliced circuit with distance-two output code S is self-testing for the fault set $F_t = \bigcup_i F_i$.

Proof: For any fault f in any F_i, there is an input c in C_i and a corresponding circuit input i' in N such that $Z_i(c,f) = \overline{Z_i(c,\lambda)}$. Furthermore, no other ouput bit is affected by f. Thus the circuit output $Z(i',f)$ is distance one from $Z(i',\lambda) \in S$ and therefore not in S. So the circuit is self-testing for any fault in any F_i, and hence it is self-testing for any fault in $F_t = \bigcup_i F_i$. ■

Due to Thm 5.2 we may prove self-testing of a bit-sliced circuit by considering each bit slice separately. The problem is further reduced in many cases because the bit slices B_i are identical, as are the sets of active input combinations C_i. The problem is then that of showing that the set F_B of testable faults for the standard bit slice contains all reasonable faults. The standard set C_B of active input combinations is determined by inspection of N. In the remainder of this section we suggest how to determine F_B for a bit slice, given a structural specification of the circuit and C_B.

The problem of determining F_B can be attacked using any method of

finding which faults in a circuit are detected by a particular test. Such a method would be used to find the set of faults detected by each active input combination to a bit slice, and the union of these sets would be the tested fault set for the bit slice. Examples of existing methods of finding faults detected by a test are Roth's "test-detect" [7] and Armstrong's deductive method [8]. Another method, described in [9], employs Reese's gate equivalent model (GEM) [10]. In this method, the GEM of a bit slice is derived, and tested faults are determined by assigning input literals the values they receive in active input combinations. This method was used to generate Table 6.2 in the next section.

An alternative approach to verifying the self-testing property is to fix F_B as some known fault set and then prove that all faults in that set are tested by some active input combination. This could be done using conventional test generation techniques, generating tests for each fault until a test which is also an active input combination is found; such a procedure would be rather inefficient. However, Wakerly and McCluskey [11] give a Karnaugh map method which can be used to verify that any particular test set detects all single stuck-at faults in a general single-output network. The method requires deriving the GEM of the network, mapping the PI-sets or SI-sets [10], marking the active input combinations, and visually checking for "growth" and "existence" tests. The method can also be used to determine which faults are detected by a particular test and was used by the author to verify the correctness of Table 6.2 in the next section.

6. A PARTIALLY SELF-CHECKING NETWORK FOR ARITHMETIC AND LOGICAL OPERATIONS

The circuit of Fig. 6.1 can be used to perform all 16 Boolean functions of two input variables A_i and B_i by appropriately setting the control input vector $\langle S_3 S_2 S_1 S_0 \rangle$. The circuit may be replicated to form a bit-sliced functional circuit to perform any of these operations on two input vectors A and B. For each value of $j = \langle S_3 S_2 S_1 S_0 \rangle$, Table 6.1 gives $f_j(A,B)$.

If input vectors A and B are encoded in a distance-two error detecting code S, and if a function $f_j(A,B)$ preserves* this encoding, then according to Thm 5.1 the functional circuit is fault-secure. The secure fault set of the circuit contains all faults which affect only a single bit slice, and the secure input set is

$$I_j = \{\langle S_3 S_2 S_1 S_0 AB \rangle \mid (\langle S_3 S_2 S_1 S_0 \rangle = j) \wedge (A,B \in S)\}.$$

If the encoding is preserved by $f_j(A,B)$ for a number of j, say $j \in J$, then the secure input set of the circuit is $I = \bigcup_{j \in J} I_j$.

Due to Thm 5.2, the functional circuit is also self-testing for certain faults when the function $f_j(A,B)$ preserves the encoding of A and B. Assuming that input bits A_i and B_i take on all four possible combinations and that the function $f_j(A,B)$ is code-preserving, Table 6.2 shows which single stuck-at faults in a bit slice are tested by selected functions. (The table includes only one member from each class of structurally equivalent faults.)

*A function $f_j(A,B)$ preserves S if $A,B \in S$ implies $f_j(A,B) \in S$.

