EMULATION APPLIED TO RELIABILITY ANALYSIS
OF RECONFIGURABLE, HIGHLY RELIABLE, FAULT
TOLERANT COMPUTING SYSTEMS FOR AVIONICS

GERARD E. MIGNEAULT

APRIL 1979
This paper proposes that emulation techniques can be a solution to a difficulty arising in the analysis of the reliability of highly reliable computer systems for future commercial aircraft, and thus should warrant investigation and development.

The paper first establishes the difficulty, viz., the lack of credible precision in reliability estimates obtained by analytical modeling techniques. The difficulty is shown to be an unavoidable consequence of: (1) a high reliability requirement so demanding as to make system evaluation by use testing infeasible, (2) a complex system design technique, fault tolerance, (3) system reliability dominated by errors due to flaws in the system definition, and (4) elaborate analytical modeling techniques whose precision outputs are quite sensitive to errors of approximation in their input data.

Next, the technique of emulation is described, indicating how its input is a simple description of the logical structure of a system and its output is the consequent behavior. Use of emulation technique is discussed for "pseudo-testing" systems to evaluate bounds on the parameter values needed for the analytical techniques.

Finally an illustrative example is presented, albeit for a fanciful small scale application, to demonstrate from actual use the promise of the proposed application of emulation.

INTRODUCTION

Research efforts are underway to develop more efficient civil transport aircraft for the future. One facet of the effort involves active control technology which implies greater reliance upon computer systems in order to obtain maximum benefits. This paper discusses the need and justification of development and investigation of emulation techniques as adjuncts to theoretical reliability analysis models of fault tolerant avionic computer systems.

REQUIREMENT FOR FAULT TOLERANCE

Designs of fault tolerant computer systems have arisen in response to anticipated needs of future civil aircraft (Bjurnan, B. E. et al., 1976), (Hopkins, A. L. et al., 1978), (Wensley, J. H. et al., 1978). Requirements for reliability of systems and associated components have been inferred from the expression "extremely improbable" in regulatory documentation pertaining to safety in commercial transport aircraft (FAA, 1970). The following, variously worded, informal statements indicate the range of interpretations:

"Thus we have a reliability requirement of $10^{-9}$ per hour of operation for a level 1 or level 2 function with no internal or external backup ..." * (Ratner, R. S. et al., 1973)

"... a number less than or equal to $1 \times 10^{-9}$ has been imposed ... to represent the probability of an event designated as extremely improbable. ... Loss of the CCV/FBW function, given a fault-free system at dispatch, shall be extremely improbable." ** (Bjurnan, B. E. et al., 1976)

"... the computer's failure rate will be designed below $10^{-9}$ failures per hour in flights of up to ten hours duration, with a preferred goal of $10^{-10}$ failures per hour." (Smith, T. B. et al., 1978)

"... the extrapolated failure of the design in context with production system application shall not exceed $10^{-9}$ computer-related system failures in flights up to ten hours." (sic) (NASA, 1978)

As an average of the interpretations, and for discussion purposes, an equally informal statement is adopted here as the requirement, viz.,

the probability that a system containing no failed components at the start of operation will fail during the first ten hours of operation will be less than approximately $10^{-9}$

in which the term "failed components" refers, in a conventional manner, to failures caused by physical defects occurring randomly in time, and in which a system is considered to have failed when it has not correctly performed the function required of it as a subsystem in a larger, encompassing system.

Temporarily disregarding failures due to causes external to systems or to inadequately or incorrectly designed and implemented systems, one can determine that, in order to satisfy the reliability requirement, a computer system constructed of devices (in turn constructed of more basic components) with independent

*Levels pertain to criticality of functions.

**CCV/FBW = control configured vehicle / fly by wire.
failure distributions and constant failure rates would require, if it were intolerant of the failure of any of its constituent devices, a mean time to failure (MTTF) of approximately ten billion (10^{10}) hours for the least reliable of the devices. Such a system is unlikely to see the light of day in the near future, to say the least, since realistic, available devices such as processors, memories, etc., from which such a system can be constructed, do not have such lengthy MTTF’s; values in the range from 10^{2} to 10^{5} are more reasonable. Consequently, computer systems intended to satisfy the reliability requirement have been designed to tolerate failures.

