Fault-Tolerant Computing: An Introduction and a Viewpoint

WILLIAM C. CARTER

AFTER approximately 20 years of obscurity, the field of fault-tolerant computing was revived by the formation of the IEEE Technical Committee on Fault-Tolerant Computing, by a series of articles in COMPUTER for January/February 1971, by the 1971 International Symposium on Fault-Tolerant Computing in Pasadena, Calif., and by an IEEE TRANSACTIONS ON COMPUTERS Special Issue on Fault-Tolerant Computing in November 1971. Interest and activity continued apace, and the 1972 International Symposium on Fault-Tolerant Computing was held in Newton, Mass. Most of the excellent papers in this second IEEE TRANSACTIONS Special Issue on Fault-Tolerant Computing were presented at that symposium. As an introduction to these papers, consideration of the ultimate goals of this discipline and the universe in which our work is being done is most appropriate.

J. von Neumann provided the initial theoretical goals and universe for fault-tolerant computing, and the following series of quotations shows the penetration of his thought.

The basic problem of fault tolerance caused by component failures is well defined.

In a complicated network, with long stimulus-response chains, the probability of errors in the basic organs makes the response of the final outputs unreliable, i.e. irrelevant, unless some control mechanism prevents the accumulation of these basic errors. [1, p. 346]

His method of attack was well stated and remains particularly useful.

... von Neumann believed that in starting a new science one should begin with problems that can be described clearly, even though they concern everyday phenomena and lead to well known results, for the rigorous theory developed to explain these phenomena can provide a base for further advances. [2, p. 20] His problems of reliability and self-reproduction are of this kind.

His statement on current (1950) engineering practice shows the problems of having to deal immediately with single faults (still the case too often today).

Our modus procedendi with respect to malfunctions in our artificial automata is entirely different (from that of living organisms). Here the actual practice, which has the consensus of all experts of the field, is somewhat like this: Every effort is made to detect (by mathematical or by automatic checks) every error as soon as it occurs. Then an attempt is made to isolate the component that caused the error as rapidly as feasible. This may be done partly automatically, but in any case a significant part of this diagnosis must be effected by intervention from the outside. Once the faulty component has been identified, it is immediately corrected or replaced. [3, p. 305]

He proposed that we learn enough to imitate nature in our treatment of the errors generated by component faults.

The basic principle of dealing with malfunctions in nature is to make their effect as unimportant as possible and to apply correctives, if they are necessary at all, at leisure. In our dealings with artificial automata, on the other hand, we require an immediate diagnosis.

... All of this comes back to one thing. With our artificial automata we are moving much more in the dark than nature appears to be with its organisms. We are, and apparently, at least at present, have to be, much more "scared" by the occurrence of an isolated error and by the malfunction which must be behind it. Our behavior is clearly that of overcaution, generated by ignorance [3, p. 306]

Our current definition of fault-tolerant computing, the ability to execute specified algorithms regardless of hardware failures [5], is also anticipated.

It is very likely that on the basis of the philosophy that every error has to be caught, explained, and corrected, a system of the complexity of the living organism would not run for a millisecond. Such a system is so well integrated that it can operate across errors. An error in it does not in general indicate a degenerative tendency. The system is sufficiently flexible and well organized that as soon as an error shows up in any part of it, the system automatically senses whether this error matters or not. If it doesn't matter, the system continues to operate without paying any attention to it. If the errors seem to the system to be important, the system blocks that region out, by-passes it, and proceeds along other channels. The system then analyzes the region separately at leisure and corrects what goes on there, and if correction is impossible the system just blocks the region off and by-passes it forever. The duration of operability of the automaton is determined by the time it takes until so many incurable errors have occurred, so many alterations and permanent by-passes have been made, that finally the operability is really impaired. This is a completely different philosophy which proclaims that the end of the world is at hand as soon as the first error has occurred. [2, p. 71]
Most people are well acquainted with von Neumann's introduction of masking redundancy. (The usual engineering application has been triple modular redundancy, TMR.) However, the original definitions are illuminating.

