tions, since the parity of the path (X, N) [or of (Y, N)] cannot be determined. By running the Deduction Algorithm with the initial model, an inconsistency is immediately encountered because the value of N in t_4 is 0 when derived from the values of X and Y, and 1 when derived from the values of S and CI. At this point the suspected modules are $M1$ and $M2$. Fig. 7(b) shows the decision tree of the hierarchical effect–cause analysis. The top-level decision node represents the selection of the model. Selecting $M2$ for expansion immediately leads to an inconsistency as follows. The complete normal path which justifies S_{45} is either $(CI, CI2, T, T1, U, S)$ or $(CI, CI2, T, T2, V, S)$; this implies that $CI2$ and T are normal and have complementary values in t_4 and t_5. But this is contradicted by N having value 0 in both t_4 and t_5. Then the algorithm restores $M2$ as a module and expands $M1$. The values of N are determined from those of S and CI, and the values of $T3$ from those of CI and N [see Fig. 7(c)]. Global implications based on N_{12} and N_{13} identify $X2$, Q, $Y2$, and R as normal. After all the implications are performed, a decision is made to justify N_{14} by the path $(X, X2, Q, N)$. Since L and its predecessors are normal, the fault is $L1$ s-a-1. If our goal is only to identify the faulty module (which in this case is $M1$), then the exact location of the fault inside the faulty module is not of interest. Assuming LIMIT1 = 1, the other potential solution [indicated by dashed lines in Fig. 7(b)], which would diagnose $L2$ s-a-1, is not generated. If we are interested in diagnosis inside the faulty module, then our method identifies an *open* between L and either $L1$ or $L2$. □

V. Conclusions

In this correspondence we presented several extensions to the effect–cause analysis method of fault diagnosis in combinational circuits. We generalized the method to circuits modeled at the module-level. Compared to the use of a gate-level model, the use of higher level models significantly reduces both the memory requirements and the execution time of the Deduction Algorithm. The actual fault in the circuit under test is located to within an equivalence class of multiple pin faults. This is particularly appropriate in a testing environment where the modules can be individually verified to be fault-free.

The higher the level of modeling, the more restricted is the fault domain. This tradeoff forms the basis of the *hierarchical effect–cause analysis*, in which we repeatedly change the model of the circuit, until the fault domain implied by the model contains a fault that is compatible with the response of the circuit. Assuming a single faulty module leads to a simple strategy of replacing one suspected module by its gate-level model, while maintaining the higher level models (and the resulting computational advantages) for the rest of the circuit.

We derived global consistency properties of the logic values under the stuck fault model, and we used them in formulating a new type of effect–cause analysis that performs path-oriented decisions and implications. The generalized concept of inversion parity allows the application of the path-oriented approach to modular circuits. The use of a higher level of modeling results in a simplified path structure. The path-oriented approach leads to a global search process that can avoid incorrect local decisions made by the line-oriented algorithm.

Acknowledgment

The author is very grateful to Prof. M. A. Breuer for valuable advice and guidance.

References

[1] M. Abramovici and M. A. Breuer, "Multiple fault diagnosis in combinational circuits based on an effect–cause analysis," *IEEE Trans. Comput.*, vol. C-29, pp. 451–460, June 1980.

[2] M. Abramovici, "Fault diagnosis in digital circuits based on effect–cause analysis," Univ. of Southern California, Los Angeles, Electron. Sci. Lab. Rep. 508, June 1980.

[3] R. P. Batni and C. R. Kime, "A module-level testing approach for combinational networks," *IEEE Trans. Comput.*, vol. C-25, pp. 594–604, June 1976.

[4] A. Goundan and J. P. Hayes, "Design of totally fault locatable combinational networks," *IEEE Trans. Comput.*, vol. C-29, pp. 33–44, Jan. 1980.

[5] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J. Res. Develop.*, vol. 10, pp. 278–291, July 1966.

[6] J. Savir, "Testing for single faults in modular combinational networks," *J. Des. Automat. Fault-Tolerant Comput.*, vol. 3, pp. 69–82, Apr. 1979.