A CONSEQUENCE OF FAULT TOLERANCE

Several characteristics of fault tolerance give rise to a need to examine explicitly the reliability implications of a failure mode conventionally handled implicitly by testing actual systems.

One rather obvious characteristic of a fault tolerant system is redundancy of components -- at the very least when in an initial condition free of failed components. In the case of systems with requirements for reliability stated in terms of the first few hours or a small fraction of expected equipment lifetimes, the characteristic implies renewal activities which will be often repeated. While some form of verification that systems are still in a (perceived) fault-free condition will be a minimum renewal activity, the MTTF’s of realistic, avionic devices insure that a not insignificant amount of repair activity will also be needed -- to return systems to the fault-free, initial condition needed to fulfill the assumptions underlying the reliability estimates. The characteristic further suggests, other things being equal, that the more “multifunction” the constituent devices are, the more efficient the systems are in terms of total equipment used and maintained. Therefore, there is an economic pressure for designs utilizing multifunction devices such as microprocessors with software. However, a cost is incurred in a greater complexity in the synthesis, logic and analysis of systems with parallel and/or intersecting signal and data paths and time-shared use of resources and algorithms.

Another necessary characteristic of a fault tolerant system is its possession of an agent or mechanism capable of detecting failures in devices or components and utilizing available redundancy to nullify failures. This characteristic may be achieved in a passive manner when some convenient property of nature permits -- a simple example is parallel rather than series wiring of Christmas tree lamps to avoid an open circuit failure caused by one defective lamp -- or, as appears more likely to be necessary in complex systems, in an active manner by the addition to a redundant system of still more devices and/or logic to act as detectors and nullifiers. Of course, a price is paid again in increased complexity.

There is a notion which merits a few words as it occasionally arises at this point. The notion is that the reliability requirement is unnecessarily stringent, as witnessed by the ten billion (10^{10}) hour MTTF previously cited. However, that value was for a fault intolerant system, a “series” system, and is inappropriately an approximation of the MTTF of a fault tolerant system of equivalent reliability at an extremely early stage of its expected operation, i.e., ten hours, for one reason because the variance of Figure 5 compares the failure density distribution of two systems having the same mean (i.e., same MTTF).

Density A is a series system. Density B is a representation (specifically a 2 out of 5) of a parallel redundant system. Clearly, at an early stage in their operation, the parallel system has a greater reliability. An upper approximation is provided by the MTTF of systems composed of several r-out-of-n subsystems in series. A system consisting of a single r-out-of-n subsystem serves as a reasonable upper-bound estimate of the MTTF of a fault tolerant system when the representative constituent device chosen is the fault tolerant system’s “worst” (i.e., the device type with the greatest MTTF in the set of constituent devices that would be in this set). Assuming, as before, that constituent devices have independent failure rates, one can show that an r-out-of-n system has a MTTF not very much different from that of its constituent devices, and quite likely less because of factors accounted for by “coverage”, possible configurations having a stated number of operating constituent devices. The transition rate out of coverage is known as “failure” parameters C_i, conditional probabilities of successful transition, are included. Successful transition is assumed to mean immediate system failure. Usually the coverage parameters are associated with systems having active recovery processes, but they are also applicable to passive mechanisms as long as there are transitions which can go away among distinct Markov process, one can develop the appropriate differential equations for the stochastic process represented by the expression

\[ 1 - e^{\lambda t} \sum_{j=1}^{n} \frac{\lambda^j}{j!} \left( e^{\lambda t} - 1 \right)^j \]

where \( a_0 = 1 \) and \( a_j = \prod_{i=1}^{j} C_i \) for \( j = 1, 2, \ldots, (n-r) \).

