Note that typically we have

ORD-OUT(SO, S) = >SYNT-OUT(SO,S) = >ERR-OUT(SO, S).

IV. Conclusion

This correspondence attempts to formalize the use of executable assertions in the detection of and recovery from errors in programs, by introducing (respectively) the notions of self-checking programs and self-stabilizing programs.

A formal verification system is provided, which enables one to formally verify that a given program is self-checking. The self-checking property of a program can be viewed in two ways as follows:

- first, as the ultimate property of an asserted program,
- second, as an economical alternative to correctness. When a program is so complex that its proof of correctness is unreasonably expensive, one may want to prove instead that the program is self-checking by proving that its frame of assertions is correct in the sense defined in this correspondence. A frame of assertions is typically much shorter than the asserted program that it supports.

Self-stabilizing programs are derived from self-checking programs by adding a recovery capability and much of the results and tools derived for self-checking programs apply to self-stabilizing programs. The total recovery and the various philosophies of partial recovery are discussed, and the relationship between the "power" of the recovery routines and the overall quality of the program is shown. More research is under way concerning this relationship.

Acknowledgment

The author wishes to thank the anonymous reviewers for their comments on the initial manuscript of this correspondence.

References

[1] D. Andrews, "Using executable assertions for testing and fault tolerance," presented at 9th Fault-Tolerant Comput. Symp., Madison, WI, June 20–22, 1979.

[2] R. L. Glass, "A benefit analysis of some software reliability methodologies," *ACM SIGSOFT, Software Eng. Notes*, vol. 5, Apr. 1980.

[3] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. Ass. Comput. Mach.*, vol. 2, Oct. 1969.

[4] K. Jensen and N. Wirth, *PASCAL User Manual and Report*, 2nd ed. New York: Springer-Verlag, 1978.

[5] A. Mili, "Self-checking programs: An axiomatic approach to the validation of programs by the use of assertions," Ph.D. dissertation, Dep. Comput. Sci., Univ. of Illinois, Urbana, Dec. 1980.

[6] B. Randell, "System structure for software fault tolerance, *IEEE Trans. Software Eng.*, vol. SE-1, June 1975.

The Containment Set Approach to Upsets in Digital Systems

ROBERT E. GLASER AND GERALD M. MASSON

***Abstract*—Fault analysis of digital systems is highly dependent upon the fault model employed. Much previous work utilizes fault models known to contain inaccuracies in order to permit mathematically tractable analysis. In this correspondence a new approach is taken which combines faults, hardware, and software together into one overall model. This new model is shown to be useful for the consideration of intermittent/transient faults. It supports a new method, based on the novel concept of a containment set, for realizing transient fault tolerance without massive redundancy. It also allows for a new approach to system fault tolerance evaluation and validation which uses a transition matrix which is defined in terms of the containment set.**

***Index Terms*—Containment set, control system, error, fault, fault tolerance, transition matrix, upset.**

Manuscript received October 2, 1981; revised January 7, 1982. This work was supported by NASA under Grant NSG 1442.

The authors are with the Department of Electrical Engineering and Computer Science, The G.W.C. Whiting School of Engineering, The Johns Hopkins University, Baltimore, MD 21218.

I. Introduction

In this correspondence the effects of intermittent and transient faults on digital systems are considered. In particular, a theory is developed which can be directly applied to microprocessor based control systems. Several approaches to fault-tolerant microprocessor design have been reported in the literature. For example, the use of CPU modifications to enhance fault detection is described in [1]. This self-checking VLSI microprocessor detects internal faults and flags external circuitry, which can then initiate recovery. Variations of the 6800 CPU which can detect invalid op-code fetches have been produced. Discussion of the effectiveness of such limited detection mechanisms can be found in [2] and [3].