[7] S. A. Szygenda and A. A. Lekkos, "Integrated techniques for functional and gate-level digital logic simulation," in *Proc. 10th Des. Automat. Conf.*, June 1973, pp. 159–172.

Totally Self-Checking Checker for 1-out-of-n Code Using Two-Rail Codes

JAVAD KHAKBAZ

***Abstract*—A new design for a totally self-checking 1-out-of-n checker is presented. A comparison with other existing methods [1], [10] is given. It is shown that for many practical values of n the new scheme requires less hardware and/or is faster than the other methods. The entire checker can be tested by applying all of the n possible 1-out-of-n inputs.**

***Index Terms*—Checker, code disjoint, 1-out-of-n code, programmable logic array (PLA) totally self-checking (TSC), two-rail code.**

I. Introduction

An encoding technique that is useful for unidirectional error detection is the so-called m-out-of-n encoding. Such code words are n bits long and have *exactly* m 1's in them. Much work has been done on the design of totally self-checking (TSC) m-out-of-n code checkers; for example, see [1], [10], [6], and [12].

The special case of 1-out-of-n code is of particular interest because it appears very frequently in computer systems. The most notable example is the output lines of the address decoders for read-only or random access memories. Of the works cited above only [1] and [10] consider this particular case. The solutions given in these two papers consist of a two-level design. First, the 1-out-of-n code is translated into a k-out-of-$2k$ code, and then a totally self-checking checker for the k-out-of-$2k$ code is designed. In [7] it is shown how to build TSC 1-out-of-n code checkers by building a tree of TSC 1-out-of-m_i checkers ($m_i < n$). This would result in less hardware cost, but the checker would be slower. Up to now, these approaches were believed to be the most efficient ones [6]. Here we show that if we first translate the 1-out-of-n code into a two-rail code, and then use a totally self-checking checker for the two-rail output of the translator, we can get a more efficient design for most practical values of n.

In this correspondence we orient the discussion and circuits to nMOS implementation [8], as MOS seems to be the present trend of technology for LSI and VLSI circuit designs. However, the design can be generalized to other technologies with little effort. Thus, we assume NOR logic, as explained in [8]. In circuit diagrams, we represent a device (i.e., a transistor) by putting small circles at the intersection of the input and the output lines of the device. For example, Fig. 1 depicts a function Z of inputs A, B, C, D, and E, where $Z = (A + D)'$, and hence the term *NOR logic*.

The set of faults that will be considered include stuck-at, miss-

Manuscript received September 28, 1981; revised January 12, 1982. This work was supported by the U.S. Army Electronics Research and Development Command under Contract DAAK20-80-C-0266.

The author is with the Center for Reliable Computing, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, CA 94305.

0018-9340/82/0700-0677$00.75

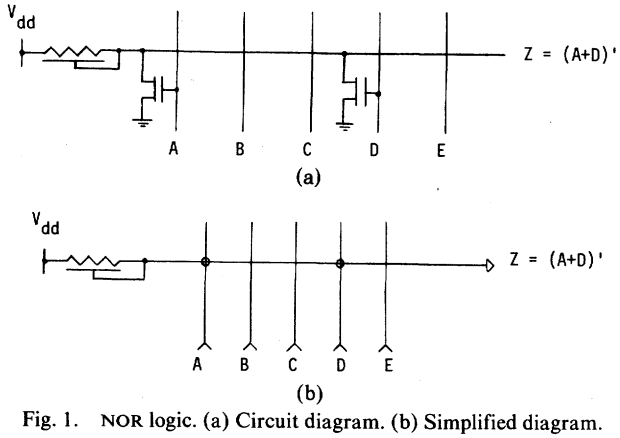

Fig. 1. NOR logic. (a) Circuit diagram. (b) Simplified diagram.