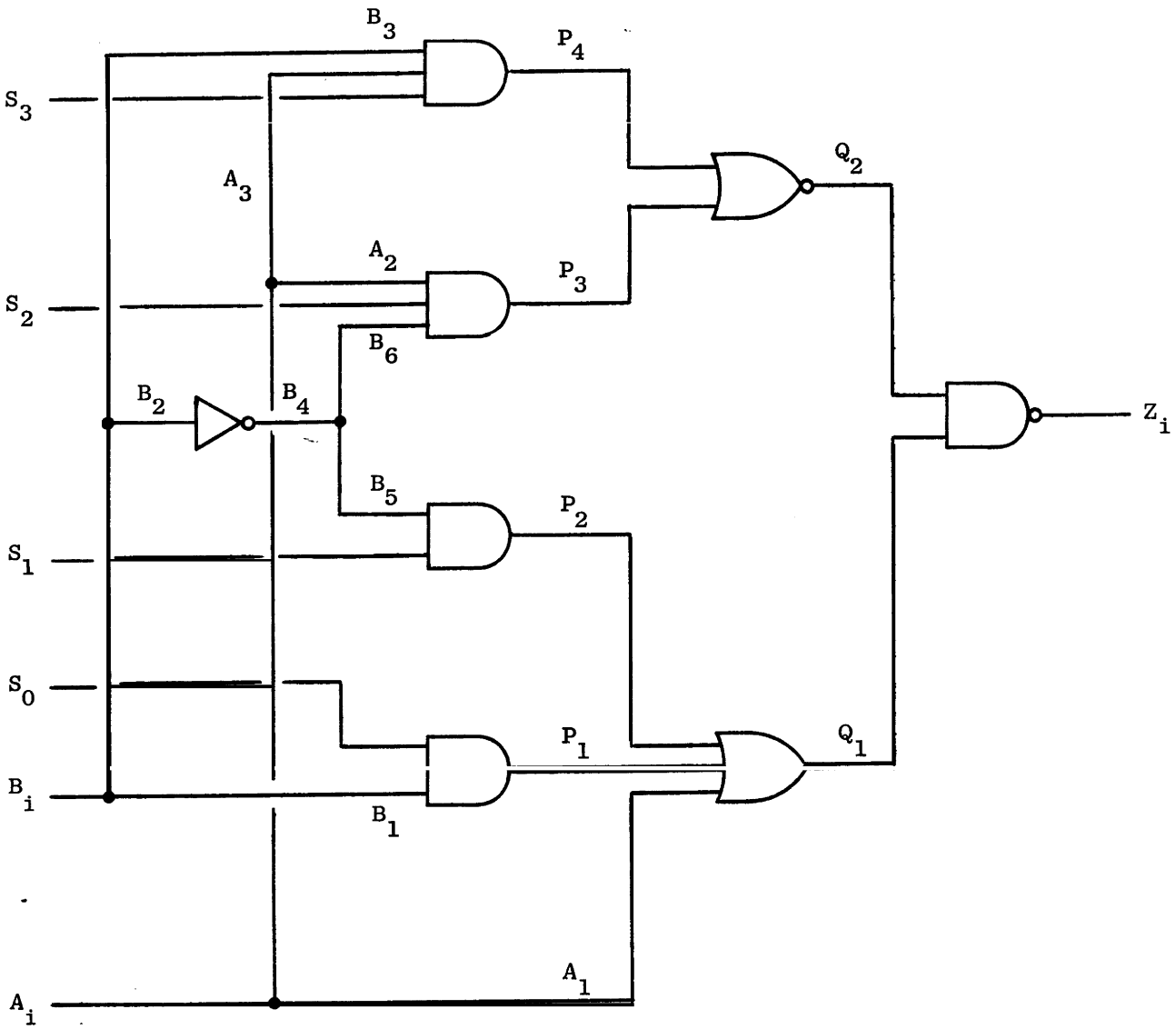


Fig. 6.1 Bit slice to perform any logic function of two variables

TABLE 6.1

$j = S_3S_2S_1S_0$	$f_j(A,B)$	$j = S_3S_2S_1S_0$	$f_j(A,B)$
0 0 0 0	$\overline{A}$	1 0 0 0	$\overline{A}+B$
0 0 0 1	$\overline{A+B}$	1 0 0 1	$\overline{A \oplus B}$
0 0 1 0	$\overline{A} \cdot B$	1 0 1 0	B
0 0 1 1	0	1 0 1 1	$A \cdot B$
0 1 0 0	$\overline{A \cdot B}$	1 1 0 0	1
0 1 0 1	$\overline{B}$	1 1 0 1	$A+\overline{B}$
0 1 1 0	$A \oplus B$	1 1 1 0	A+B
0 1 1 1	$A \cdot \overline{B}$	1 1 1 1	A