Probably the simplest and most prevalent external hardware addition used for fault detection is a *watchdog timer*. In normal operation, the microprocessor periodically pulses the external timer. The software is written such that the processor is guaranteed to pulse the timer before a specified time elapses. The timer is retriggerable, so that under normal operation it never times out. Should the processor fail to trigger the watchdog within the allotted time period, the timer initiates recovery action, which can be as simple as resetting the processor, or as complex as calling a test program which thoroughly exercises the entire system and logs the results. A watchdog timer implementation can be found in [4], and discussion of the effectiveness of watchdog timers can be found in [2] and [3].

Triple modular redundancy (TMR) has been applied to microprocessors. This form of massive redundancy provides more effective recovery than the simpler, less costly schemes. A TMR design around the 8080 CPU is described in [5]. Determining the optimum points to place voters is not a simple task. Memory can be treated in a variety of ways with regard to voter placement, and voter reliability is critical. Reliability improvement with various connections is reported in [6].

The goal of this correspondence is to describe an alternative approach to fault modeling leading to a new tolerance method which does not utilize massive redundancy, yet provides provable tolerance to intermittent and transient faults.

II. Upset Modeling

A particularly elusive problem relevant to the area of fault-tolerant computing is that of obtaining a useful and accurate fault model. Detailed models of specific fault sources tend to have little application to overall system behavior prediction, and general mathematically tractable fault models bear little resemblance to real world situations. The approach to be taken in this correspondence is to model the entire digital system consisting of the original fault-free system with the addition of faults. Instead of attempting to combine two independent models—the system model and a fault model—an integrated view will model them as a single entity.

A. Faults, Errors, and Upsets

Failures are circuit related, and are due to problems such as opens and shorts. Intermittent failures are caused by the combination of internal failures and external stimuli, causing the circuit to intermittently operate improperly. These circuit failures lead to faults, errors, and upsets.

Faults: A *fault* is a logical difference at the site of a circuit failure between faulty and fault-free devices [7]. An *intermittent fault* is, in part, due to a physical failure, and is expected to make transitions between active and inactive states during the lifetime of the equipment. A *transient fault* is not due to a circuit failure, but an envi-

ronmental condition which the equipment was not designed to tolerate. This is sometimes referred to as a malfunction [8].

The fault classes of concern here are intermittent and transient (I/T) faults. Since these faults are caused either in part, or totally, by environmental factors, the source of faults is considered to be the hostile environment. Information at this fault source level is not digital, but analog in nature. For example, the actual fault source can be an internal logical signal modification such that it satisfies neither the logic high nor low digital requirements. In such a case, it is not possible, in general, to predict how a digital circuit will react. Moreover, while the digital circuit is a clocked, synchronous digital system, the actual faults caused are seldom "well behaved" in the sense that the fault is only synchronously active or inactive with system clock. The well-behaved assumption greatly simplifies predictions of the fault's operational implications, but clearly does not reflect real situations.

Errors: An incorrect logic value at a fault site propagates to other parts of the circuit. These logic differences between faulty and fault-free systems are called errors, with the implication that an error at a failure site is called a fault. Errors can be considered on all lines of a circuit; they appear on all differing lines between faulty and fault-free units internal to the control system itself.

It has been seen that the fault level is not useful as a description of fault/system interaction in complex digital circuits because observation is not possible at this level. The error level is one step removed from the fault level. An I/T fault can cause a state change in the digital system. A continuous string of errors can result if the faulty circuit is not forced into the correct state, that of the fault-free circuit. Simple loss of synchronization will yield endless errors, after the I/T fault has disappeared.

In modern digital systems, consisting of LSI integrated circuits, all circuit lines are not accessible; only pins of the packages are observable. In the attempt to find the lowest observable level from which to view faults, a complete description of errors is not usable, since most of these observation points are not available. A subset of circuit lines are available, but even accepting error propagation latency the conventional definition of error suffers from the loss of synchronization problem. To make errors a useful point of observation, a new definition is made such that logic signal differences at the observation points between faulty and fault-free systems will only be interpreted as errors when the digital system is actually being driven by a fault. If a set of logic signal differences are observed over one clock cycle, then an *error* will be noted over the next clock cycle only if there are logic signal differences at the observation points between the faulty circuit and a fault-free circuit which has been forced into the exact same state as that of the faulty circuit at the end of the previous clock cycle. However, as will now be discussed, the upset level is a more useful perspective for the goals of this research.