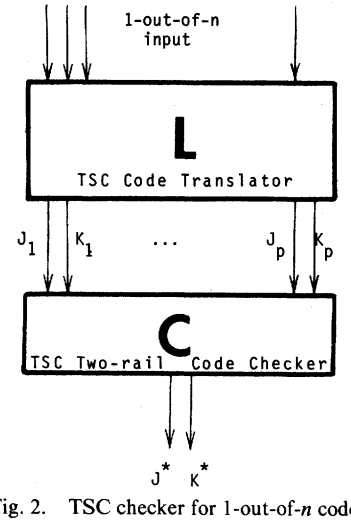

Fig. 2. TSC checker for 1-out-of-n code.

ing/extra device (i.e., spurious absence or presence of devices at cross points), and short between adjacent parallel lines [12]. We assume that a short fault results in ANDing of the logic values of the lines involved. Given such a set of faults, we say that a circuit is *fault-secure* if in the presence of a fault, any normal input to the circuit results in either the correct output or an output that indicates an error (i.e., the output is a noncodeword). A circuit is said to be *self-testing* if for any fault there is a normal input to the circuit that results in an error indication at the output (i.e., results in a noncodeword output). If a circuit is both fault-secure and self-testing, it is said to be *totally self-checking* (TSC). Finally, a circuit is said to be *code disjoint* if with no fault present in the circuit, an erroneous (noncode) input results in a noncode output and vice versa. For a more formal treatment of these definitions, see [11]. It should be noted that a code checker has to be code disjoint; that is, it has to be able to tell whether or not its inputs form a codeword. Furthermore, we require that the checker be totally self-checking, since the faults within the checker also must be taken care of.

II. A TSC Checker for 1-out-of-n Code

We design the checker in two steps (Fig. 2) as follows:

1) design a circuit L to translate 1-out-of-n into a two-rail code,

2) design a TSC two-rail checker, called circuit C.

For a 1-out-of-n input, circuit L has $2p$ pairs of output lines, where $p = \lceil \log_2 (n) \rceil$. Fig. 3 shows the code translator L for $n = 16$. Let us number the input lines of L as 1 through n, as exemplified in Fig. 3. Consider the p-bit binary representations for numbers i, $i = 1, 2, \cdots, n$. If in this representation a 0 is interpreted as the absence and a 1 as the presence of a device, then the p-bit representation of any such

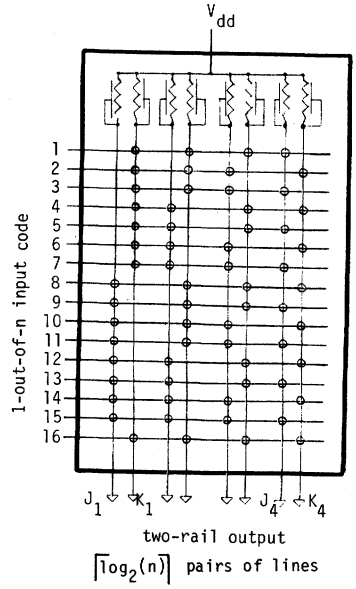

Fig. 3. Code translator for 1-out-of-16 input code.

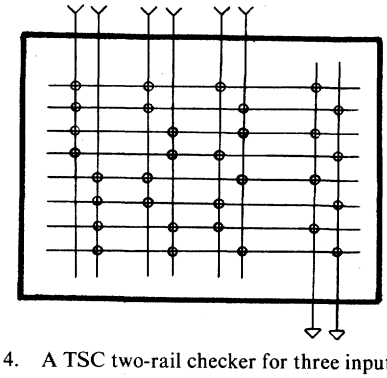

Fig. 4. A TSC two-rail checker for three input pairs.

number i corresponds to presence or absence of devices on the intersection of input line i with the J-lines, from left to right. For example, in Fig. 3, where $n = 16$ and hence $p = 4$, input line 7 has no device on its intersection with the J_1 line, but has devices on its intersections with all other J-lines. This corresponds to binary number 0111, which is the 4-bit representation of number 7. Note that since in Fig. 3 $n = 16$ (a power of two), the 4-bit representation of n is 0000, which means that there is no device on the intersections of input line 16 with the J-lines. Since the output lines of the translator are to form a two-rail code, there is a device on the intersection of input line i with K_j line if and only if there is not a device on the intersection of input line i with the J_j line. In this way, the K-lines form an encoding of the active input line's number, and the J-lines form its complement.