Ratios of system MTTF to constituent device MTTF are tabulated for various combinations of values of \( r, n, \) and \( C_i \) in Tables 1 and 2. In Table 1, \( C_i = 1 \) for all \( i \), implying that coverage is perfect. Although the ratios are independent of the constituent device’s failure rate (or equivalently, MTTF), not all combinations of \( r \) and \( n \) are useful, given a specific device failure rate. Therefore, a device with MTTF less than 10^{9} hours could be used to construct systems of zones \( r = 3 \) and \( n \) lower but not zones \( r = 4 \) or higher. More specifically, a device with MTTF of five thousand (5*10^{3}) hours would not be used to construct systems of zones 4 and 5. In Table 2, \( C_i = 0.9 \) and \( C_i = 0.1 \).
An additional assumption is made -- that the software and hardware are independent -- to keep the confidence (in the might not better be modeled as error "bursts"). Figure 2(b) is a simpler representation of the same difficulty, when lifetime-use testing of actual systems is not feasible. Of establishing with acceptable reliability goal. In Figure 3, the two models are combined to represent simply a system subject to and since there has been no credible empirical evidence available for the selection and justification of any failure rate, one might expect to not see a second, identical software error, assuming the initial error to have been triggered by unusual test and selection, in accordance with stated acceptance criteria, from among alternate algorithms in a parallel processors are conceivable mechanizations. The "logical redundancy" techniques are therefore seen to analogous to "hardware redundancy", techniques for designing systems with "logical redundancy" to attempt to prevent system failures attributable to residual definitional flaws are becoming a subject of research -- and development. The software fault tolerance studies at the University of Newcastle-upon-Tyne are a leading example of recent innovations (Randell, B., 1975). Analogously to "hardware redundancy", techniques for designing systems with "logical redundancy" to attempt to prevent system failures attributable to residual definitional flaws are becoming a subject of research -- and development. The software fault tolerance studies at the University of Newcastle-upon-Tyne are a leading example of recent innovations (Randell, B., 1975). Parallel and concurrent alternate software algorithms in parallel processors are conceivable mechanizations. The "logical redundancy" techniques are therefore seen to parallel hardware. Fault tolerant software lends itself to an especially simple behavior model, as in Figure 2(a), on the assumption that successful recovery from a software (or logic) failure implies immediate return to the initial (software) state. The rationale for the assumption is that the flaw responsible for the software error has always been present in the system, having merely not been previously activated, so to speak; the system remains ready to function as before (i.e., correctly) once it has survived the software data error. Indeed, one might expect to not see a second, identical software error, assuming the initial error to have been triggered by unusual data not likely to soon be seen again. (As an aside, experiments using the emulation technique to be discussed suggest themselves to determine whether or not software data errors might not better be modeled as error "bursts"). Figure 2(b) is a simpler representation of the same recovery/failure process. Again, for the sake of simplicity, software is assumed to have a constant failure rate, \( \lambda \), and fault tolerant software is assumed to have an aggregate recovery parameter, \( \kappa \), analogous to the coverage parameters of the r-out-of-n hardware model. Immediate system failure is assumed to the result of lack of successful recovery. No further elaboration of a software model is attempted since there has been no credible empirical evidence available for the selection and justification of any particular, more complex, general model of system failure due to software (Thibodeau, R., 1978), let alone the more general case of residual definitional flaws.

TECHNIQUES FOR ADDRESSING DEFINITIONAL FLAWS

Analogously to "hardware redundancy", techniques for designing systems with "logical redundancy" to prevent system failures attributable to residual definitional flaws are becoming a subject of research -- and development. The software fault tolerance studies at the University of Newcastle-upon-Tyne are a leading example of recent innovations (Randell, B., 1975). Parallel and concurrent alternate software algorithms in parallel processors are conceivable mechanizations. The "logical redundancy" techniques are therefore seen to parallel hardware. Fault tolerant software lends itself to an especially simple behavior model, as in Figure 2(a), on the assumption that successful recovery from a software (or logic) failure implies immediate return to the initial (software) state. The rationale for the assumption is that the flaw responsible for the software error has always been present in the system, having merely not been previously activated, so to speak; the system remains ready to function as before (i.e., correctly) once it has survived the software data error. Indeed, one might expect to not see a second, identical software error, assuming the initial error to have been triggered by unusual data not likely to soon be seen again. (As an aside, experiments using the emulation technique to be discussed suggest themselves to determine whether or not software data errors might not better be modeled as error "bursts"). Figure 2(b) is a simpler representation of the same recovery/failure process. Again, for the sake of simplicity, software is assumed to have a constant failure rate, \( \lambda \), and fault tolerant software is assumed to have an aggregate recovery parameter, \( \kappa \), analogous to the coverage parameters of the r-out-of-n hardware model. Immediate system failure is assumed to the result of lack of successful recovery. No further elaboration of a software model is attempted since there has been no credible empirical evidence available for the selection and justification of any particular, more complex, general model of system failure due to software (Thibodeau, R., 1978), let alone the more general case of residual definitional flaws.