Upsets: A digital system is designed to perform some function. The primary concern is how faults affect the performance of the design function. The presence of faults perturbs the system or "upsets"it. The viewpoint which observes fault effects at this higher, functional level, will be referred to as the *upset* level. The transfer function described by the system outputs, in relation to the signal inputs, is the observation point for the upset level.

At the upset level, a system is viewed as responding to the arrival and departure of an I/T fault in two stages—the same as standard system theory separates a general system's response to any input. This system's response to the I/T fault input will be the same as any other system's response to an input: there will be components both of the transient response and the steady-state response. The difference here is that with standard system theory, the steady-state response is due to a driving input, which remains, and the transient response is due to the system's response to the input change. In the case of the control system's response to I/T fault inputs, the I/T fault arrives and then disappears. In the purest sense then, the system's response would be composed entirely of the transient response with no steady-state response. However, for this digital system driven with I/T fault inputs, state changes can cause lasting effects on the system after the departure of the I/T fault. These effects will be classified as *steady-state* effects at the upset level. Fault effects present during and shortly after the time when the system is being driven by the I/T fault will appropriately be called *transient* effects. (The word "transient" here is used differently from that in the term "transient fault." There can be both "transient" and "steady-state" responses to a single "transient fault." The single word "transient" is retained in this correspondence for each of these separate meanings because both are in agreement with standard usages.)

It is possible to observe transient and steady-state effects of I/T faults because of the functional interpretation of the upset level. Typically, transient responses are incorrect data values, loss of time or synchronization, or skipping a computation step. For example, consider a digital control system behaving partially as a moving average filter; the outputs depend primarily on current and recently applied inputs. Effects of an input disappear with time. For this type of control system, transient output perturbations due to an I/T fault will also vanish over a period of time. This will be true only if the system function remains unchanged by the fault.

Steady-state responses are functional transformations. After the departure of the I/T fault, the system is no longer performing the same transfer function between its signal inputs and outputs as prior to the I/T fault arrival. For the moving average filter example, a steady-state response to an I/T fault modifies the filter algorithm. Since the filter may no longer be a moving average type at all, there can be no expectation that output perturbations will disappear with time; in fact, with a function change the fault effects will not disappear.

A sophisticated control system could monitor internal states and external events to determine if I/T faults have caused a transient effect on data processing, and initiate recovery procedures when detected. Transient system responses can be tolerated in this fashion. Steady-state system responses to an I/T fault would transfer execution from the control algorithm, after which there would be no reasonable hope for system recovery. Because of the relatively drastic consequences of a single I/T fault that the steady-state fault response can show at the upset level, as compared with the transient response, only the steady-state effects of I/T faults will be considered here.

B. Containment Sets

To concentrate on the steady-state fault response of a digital system, the system's functional I/O relationships must be characterized. When the operation of the system can be completely described in terms of a finite set of mutually exclusive functional states covering all possible transfer functions, this set of functional states will be referred to as the *containment set*. All possible system states must cause functional operation of one of the elements of the containment set. This set must include all possible valid functional states of the fault-free system, but this set must also include invalid functional states, not explicitly designed into the system but into which the system can nevertheless be driven by an I/T fault. Ignoring transient effects of I/T faults, the steady-state effects can be described in terms of the containment set; and this set is completely defined by the structure of the fault-free system. No fault information or conventional system fault analysis is needed to obtain the containment set. This advantageous condition is a result of not considering solid (permanent) faults, and then ignoring the transient effects of the remaining faults. Nevertheless, this containment set is the basis for the analysis of lasting effects of I/T faults, and the likelihood of practically determining containment sets can be seen to be promising. It is not at all clear that a finite containment set exists for all digital systems; indeed, in general, this is not true. However, it has been found that although some system modifications may be necessary to produce a useful, finite containment set, these modifications are not necessarily overly restrictive [9].