A design for a totally self-checking two-rail code checker is described in [2] and [11]. For two-rail inputs, this checker essentially acts as an XOR tree with two output lines. One is the parity of, say, the K-bits of all the input pairs, and the other is simply the inverse of the first one. Here we consider an implementation using programmable logic arrays (PLA's) given in [12]. For p pairs of inputs, such a PLA has $2p$ bit lines, 2^p product terms and two output lines. The output code space for this checker is the 1-out-of-2 code. An example of such a checker, for $p = 3$, is shown in Fig. 4. The arrangement of product terms and output lines in this design is specifically selected to make the PLA self-testing for any single fault of the types mentioned in the Introduction. For more detail, see [12]. Such two-rail checker circuits can be connected together in form of a tree to make a two-rail code checker for a larger number of input pairs [2].

TABLE I
INPUT TEST PATTERNS FOR CIRCUIT L

Fault in L	Test input (for L)	Output of L
input p stuck at 0	line p selected	for all i, K_iJ_i=11
input p stuck at 1	line q selected, p≠q	for some i, K_iJ_i=00
input p short to input p+1	line p selected	all K_iJ_i = 11
J_i stuck at 0	input for which J_i=1	J_iK_i = 00
J_i stuck at 1	input for which J_i=0	J_iK_i = 11
J_i short to K_i	any input	J_iK_i = 00
K_{i-1} short to J_i	line 1 or 3 selected	for some j, J_jK_j=00
missing device	line through device selected	for some i, K_iJ_i=11
extra device	line through device selected	for some i, K_iJ_i=00

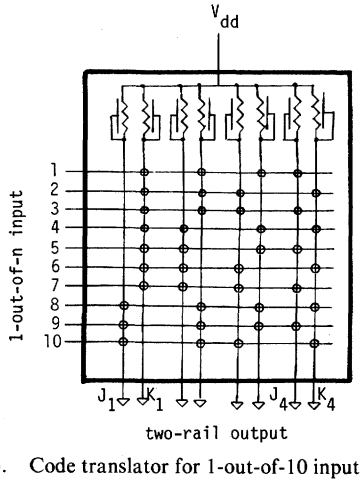

Fig. 5. Code translator for 1-out-of-10 input code.

We now briefly explain how the design of Fig. 2 constitutes a TSC 1-out-of-n code checker. It can be verified that the output of the code translator L, as typified in Fig. 3, is a two-rail word if and only if its input is 1-out-of-n. Note that if all inputs are 0, then all the outputs of L are 1. If two or more of the inputs are 1, then at least one JK output pair becomes 00. Therefore, L is code disjoint with respect to the 1-out-of-n input code space and two-rail output code space. Furthermore, the two-rail checker C of Fig. 2 is, by construction, code disjoint for two-rail inputs and 1-out-of-2 outputs. Therefore, the LC system of Fig. 2 is code disjoint with respect to the 1-out-of-n input code space and 1-out-of-2 output code space. To show that this checker is fault-secure with respect to any single fault, we can first verify that any single fault inside L either results in the correct output for L, and hence the correct output for the whole system of Fig. 2, or such a fault results in a non-two-rail output for L. In this latter case, since C is code disjoint, the output of C would not be 1-out-of-2. If the single fault occurs in C (including the connections between L and C), since C is code disjoint and fault-secure, the output of the system will either be correct or will be non-1-out-of-2. So the entire checker of Fig. 2 is fault-secure. All that is left to do is to show that the 1-out-of-n checker is also self-testing with respect to single faults. Table I lists the input patterns that are necessary for testing for various single faults in L. If n is a power of 2, as is the case in the circuit of Fig. 3, then by applying all the possible n codeword inputs to circuit L, all possible output codewords of L (input codewords of C) are generated. Since C is a TSC checker for two-rail codewords, all single faults in C will be thus detected. However, when n is not a power of 2, in general, we may not be able to test for every fault in C. For example, Fig. 5 shows the translator L for a 1-out-of-10 code. Since

TABLE II
INPUTS FOR TESTING $C1$ AND $C3$

input for L *	$K_1 \ldots K_p$ **
1	00...01
2	00...10
3	00...11
2^{p-1}	10...00
$2^{p-1}+1$	10...01

* Input i is the input with line i = 1.