Table 6.1: Functions performed by the circuit of Fig. 6.1

For example, suppose A and B are vectors from an error-detecting code S consisting of all even parity n-bit vectors where n is even. The code S is preserved by the operations $A \oplus B$, $\overline{A \oplus B}$, A, B, $\overline{A}$, $\overline{B}$, 0, and 1. Inspection of Table 6.2 reveals that all single stuck-at faults in a bit slice are detected by $A \oplus B$ and $\overline{A \oplus B}$ or by A, B, $\overline{A}$, and $\overline{B}$. If the normal input set of the functional circuit contains any such set of code-preserving operations which tests all faults in each bit slice, then the circuit is self-testing. The circuit is also fault-secure for code-preserving operations and hence it is totally self-checking.

TABLE 6.2

stuck-at-0

$f_j(A,B)$	A	Al	B	B_4	S_0	S_1	S_2	S_3	P_1	P_2	P_3	P_4	Z
0					X	X			X	X			
1							x	x			X	X	X
$A \oplus B$	X	X	X	X		X	X			X	X		X
$\overline{A \oplus B}$	X	X	X		X			X	X			X	X
A	X				X	X	x	x	X	X	X	X	X
B			X	X		X		X		X		X	X
$\overline{A}$	X	X											X
$\overline{B}$			X	X	X		X		X		X		X

stuck-at-1

f_j (A,B)	A	A_2	A_3	B	B_1	B_3	B_4	B_5	B_6	S_0	S_1	S_2	S_3	Q_1	Q_2	Z
0												X	X			X
1										X	X			X	X	
$A \oplus B$	x	x		X			X	X	X	X			X	X	X	X
$\overline{A \oplus B}$	X		X	x	x	X					X	X		X	X	X
A	x	x	X													X
B				X		x	x	X		X		X		X	X	X
$\overline{A}$	X									X	X	X	X	X		X
$\overline{B}$				x	x		X		X		X		X	X	X	X

Table 6.2: Fault tests for the circuit of Fig. 6.1

Since the bit-sliced functional circuit of Fig. 6.1 is totally self-checking when used in the manner described above, it can be employed in a partially self-checking network which re-encodes the output for those functions which are not code-preserving. For example, we can use the circuit in a partially self-checking two-input universal logic unit in a machine whose data is parity-encoded as described above. The function selection vector $\langle S_3 S_2 S_1 S_0 \rangle$ and the checker enable control $\langle c_1 c_0 \rangle$ could be supplied by a microprogrammed control unit. (Checking the control is discussed in [12].) The logic unit would operate in secure mode for the code-preserving operations and in insecure mode for the non-code-preserving operations such as AND and OR.

Four copies of the bit slice of Fig. 6.1 are used along with some carry logic in an existing MSI chip, the 74181 4-bit arithmetic and logic unit [13,14]. In this chip, the logic functions of Table 6.1 are performed when a control lead M is set to 1 to disable internal carries. When M is set to 0, internal carries are enabled and the unit performs arithmetic operations.

Because of its carry logic the 74181 is not a bit-sliced circuit. However, if the input operands A and B are encoded in a distance-two arithmetic error-detecting code, then the output is a code word for the addition and subtraction operations. A single stuck-at fault causes an error with arithmetic weight at most one, producing a non-code word. Thus it is possible to show that for code-preserving operations the circuit is fault-secure for all single faults.*

*Except faults on control leads S_3, S_2, S_1, S_0, and M which occur before these leads fan out to the individual bit slices.

When used to perform addition and subtraction on data in an arithmetic code the 74181 is self-testing for faults which affect the carry logic. Faults in the logic unit bit slices (Fig. 6.1) are also tested. With carries disabled (M=1), logic unit operations are performed and faults are tested by code-preserving operations according to Table 6.2. Code-preserving operations are A and B for any arithmetic code, and also $\overline{A}$, $\overline{B}$, 0, and 1 for the low-cost codes [15]. With carries enabled (M=0), arithmetic operations are performed. During addition and subtraction the $\overline{A \oplus B}$ and $A \oplus B$ functions of the logic unit are used, and the corresponding faults indicated in Table 6.2 are tested. Thus the 74181 is self-testing for all single faults. provided that the following occur in normal operation: (a) either addition or subtraction to test the carry logic; (b) any combination of addition, subtraction, and code-preserving logic unit operations which tests all logic unit faults; and (c) at least one arithmetic and one logic unit operation to test the carry-enabling circuitry.