Therefore, the steady-state fault response can be described in terms of the containment set, and the transitions between elements of this set are of acute interest.

Transition Matrix: While it is clear that the types of I/T faults

have no bearing on the makeup of the containment set, nevertheless the faults do induce the transitions among the functional states in the set; hence, there is no expectation that the fault types do not affect these transitions. However, for a given fault class, a *transition matrix* $T = [p_{ij}]$ can be used to specify the probabilities of each of the transitions. Let the containment set be $\{L\} = \{L_0, L_1, \cdots L_{n-1}\}$. Then the entry p_{ij} in T gives the probability of a transition from containment state L_j to containment state L_i, given that an error is detected. The transition matrix will be dependent upon the choice of observation points for the detection of errors.

Thus, a linear difference equation defines the functional state behavior of the system

$$L(k+1) = TL(k), \qquad \text{where } L(k) \text{ is a column vector}$$

$$L(k) = \begin{bmatrix} p_0(k) \\ p_1(k) \\ \vdots \\ p_{n-1}(k) \end{bmatrix}$$

such that $p_t(k)$ is the probability of the digital system being in containment state $L_t \in \{L\}$ after k upsets. If L_0 were the known initial state, then after k upsets the functional state probabilities are given by

$$L(k) = T^k \begin{matrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{matrix} .$$

More generally, if the system starts with initial functional state L_t with probability p_t, then after k upsets the functional state probabilities are given by

$$L(k) = T^k \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix} .$$

It can be seen that knowledge of the transition matrix allows a probabilistic description to be made of the functional state traversal, given the initial state. Given the digital system, its containment set, a set of error observation points, and a class of I/T faults, the transition matrix can be measured experimentally. Possibly, methods can be developed which produce the transition matrix with minimal experimentation. Regardless of how it is actually generated, once obtained it provides a totally new dimension to I/T fault analysis.

System Validation: The transition matrix can be used to assess the I/T fault tolerance of various implementations of functional requirements for a given system. As a simple illustration of this, suppose that for a given I/T fault environment, all that is of interest is the result of a single upset in a digital system. Suppose further that in the containment set $\{L\}$ there is only one valid functional state L_0. Then the entry in the first column, first row of T gives the probability of the system remaining in L_0, given an error. Comparing this entry in T for different versions of functionally equivalent implementations provides a measure of their fault tolerance. Fig. 1 shows state transition diagrams for two possibilities for T for the simple case where there is only one invalid functional state. For the implementation corresponding to Fig. 1(a)

$$T^* = \begin{bmatrix} 3/4 & 15/16 \\ 1/4 & 1/16 \end{bmatrix}$$

and for the implementation corresponding to Fig. 1(b)

$$T^{**} = \begin{bmatrix} 7/8 & 1/8 \\ 1/8 & 7/8 \end{bmatrix} .$$

Obviously, T^{**} is the more transient fault-tolerant implementation for single upsets. Suppose, however, that the situation was changed such that a large number of upsets were likely. Now T^k for large k becomes important. Interestingly,

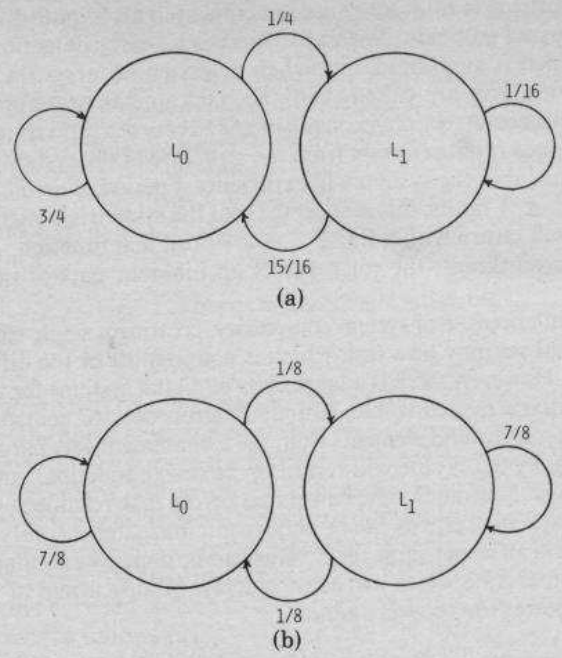

Fig. 1. Two 2-loop implementations. (a) T^*. (b) T^{**}.