** $p = \lceil \log_2(n) \rceil$.

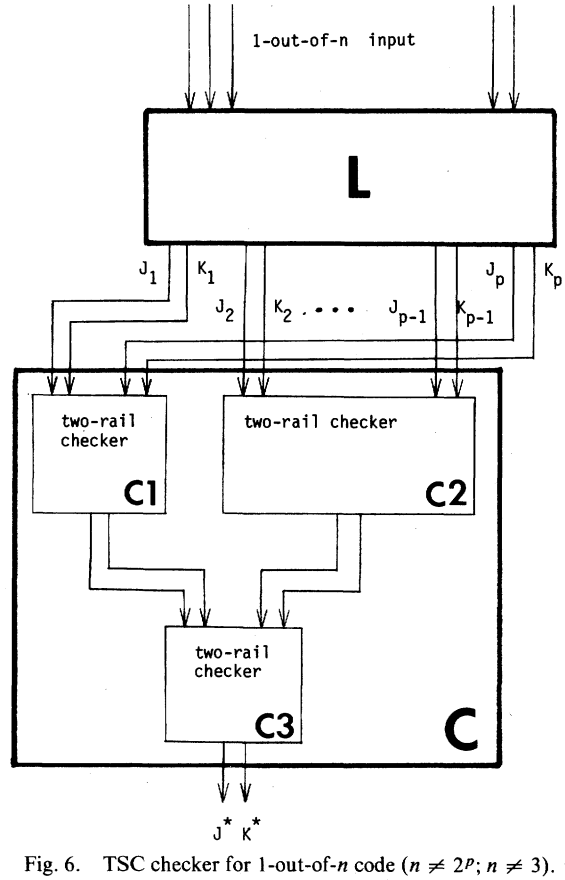

Fig. 6. TSC checker for 1-out-of-n code ($n \neq 2^p$; $n \neq 3$).

there are only 10 possible code inputs for this circuit, we can only generate 10 of the possible 16 two-rail outputs for L, which in general are not sufficient for testing C. For such a situation, we rearrange C to obtain the circuit of Fig. 6. Then it is not hard to see that by applying the n possible codeword inputs to the circuit of Fig. 6, subcircuit $C2$ receives all of its possible input codewords, and hence is tested exhaustively. Table II lists those inputs of circuit L that result in exhaustive testing of subcircuits $C1$ and $C3$.

From Table II we conclude that by applying inputs 1, 2, 2^{p-1}, and $2^{p-1}+1$ to L, circuit $C1$ of Fig. 5 will be tested exhaustively. Finally, inputs 1, 2, 3, and $2^{p-1}+1$ of L will test circuit $C3$ of Fig. 5 exhaustively. Therefore, the circuit of Fig. 2 is a TSC checker for 1-out-of-n codes if n is a power of 2. If n is not a power of 2 (and $n > 3$), then the arrangement of Fig. 6 results in such a TSC checker. In either case, the entire checker can be tested for any single fault by applying the n possible codeword inputs.