8.3.1. The heuristic argument. The basic idea in this procedure is very simple. Instead of running the incoming data into a single machine, the same information is simultaneously fed into a number of identical machines, and the result that comes out of a majority of these machines is assumed to be true. It must be shown that this technique can really be used to control errors. . . . [1, pp. 347-348]

von Neumann's statements that the proper use of redundancy is to allow operation to proceed pinpoint the problem on which much of our present work has focused.

von Neumann proceeded to discuss what he thought was a deeper cause of the discrepancy in size between natural and artificial automata. This is that many of the components of the natural system serve to make the system reliable. . . . And to say that they (redundancies) are precautions against failure is to overstate the case, to use an optimistic terminology which is completely alien to the subject. Rather than precautions against failure, they are arrangements by which it is attempted to achieve a state where at least a majority of all failures will not be lethal. There can be no question of eliminating failures or of completely paralyzing the effects of failures. All we can try to do is to arrange an automaton so that in the vast majority of failures it can continue to operate. [2, pp. 58-60]

Those computing areas where the penalty for failure is high, such as aerospace and, to a lesser extent, real-time applications, accepted the goals and universe defined so well by von Neumann. The papers in this issue vigorously present new ideas in this universe. The basic problem of logic test determination for multiple component faults, introduced by LSI failure modes, is considered in the first two papers. Clegg [6] extends the idea of fault equivalence classes to introduce SPOOF's—a new means of characterizing both network structure and output function in a single algebraic expression. Using SPOOF's, the effects of multiple "stuck-at" faults may be determined, or alternatively, the failures causing a specific erroneous output may be listed. Du and Weiss [7] use the information flow concept of Hartmanis and Stearns to develop the fault tree of a circuit. Then they examine circuit faults neither individually nor in indistinguishability classes, but rather in terms of the classes defined by the fault tree, which is determined by the subfunctions from which the circuit is constructed. This leads to a fault detection test set that detects all possible multiple faults. Breuer [8] considers the problems of the characterization and detection of intermittent faults in digital circuits. This previously ignored problem is important since, in many technologies (new and old), intermittency is a predominate mode of failure (this is the main reason for the current popularity of the instruction retry feature in current computers). The characterizing fault pattern for intermittent faults includes permanent faults as special cases. It is shown that no single test sequence will detect all possible fault patterns, confirming the practical difficulties well known to field engineers. This interesting beginning of a theory concerning tests for intermittent failures will enable better diagnostic tests to be devised.

Two papers on modeling leave the familiar field of redundant processors and explore new ground. Hecht [9] proposes an economic model that considers the computer as a tool for the spacecraft systems engineer. Any increase in cost for computer reliability is evaluated against the expected decrease in cost of system failure. Using this less parochial viewpoint, a computer memory configuration is optimized for a 5-year space mission. Arnold [10] considers system coverage for a repairable redundant system. The magnitude of system coverage is given by the size of the set of single faults that will cause a system failure in a redundant (usually duplicated) system. As in the case of self-repairing systems, even a small number of such faults will severely degrade the mean-time-to-system-failure and expected down time for an otherwise highly reliable system.

The next set of papers consider fault-tolerant system design. Cook, Sisson, Storey, and Toy [11] consider the problem of designing a self-checking microprogram control for a repairable duplicated processor. The techniques appropriate to the traditional fault-tolerant universe are well used. Component failure types are analyzed to determine the most probable failure mode (multiple faults are highly probable). The checking techniques are matched to the expected type of faults, and m-out-of-n-codes are chosen. However, in practical cases, reliability is not all important. Performance and reliability are both considered, and best practical results for multiple fault detection are obtained by interleaving control bits and addresses in the control store.

Anderson and Metze [12] describe in detail the general design methods for totally self-checking check circuits for m-out-of-n codes, including a thorough fault analysis. Totally self-checking m-out-of-n checkers provide an error indication whenever the input is not an m-out-of-n code, or whenever a fault occurs within the checker itself. Since the checker checks itself, there is no need for additional maintenance access or periodic exercise of the checker to verify its ability to detect errors.