$$\lim_{k\to\infty} (T^*)^k = \begin{bmatrix} 15/19 & 15/19 \\ 4/19 & 4/19 \end{bmatrix}$$

and

$$\lim_{k\to\infty} (T^{**})^k = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix} .$$

Hence, T^* is the more fault-tolerant implementation for a multiple upset environment.

This example serves to show that system comparison for fault tolerance may not supply an answer which is correct for all situations. Environmental, task, and mission time considerations must be taken into account with the use of transition matrices and the associated containment sets. This approach is a very useful analysis, evaluation, and validation tool.

III. Fault Tolerance

The approach to I/T fault analysis through the ideas of the containment set and the transition matrix suggests and makes feasible a new method for obtaining fault tolerance in digital systems. The system can be modified in the attempt to improve its behavior in the presence of faults, indicated by the transition matrix. If the system is not overly complex, there is the possibility of modifying it so that all valid functional states can be removed from the containment set. If this can be done, the transition matrix reduces to the scalar value 1, and no upset or series of upsets can possibly remove the digital system from its intended function. The one element of the containment set corresponds to proper system operation; no other functional operations are possible. All steady-state effects of I/T faults disappear.

The transient effects of I/T faults are of importance, and much theoretical research has been done into the prediction and measurement of error latency times, chiefly for use in determining state vector storage periods for duplex redundant systems. It is clear that it is impossible to eliminate or mask transient effects of I/T faults without massive redundancy. For the many applications where duplication or triplication of equipment is not desired but where some degree of fault tolerance is required, the steady-state approach of transition matrix analysis can be used very effectively. In these cases, temporary interruptions of control or data values must be acceptable, as long as proper control returns within a reasonable period of time. The steady-state fault effects are seen to be much more important than transient effects.

This approach to fault tolerance in digital systems is quite different form the usual methods. Normally, there is an error detection procedure which is monitored, and when detection occurs a set of recovery procedures are called. With the containment set approach there is neither a detection mechanism nor a recovery procedure. The fault tolerance results entirely from the structure of the system itself. It is assumed that the system will experience a period of errors when subjected to I/T faults; this is accepted, but the system is constructed so that it will return within finite time to its original function. There is no error masking—the tolerance is an inherent part of the design.

In general, because of system complexity, creating a single element containment set may be a task which is not possible or too difficult to achieve. However, there is a large class of digital systems for which this approach is indeed viable: namely, microprocessor controllers.

Removing all invalid elements from the containment set guarantees that eventually the system will return to the design function, but says nothing about how much time can elapse before task resumption. An analysis of specific systems can be done to minimize this recovery time and to obtain an upper limit for it. This can be done by examination of each system state which can cause a delay. Modifications of these states can lower the recovery times.

IV. Conclusions

The principle requirement for practical system application is finding a set of operational states which are complete, mutually exclusive, and finite, to allow use as a containment set. This allows digital system fault phenomena to be viewed through a high level upset perspective. This integrated approach to the fault/system/program complex can then yield provable, yet useful, design and validation techniques.

Upset theory has been applied to microprocessor controllers, and methods to obtain containment sets for 8085-based systems developed [9]. A testbed system was used to verify the practicality for the implementation of crash-proof controllers based on upset theory. This new method has been found to be far superior to the conventional watchdog timer method for applications where the need for fault tolerance is not critical enough to warrant an expensive approach such as TMR. The cost increase for fault tolerance is approximately the same as that of the watchdog timer method.

Microprocessor controllers are small enough that it is possible to reduce the containment set to valid states by removing all erroneous states. Larger systems most likely will not offer this luxury. The containment set would then categorize the various erroneous states which are not removable, and the fullest use of transition matrices would be made. The application of upset theory to the design of transient fault-tolerant microprocessor controllers is just one of its uses in the field of transient fault analysis.