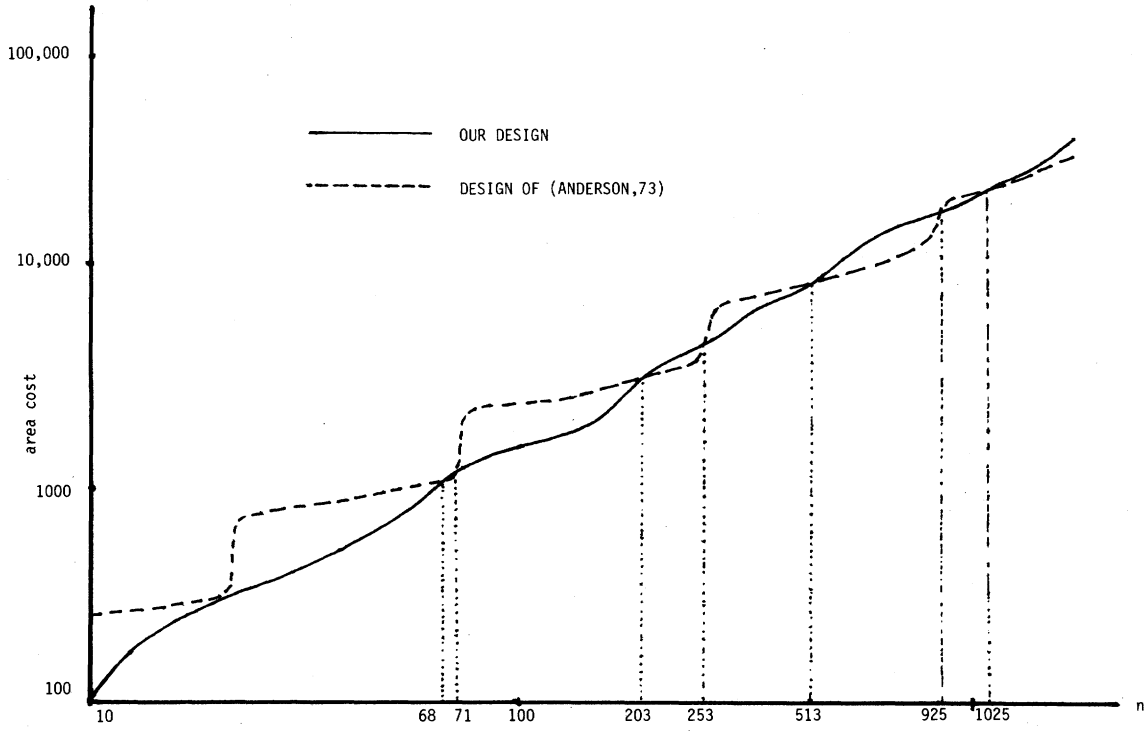

Fig. 7. Cost in area: A comparison.

III. Comparison with Other Designs

As mentioned earlier, the previous designs for TSC 1-out-of-n code checkers involved a translation from 1-out-of-n to k-out-of-$2k$ code and then a checker for the resulting k-out-of-$2k$ code. The design of the translator is very simple and can be achieved by a row of $2k$ OR gates, [1]. Two designs for the TSC k-out-of-$2k$ code checkers have been proposed. In [1] the merged version of this checker is described which has three gate levels and requires 2^k input test patterns. Reddy provides an alternative design for the k-out-of-$2k$ checker [10]. This design is cellular and requires only $2k$ test patterns, but has more delay. For our application, where *all* of the n possible 1-out-of-n codeword inputs to LC are required for testing L, such a saving in number of test patterns for testing C cannot be taken advantage of. Furthermore, since we are using PLA implementations for our circuits and the cellular design in [10] does not lend itself to such an implementation, we only consider Anderson's design for comparison purposes. To estimate the area taken up by the PLA's, we use the standards of [8]. The unit of area is immaterial because our purpose is to provide a relative estimate. Recall that a two-rail checker with many input lines can be implemented as a tree, with each node having a smaller number of inputs. Each PLA is assumed to contribute the equivalent of two gate delays.

An Example

Consider designing a TSC checker for the 1-out-of-10 code.

First Method: Translate 1-out-of-10 into a two-rail code, as in Fig. 5. The translator will have $\lceil \log_2 (10) \rceil = 4$ pairs of outputs. Thus the area of the translator is $10 * 2 * 4 = 80$.

Next, use a two-rail checker C as in Fig. 6. Each of $C1$, $C2$, and $C3$ circuits has 2 input pairs. These will require an additional 72 units of area. Thus, the entire 1-out-of-10 checker has an area of 152 units. The total delay through this checker is 5 gate delays (note that circuit L has 1 gate delay).

Second Method: Translate 1-out-of-10 into k-out-of-$2k$ code [1]; here, $k = 3$. Thus, the area of the translator is 60 units. The k-out-of-$2k$ checker implemented by two PLA's has an area of 196 units. Thus, the total cost in area for this method is 256 units.