Bouricius et al. [13] do a model-directed design of a bubble-memory organization with self-checking translators to achieve high reliability. The objective of this study is to develop good fault-tolerant design and analysis methods adequate for newly emerging technologies, and to prove the practicality by example. The reliability modeling study justifies the design philosophy adopted of employing memory data encoding and a translator to correct single group errors and detect double group errors, together with a spare memory submodule and switching replacement to enhance the overall system reliability. New modeling methods, a new code translator, and new switching techniques are employed. Lack of information about the failure modes of the new bubble technology is overcome by a parametric study. A basic problem in replacing modules by a standby spare (as proposed in the preceding
paper) is the reliability of the circuits resulting from the spare switching strategy used. Siewiorek and McCluskey [14] define a switching strategy as the procedure used to decide which spares are switched in to replace failed units in the NMR core of a system protected by hybrid redundancy. In order to compare various switching strategies, a design-independent criterion, one that is independent of logical implementation, the number of states in the switch, is used. This first theoretical study of this problem develops an optimal strategy if every spare can be switched into every voter position (totally assigned). Partially assigned switching strategies are considered, and it is shown that designs where all spares are assigned to the same \( t + 2 \) of the \( 2t + 1 \) voter positions have as good a reliability as the more complex totally assigned switching strategy. Effects of the simplified switch on the switch and modules reliability are considered, and various configurations are compared with surprising results. Two improvements in coding techniques that influenced the bubble-memory study are described by Carter, Duke, and Jessep [15]. First, within the fixed structure of a Hamming SEC/DSED code, an improvement can be obtained in circuit cost and operation speed over more conventional code implementations. Second, the mechanics of error correction in a fault-tolerant computer may be carried out via conventional hardware means, or by use of the existing system facilities, such as the combination of the microprogram unit, local store, and the arithmetic-logic unit. The techniques depend upon the multiple use of hardware for distinct functions.

In addition to papers describing techniques necessary for the system designs considered, other interesting studies were available. The advantages of hybrid redundancy (N modular redundancy with standby sparing) depend upon the design of the majority voters, disagreement detectors, and circuits implementing the switching strategy. Siewiorek and McCluskey [16] use new hardware technology to propose an interactive cell switch that is demonstrated to save at least 25 percent, and more than 80 percent in some instances, of the complexity of a switch design presented elsewhere in the literature. The use of threshold rather than majority voters is considered, and is shown to yield a simpler design in some cases. Finally, five different switch designs are compared on a cost and complexity basis.

In a more general view, Osman and Weiss [17] use the \((n, m, r)\) basis for decomposing switching functions to give design procedures that permit more economical realizations than can be obtained using classical parity and triple modular redundancy schemes for obtaining logic circuits with the corresponding property. Dynamically self-checked or fault-tolerant realizations of switching functions and sequential machines are proposed under the fault tree circuit fault model that permits arbitrary logic faults in a single logic module, where the modules are explicitly defined. These realizations permit considerable logic sharing, organized around an \((n, m, r)\) basis for decomposing switching functions, and thus, realizations using fewer circuits.

Not much has been published about the error-correcting properties of redundant residue number systems. Bari and Maestrini [18] use a new more natural approach that simplifies previous proofs and allows new results to be proved. In particular, a single redundant modulus will allow the correction of the set of errors affecting a single bit in the code.

The papers chosen for this Special Issue and those appearing in previous issues indicate clearly that methods for hardware design, especially for central processors, are under control. New vigorous ideas are being expanded, and designers no longer have the excuse of saying that there are but few techniques and that these have unknown cost. However, there is grave doubt that this is in any way a complete universe or sufficient set of goals. When von Neumann spoke as a user, not a designer, he made the following statement.

When a problem in pure or in applied mathematics is "solved" by numerical computation, errors, that is deviations of the numerical "solution" obtained from the true, vigorous one, are unavailable. Such a "solution" is therefore meaningless, unless there is an estimate of the total error in the above sense. [4]

Such errors are caused by more than hardware faults. Incorrect programs or programs that contain fallacies will result in errors that render the results meaningless. Work was begun by Floyd and Manna on proving programs valid. For an excellent tutorial, see the paper by Elspas, Levitt, Waldinger, and Waksman [19], and for the current state, see London's paper [20]. Dijkstra [21] and others advocate a programming discipline that systematically constructs the desired program. Another source of errors is computer system flaws. A computer system flaw may arise from an incorrect implementation of specifications, a race or hazard previously undetected, or a timing problem that first occurs with a particular information pattern. The error is not caused by component deterioration with time, but by a mistake in design, and is corrected only by correcting the design. As computer systems become more complex, the probability of logical mistakes increases, and empirical methods of debugging become less adequate. As remarked by Arnold [10], other flaws may arise from the efforts of human operators or by badly defined human operating procedures that are part of the total operating computer system. Finally, flaws can arise from the finiteness of the computer itself—roundoff error, truncation error, slow convergence—all the error problems of numerical analysis. If correct answers to problems are desired, then such flaws and fallacies must be considered, as well as component faults. Moreover, with the advent of LSI, such errors will become more important, as less of the cost will be concentrated in hardware, particularly in the processor area. The hardware system will become decentralized, with I/O costs predominating—and thus these fault-tolerance problems must be emphasized.