Under the conditions outlined above, the 74181 4-bit ALU is self-testing and fault-secure when used to perform code-preserving operations on data in an arithmetic code; hence it is totally self-checking. Fig. 6.2 shows an implementation using 74181's in a totally self-checking arithmetic and logic unit for 16-bit operands with 4-bit check symbols in a low-cost residue code [15]. Addition here is in the 1's-complement system; addition in the 2's-complement system requires additional circuitry to correct the check symbol when a carry out of the high order data bit position occurs [15]. The

functional circuit can be employed in a partially self-checking network which performs non-code-preserving operations in insecure mode.

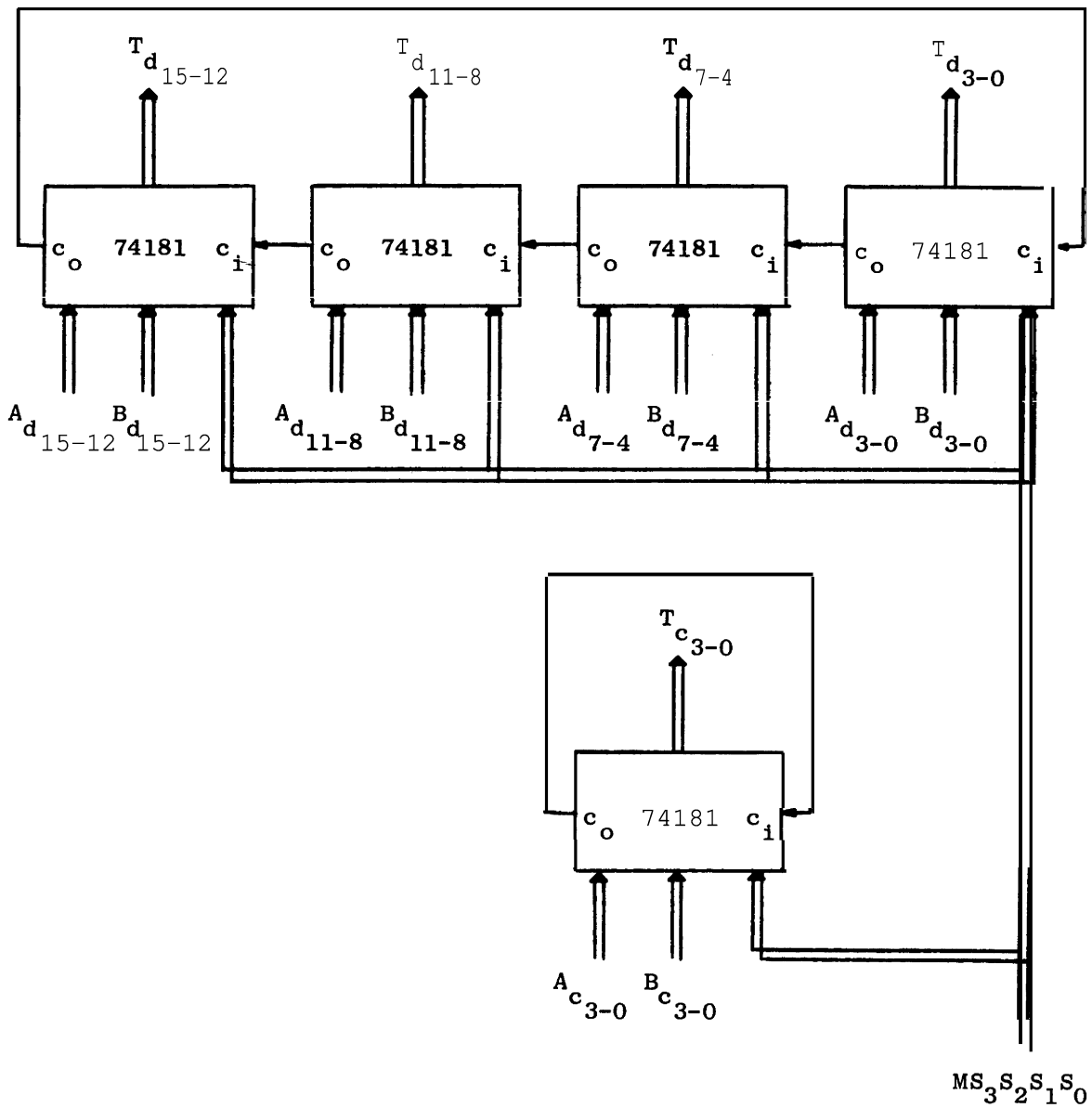


Fig. 6.2 Partially self-checking ALU using 74181 4-bit ALU chips

7. OTHER APPLICATIONS

An existing use of the partially self-checking concept is in arithmetic processors for addition, subtraction, and iterative algorithms such as multiplication and division. If data is encoded in an arithmetic code, then the adder circuit is self-testing and fault-secure for the addition and subtraction operations. However, during iterative operations the checker may be disabled until the end to increase speed, and undetected errors due to repeated use faults [15] can occur. Thus the arithmetic processor is partially self-checking, operating in secure mode for addition and subtraction and in insecure mode for the iterative algorithms.

Any totally self-checking functional circuit may be incorporated in a partially self-checking network. Such a network is useful if in addition to secure mode the functional circuit has a useful mode of operation in which the output is not a code word.

8. CONCLUSION

Several techniques are available for providing fault-detection in fault-tolerant computers. In simple systems duplication and matching might be the most inexpensive method because it requires the least control circuitry and the least design effort. However, in systems with a large number of fast registers which must be checked, or in systems which are to be made as small as possible for LSI implementation, the use of error-detecting codes is the most inexpensive means of fault-detection. Unfortunately, there is no simple code for checking logical operations such as AND and OR, and previous systems using coding have resorted to duplication for these operations. In this report we have developed a theory of partially self-checking circuits, and shown how partially self-checking networks may be used to perform logical operations. The use of partially self-checking networks is a low-cost method of performing these operations in systems employing error-detecting codes for checking arithmetic and data transfer operations.