Therefore, we save substantially in area by choosing the first, rather than the second method. However, we lose in terms of speed, since the second design has only 4 gate delays.

TABLE III
Comparison of the Two Designs When n is a Power of Two

	AREA COST		GATE DELAYS	
n	P	A	P	A
4	40	70	3	4
8	112	246	3	4
16	288	292	3	4
32	704	796	3	4
64	920	1052	5	4
128	2040	2706	5	4
256	4440	6474	5	4
512	9472	9546	5	4
1024	20832	22520	5	4

P refers to the design of the present work;
A refers to the design of [1].

Fig. 7 shows the area cost for the above two methods as a function of n. From Fig. 7 we conclude the following.

1) For many values of n, $n < 1025$, our design has less area cost, but more delay, than that of [1].

2) The "jumps" in the area of the circuit of [1] occur when k is incremented such that the resulting k-out-of-$2k$ code may accommodate the corresponding 1-out-of-n code. At these points the relative saving in area is the best for our method.

3) For $n > 1024$ the areas of the translators overshadow the areas of the code checkers in both methods. Further, the translator of our design is always larger than that in [1]. Thus, for $n > 1024$ the latter method should be used. However, such large values of n occur rarely, if at all, in practice.

If n is a power of 2, where any implementation of C is possible (Fig. 2), that is, where the extra PLA level of Fig. 6 is not required, the delay for our design can be reduced to only 3 gate delays. In fact, for a few such values of n our design also requires less hardware. These results are tabulated in Table III. The reason for the increase in delay

for $n \geq 64$ is that to reduce the area the two-rail checker C is implemented as a tree with an extra PLA level.

IV. Conclusion

We have presented a new design for TSC 1-out-of-n code checkers. It is shown that for many values of $n < 1025$ our design would have less hardware cost than that of an existing scheme [1]. In fact, for a few important cases (when $n \leq 32$ is a power of 2), our checker is also faster than the other design.

Since the initial conclusion of this work, two new designs for TSC 1-out-of-n code checkers have come to the author's attention. The first is a work by Kotočová [5]. Here a TSC design for some 1-out-of-n code checkers has been described. The checker has three gate delays and can be minimized according to the number of gates or the number of gate inputs. The n input lines are grouped according to certain rules and are fed into, say, an OR-AND-OR circuit. No estimate of hardware cost is provided. The second design, described in [9], also groups the n inputs, but now into two matrices. Then algebraic manipulations, such as shifts and scalar multiplication, are done on the columns of these matrices according to an algorithm. This is then translated into a circuit implementation which always has four gate delays, and is shown to be superior to Anderson's design in the total number of gate inputs that it requires.

Finally, none of the works cited so far, including this one, provides a TSC checker for the 1-out-of-3 code. David has designed such a checker using memory elements [3]. This design involves a translation from the 1-out-of-3 code into the 1-out-of-4 code, using two bits of memory. The resulting code can then be checked by a TSC checker for the 1-out-of-4 code. Recently, another solution to this problem has been suggested [4]. Here the 1-out-of-3 code is combined with any available m-out-of-n code to obtain a reduced $(m + 1)$-out-of-$(n + 3)$ code with p code words. Then the resulting code is translated into the 1-out-of-p code, for which a TSC checker can be designed. To date, however, no combinational TSC checker has been designed for the 1-out-of-3 input.

Acknowledgment

Many thanks to Prof. E. J. McCluskey for his guidance and support. Also, special thanks to Prof. J. Wakerly for his many invaluable comments and suggestions.

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] W. C. Carter and P. R. Schneider, "Design of dynamically checked computers," in *Proc. 4th Congress IFIP*, vol. 2, Edinburgh, Scotland, Aug. 5–10, 1968, pp. 878–883.

[3] R. David, "Totally self-checking 1-out-of-3 checker," *IEEE Trans. Comput.*, vol. C-27, pp. 570–572, June 1978.

[4] P. Golan, "A new totally self-checking checker for 1-out-of-3 code," in *Proc. 4th Int. Conf. on Fault-Tolerant Syst. Diagnostics*, Brno, Czechoslovakia, Sept. 28–30, 1981, pp. 246–247.