M. G. Smith has given the following examples of the impact of LSI [22].

For the purpose of this illustration, we assume that the configuration, in contemporary terms, uses two large processors, plus one stand-by/batch processor, additional memory, and other supporting hardware. Display terminals with storage and logic are also assumed. The re-
required components costs not considered previously were drawn from Auerbach [23] sources, with nominal price reductions assumed.

<table>
<thead>
<tr>
<th>A Hypothetical 5000-Terminal Configuration</th>
<th>Cost Circa 1970</th>
<th>Cost Late '70's</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminals (5000)</td>
<td>55 percent</td>
<td>57 percent</td>
</tr>
<tr>
<td>Lines and Modems</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Multiplexors</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>CPU Processor (3)</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>Channels, etc.</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Main Memory—4Mbytes</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>Bulk Memory—16Mbytes</td>
<td>100 percent</td>
<td>100 percent</td>
</tr>
<tr>
<td>Files</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>Other</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

Perhaps the most conspicuous feature of this table is the dominance of the terminal costs. Second, perhaps, is the relative disappearance of the main and bulk memory costs. Next is the relatively small proportion of costs dedicated to transmission.

Fig. 1, also from Smith, shows the increasing importance of programming and operating costs, and thus the growing importance of operating flaws and programming fallacies. This change in universe necessitates a new definition for fault-tolerant computing. The proposed change is that fault-tolerant computing be defined to be the ability to execute specified algorithms correctly regardless of hardware failures, total system flaws, or program fallacies. This change in definition reflects the change in emphasis documented above. It also follows the topics proposed for the 1973 International Symposium on Fault-Tolerant Computing. For methods of attack, we can again follow von Neumann [2, p. 20].

... von Neumann believed that in starting a new science one should begin with problems that can be described clearly, even though they concern everyday phenomena and lead to well known results, for the rigorous theory developed to explain these phenomena can provide a base for further advances.

ACKNOWLEDGMENT

The Guest Editor wishes to thank all the referees and contributors for their help and courtesy. Particular thanks are due to Dr. R. A. Short, the Editor-in-Chief of this TRANSACTIONS, for his wise advice and help.

REFERENCES

[9] H. Hecht, “Figure of merit for fault-tolerant space computers,” this issue, pp. 246-251.
Use of SPOOF's in the Analysis of Faulty Logic Networks

FREDERICK W. CLEGG

Abstract—In general, one cannot predict the effects of possible failures on the functional characteristics of a logic network without knowledge of the structure of that network.

The structure- and parity-observing output function (SPOOF) described in this paper provides a new and convenient means of characterizing both network structure and output function in a single algebraic expression.

A straightforward method for the determination of a SPOOF for any logic network is demonstrated. Similarities between SPOOF's and other means of characterizing network structure are discussed. Examples are presented that show several useful applications of this new tool. It is shown that the SPOOF provides an easy means by which the effects of any "stuck-at" fault—single or multiple—on the functional characteristics of a logic network can be determined. In addition, one may, using SPOOF's, determine just which faults, if any, can occur in a network to affect its output function in a given way. Other likely applications of SPOOF's, not yet fully explored, are indicated and areas for future research are suggested.

Index Terms—Fault detection and diagnosis, fault equivalence, fault-tolerant computing, reliability of digital systems, SPOOF's.

I. INTRODUCTION

This paper will present a new tool that is of potential use to those who wish to study the characteristics of switching circuits in which one or more of the circuit's components are subject to failure. The techniques presented are applicable to gate-type combinational logic networks. The type of failures considered is that of "stuck-at" faults—conditions whose logical effect can be specified by saying that the levels on some signal lines in the network in which they are present are stuck-at constant logical values. The condition, say $F_i$, in which some line $j$ in a network is stuck-at a constant logical 0 and some line $k$ in the same network is simultaneously stuck-at a logical 1 will be written as $F_i = j/0 , k/1$. The set of all possible stuck-at faults that can occur in a logic network is denoted as $\mathcal{F}$. In this and in all other respects, the notations used in this paper are compatible with those used [1]–[3].

In the analysis of logic networks, it is possible to obtain a certain amount of knowledge about the effects of certain faults on particular networks and their output functions given only knowledge of the Boolean functions that these networks implement. Unfortunately, however, one generally needs infor-

1 See, for example, [2, theorems 6.9(a) and (b)].