9. REFERENCES

[1] Carter, W. C., and P. R. Schneider, "Design of dynamically checked computers," IFIP 68;' vol. 2. Edinburg, Scotland, pp. 878-883, Aug. 1968.

[2] Anderson, D. A., "Design of self-checking digital networks using coding techniques," Coordinated Sci. Lab., Univ. Illinois, Urbana, Rep. R-527, Sept. 1971.

[3] Carter, W. C., K. A. Duke, and D. C. Jessep, "A simple self-testing decoder checking circuit," IEEE Trans. Comput., vol. c-20, pp. 1413-1414, Nov. 1971.

[4] Peterson, W. W., and M. 0. Rabin, "On codes for checking logical operations," IBM Journal, vol. 3, pp. 163-168, Apr. 1959.

[5] Avizienis, A., et. al., "The STAR (self-testing and repairing) computer: An investigation of the theory and practice of fault-tolerant computer design," IEEE Trans. Comput., vol. C-20, pp. 1312-1321, Nov. 1971.

[6] Rao, T. R. N., and P. Monteiro, "A residue checker for arithmetic and logical operations," Dig. 1972 Int'l. Symp. Fault-Tolerant Computing, pp. 8-13, June 1972.

L-71 Roth, J. P. et. al., "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," IEEE Trans. Electron. Comput., vol. EC-16, pp. 567-580, Oct. 1967.

[8] Armstrong, D. B., "A deductive method for simulating faults in logic networks," IEEE Trans. Comput., vol. C-21, pp. 464-471, May 1972.

[9] Wakerly, J. F., "A method of finding faults detected by tests using the GEM," Dig. Syst. Lab., Stanford,Calif., Tech. Note 31, August 1973.

[10] Reese, R. D., and E. J. McCluskey, "A gate equivalent model for combinational logic network analysis," Dig. 1973 Int'l. Symp. Fault-Tolerant Computing, June 1973.

[11] Wakerly, J. F., and E. J. McCluskey, "A graphical method of identifying fault tests in combinational logic networks," Dig. Syst. Lab., Stanford,Calif., Tech. Rep. 66, August 1973.

[12] Wakerly, J. F. "Low-cost error detection techniques for small computers," Dig. Syst. Lab., Stanford,Calif., Tech. Rep. 51, Sept. 1973.

[13] Fairchild 9341/54181, 74181 data sheet.

[14] Signetics S54181/N74181 data sheet.

[15] Avizienis, A., "Arithmetic codes: Cost and effectiveness studies for applications in digitai' systems design," IEEE Trans. Comput., vol. C-20, pp. 1322-1331, Nov. 1971.