ANALYTIC RELIABILITY ANALYSIS: HOW CREDIBLE?

The software model of Figure 2 and the r-out-of-n model of Figure 1 suffice, however, to show the difficulty, when lifetime-use testing of actual systems is not feasible, of establishing with acceptable confidence (in the statistical sense) that systems designed to satisfy the \( 10^{-9} \) requirement do achieve the reliability goal. In Figure 3, the two models are combined to represent, simultaneously and tolerant of both hardware component failures and errors due to residual definitional flaws (here, software). An additional assumption is made -- that the software and hardware are independent -- to keep the
Illustration simple again. It is possible to add more complexity in the model, but as stated before, there is no empirical evidence to justify selecting any particular model in preference to another. Also, the conclusion below is not appreciably modified. Again recognizing the model and assumptions as a Markov process, the probability of system failure is computed to be

\[ 1 - e^{-(\lambda(1-k)\tau)} \sum_{j=0}^{n-1} a_j (\lambda t + 1)^j \]

where \( a_0 \) and \( a_j \) are as before.

For a typical (and optimistic) value for \( \lambda \) (10^4 failures per hour), typical values for \( n \) (5 to 10) and the required value for \( \tau \) (10 hours), bounds on \( C_1 \), \( C_2 \) and \( u(1-k) \) required, in order for the system to satisfy the 10^-9 requirement, are calculated to be as follows:

\[ 1 \geq C_1 \geq 0.999999 \]
\[ 1 \geq C_2 \geq 0.9999 \]
\[ u(1-k) \leq 10^{-10} \]

There appears to be little margin for error in designing systems to satisfy the 10^-9 requirement. Refinement of the model cannot eliminate the difficulty in estimating precisely the reliability of such systems; it can only transform it into a need for near perfect knowledge of different parameters, for the systems must still achieve the same aggregate behavior as above.

**MORE COMPLEX MODELS**

In the process of investigating fault tolerant systems (previously, principally studies of hardware) numerous models have been developed for analyzing the reliability of such systems. Of late, investigations have also been undertaken into models to relate the system failure modes to time-variable computational and performance requirements, thus attaching the reliability of a system more tightly to its application (Meyer, J., 1977), (Beaudry, H. D., 1978). Some model evaluation schemes have been "computerized" to serve as more or less general purpose tools for the convenient analysis, in the architectural design stage, of systems composed of complex arrangements of elements, e.g., CAST (Conn. R. B. et al., 1974), CARE II (Stiffler, J., 1974), CARSSA (Bjurman, B. E. et al., 1976). Although they consider details of system behavior such as recovery (detection, isolation, reconfiguration) strategies, sparing (active, stand-by, switching) strategies, transient and intermittent fault (duration, periodicity, leakage) modes, functional dependence among devices, nonexponential failure distributions, etc., the models still are constructed essentially from parametric descriptions of aggregate system, subsystem and/or device behavior in order to make use of mathematical techniques applicable to idealized stochastic process models for reasonably efficient computation. Hence all the models must be provided with parameter values which need to be assumed or known, by some other means, in order to precisely represent any and each particular system design of interest.

**EMULATION**

**Digital Simulation**

While the word "simulation" is widely used to denote all manner of techniques for, among other purposes, analyzing the behavior of objects and their environments by means of implementation and manipulation of more conveniently malleable surrogates, here the word is limited to mean the use of computer "systems" as surrogates -- at whatever level of abstraction is meaningful to an application. The concept of system is stressed because usefulness of a simulation scheme depends upon both software and hardware -- a characteristic more effectively utilized by emulation. For example, consider the reliability analysis program previously mentioned -- CAST, etc. Although they are essentially simulation schemes which are normally discussed without regard to host computer hardware, in any actual application, host computer hardware will be an important constraint upon the amount of detail which it will be feasible to consider with the programs.