References

[1] C. P. Disparte, "A self-checking VLSI microprocessor for electronic engine control," in *Proc. FTCS-11*, June 1981, p. 253.
[2] B. Courtois, "Some results about the efficiency of simple mechanisms for the detection of microcomputer malfunctions," in *Proc. FTCS-9*, June 1979, pp. 71–74.
[3] ——, "A methodology for on-line testing of microprocessors," in *Proc. FTCS-11*, June 1981, pp. 272–274.
[4] D. R. Ballard, "Designing fail-safe microprocessor systems," *Electronics*, vol. 52, pp. 139–143, Jan. 4, 1979.
[5] D. G. Platteter, "Transparent protection of untestable LSI microprocessors," in *Proc. FTCS-10*, 1980, pp. 345–347.
[6] J. F. Wakerly, "Microcomputer reliability improvement using triple-modular redundancy," *Proc. IEEE*, vol. 64, pp. 889–895, June 1976.
[7] A. Avizienis, "Architecture of fault-tolerant computing systems," in *Proc. FTCS-5*, June 1975, pp. 3–16.
[8] ——, "Fault-tolerance: The survival attribute of digital systems," *Proc. IEEE*, vol. 66, pp. 1109–1125, Oct. 1978.
[9] R. E. Glaser and G. M. Masson, "The containment set approach to crash-proof microprocessor controller design," in *Proc. FTCS-12*, June 1982.

The Design of a Reliable Remote Procedure Call Mechanism

S. K. Shrivastava and F. Panzieri

***Abstract*—In this correspondence we describe the design of a reliable Remote Procedure Call mechanism intended for use in local area networks. Starting from the hardware level that provides primitive facilities for data transmission, we describe how such a mechanism can be constructed. We discuss various design issues involved, including the choice of a message passing system over which the remote call mechanism is to be constructed and the treatment of various abnormal situations such as lost messages and node crashes. We also investigate what the reliability requirements of the Remote Procedure Call mechanism should be with respect to both the application programs using it and the message passing system on which it itself is based.**

***Index Terms*—Atomic actions, data communication, distributed systems, fault tolerance, local area networks.**

I. Introduction

In this correspondence we describe the design of a reliable Remote Procedure Call (RPC) mechanism which we have been investigating within the context of programming reliable distributed applications. In the following we consider a distributed system as composed of a number of interacting "client" and "server" processes running on possibly distinct nodes of the system; the interactions between a client and a server are made possible by the suitable use of the RPC mechanism. Essentially, in this scheme a client's remote call is transformed into an appropriate message to the named server who performs the requested work and sends the result back to the client and so terminating the call. The RPC mechanism is thus implemented on top of a message passing interface. Some of the interesting problems that need to be faced are: 1) the selection of appropriate semantics and reliability features of the RPC mechanism, 2) the design of an appropriate message passing interface over which the RPC is to be implemented, and 3) the treatment of abnormal situations such as node crashes. These problems and their solutions are discussed in this correspondence. We shall concentrate primarily on the relevant reliability issues involved, so other directly or indirectly related issues such as type checking, authentication, and naming will not be addressed here.

The RPC mechanism described in the following has been designed for a local area network composed of a number of PDP 11/45 and LSI 11/23 computers (nodes) interconnected by the Cambridge Ring [1]; each node runs the UNIX[1] (V7) operating system. However, most of the ideas presented in this correspondence are, we believe, sufficiently general to be applicable to any other local area network system.

Manuscript received October 2, 1981; revised January 7, 1982. This work was supported by the Science and Engineering Research Council of the United Kingdom and the Royal Signals and Radar Establishment of the United Kingdom.

The authors are with the Computing Laboratory, University of Newcastle upon Tyne, Newcastle upon Tyne, England.

[1] UNIX is a trademark of Bell Laboratories.

0018-9340/82/0700-0692$00.75