[5] M. Kotočová, "Design of totally self-checking check circuits for some 1-out-of-n codes," in *Proc. 4th Int. Conf. on Fault-Tolerant Syst. Diagnostics*, Brno, Czechoslovakia, Sept. 28–30, 1981, pp. 241–245.

[6] M. A. Marouf and D. A. Friedman, "Efficient design of self-checking checkers for m-out-of-n codes," in *Proc. 7th Annu. Symp. on Fault-Tolerant Comput.*, Los Angeles, CA, June 28–30, 1977, pp. 143–149.

[7] V. I. Maznev, "Design of self-checking $1/p$-checkers," *Automat. Remote Contr.*, vol. 39, part 2, pp. 1380–1383, Sept. 1978; transl. from *Avtomat., Telemekh.*, pp. 142–145, Sept. 1978.

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

[9] V. Rabara, "Design of self-checking checker for 1-out-of-n code ($n > 3$)," in *Proc. 4th Int. Conf. on Fault-Tolerant Syst. Diagnostics*, Brno, Czechoslovakia, Sept. 28–30, 1981, pp. 234–240.

[10] S. M. Reddy, "A note on self-checking checkers," *IEEE Trans. Comput.*, vol. C-23, pp. 1100–1102, Oct. 1974.

[11] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*. New York: Elsevier, 1978.

[12] S. L. Wang and A. Avizienis, "The design of totally self checking circuits using programmable logic arrays," in *Proc. 9th Annu. Symp. on Fault-Tolerant Comput.*, Madison, WI, June 20–22, 1979, pp. 173–180.

Watchdog Processors and Structural Integrity Checking

DAVID JUN LU

***Abstract*—The use of watchdog processors in the implementation of Structural Integrity Checking (SIC) is described. A model for ideal SIC is given in terms of formal languages and automata. Techniques for use in implementing SIC are presented. The modification of a Pascal compiler into an SIC Pascal preprocessor is summarized.**

***Index Terms*—Control flow, error detection, Pascal, structural integrity checking (SIC), structured programming, watchdog processor.**

I. Introduction

The demand for detection of errors in computers during the execution of computer programs has increased. It has become quite difficult to devise tests that effectively exercise VLSI computer systems because of the great complexity of the hardware and software. The increased complexity has introduced new varieties of errors and increased the number of possible errors. Meanwhile, the introduction of many new computer applications has increased dependence on computers.

Techniques using circuit redundancy and coding theory, such as parity checking of memory, can be used within VLSI chips to provide detection of low-level errors. The increased circuit density, however, reduces access to the internal states of each module. At the system level, there remains the problem of determining ways in which VLSI components can be employed to detect errors, especially errors in the overall behavior of a computer system.

The use of auxiliary processors or processes for error detection is an extension of the well known technique of watchdog timers [18], [16]. Major issues in the design of watchdog processors are the selection of operations to be checked for errors, the means of encoding and transmitting information from the main processor to the watchdog, and the programming of the watchdog. Structural integrity checking (SIC) consists of analyzing high-level control flow structures in computer programs, attaching labels to these structures for error detection, and checking the integrity of these structures at run time by using a watchdog processor or process. SIC uses syntax driven methods to encode program structures for error detection, and to generate the program for the watchdog.

SIC detects errors in the high-level behavior of a computer system

Manuscript received September 28, 1981; revised January 12, 1982. This work was supported in part by the National Science Foundation under Grant MCS-7904864, DARPA Contract MDA903-79-C-0680, and the U.S. Army Electronics Research and Development Command under Contract DAAK20-80-C-0266. Earlier versions of this work were presented at the National Electronics Conference, Chicago, IL, October 1980 and the IEEE Distributed Data Acquisition, Computing, and Control Symposium, Miami Beach, FL, December 1980. Earlier versions also were published in "Concurrent testing and checking in computer systems" (Ph.D. dissertation), Stanford University, June 1981; and Center for Reliable Computing, Tech. Rep. 81-5, Stanford University, July 1981.

The author is with the Center for Reliable Computing, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, CA 94305.