Digital simulation, as opposed to emulation, at the level of gate logic has been discussed in the literature on computers and considered as a tool for design and fault (signature) analyses of digital logic circuits at levels of detail ranging from simple (e.g., assuming gates to have only two possible output values) to complex (e.g., allowing undefined values of gate outputs and various timing anomalies) (Szygenda, S. and Thompson, E., 1976). For the analysis of circuits the sizes of microprocessors, memories and larger, in practice simulation techniques at the aggregate, functional behavior level begin to displace gate level simulations (Monon, P. and Chappell, S., 1977) as the gate level simulation costs become prohibitive when compared to perceived benefits.

However, for the purposes of reliability analysis of fault tolerant systems, gate level simulation warrants considerable cost in view of the conclusion to be drawn from the preceding paragraphs that, at the levels of reliability of interest, the probability of failure of such systems is less dependent upon the mode of failure resulting from depletion of redundant resources than it is upon the less well understood and questionably modeled modes considered under the terms "coverage" and "definitional flaws". A similar conclusion to the effect "that the introduction of a redundancy at the hardware level increases the relative influence of software faults" is made elsewhere (Costes, A., 1978). Unfortunately, while the costs
could be suffered, in light of the benefits, gate level simulation is not a feasible technique for application to questions involving chance events and repeated trials because it is time consuming -- orders of magnitude slower than likely target systems.

Emulation vs. Simulation

In ordinary use, the word "emulation" means an endeavor to equal or excel; in the present context, it is reserved for a particular technique of implementation as possible when a host computer is microprogrammable. In order to avoid confusion, "simulation" acquires the added meaning here of being distinct from "emulation". Microprogramming is significant because it allows a final definition of a computer's "apparent" instruction set to be postponed until after the definition of hardwired logic is completed, and it does this with an acceptably small risk that the hardwired logic will need redesign. This happens because a "real" instruction set is defined by the hardwired logic, is at a quite primitive level, and is tailored especially for executing algorithms which, in turn, become operational definitions of less primitive operations -- the "apparent" instruction set.

Thus it may be said that a computer defined by an "apparent" instruction set does not really exist; it is "emulated" by microprogrammable hardware by means of microcoded algorithms. Admittedly, variations in efficiency of variant microcode operations vis-à-vis various "apparent" instruction sets may exist, but they can be ignored for the present purpose. What is notable is that, given reasonable care not to mismatch host and target computers, microprogrammable computers can perform in the role of an "apparent" computer approximately as efficiently as a hardwired version of the "apparent" computer would. Note that "emulation" is at a level of detail which permits software implemented for another, "apparent", target computer to be executed "directly" by a host computer. That is, no modification of the target software is needed to make it compatible with the host computer, and no special software on the host computer needs to be generated (more specifically, no simulation program in an "apparent" instruction set on the host to interpret the instructions of the target software and mimic the target computer) as would be needed on a nonmicroprogrammable computer.

Use as a Diagnostic Tool

Addition of diagnostic, control functions in the microcode permits a host computer to act not only as a surrogate but also as a device for observing and recording (and possibly analyzing) target software performance in an ostensibly natural environment. Such "diagnostic emulation" use is becoming more common in the development and maintenance of special software systems and is, seemingly, "emulation" in the dictionary sense. As might be expected efficient use of such a diagnostic system requires support capabilities for readily modifying microcoded algorithms defining target computers. Such facilities are beginning to be developed -- for example, EMULAB (Clausen, B. et al, 1977). What has been less well considered is the fact that such capabilities can be extended to permit analysis not only of software but also of systems (i.e., software and hardware) -- and not only as they are intended to be but also as they are not. By generating the defining microcode such that it represents target computers in sufficiently fine detail combinations of failures in individual components, anomalous data, and definitional flaws can be introduced and their effects at the system level observed rather than assumed. Thus emulation provides a conveniently manipulated failure effects analysis tool. In addition the manner in which an emulation technique is implemented, with automated diagnostic and system and environment controls, lends itself to use for "pseudo-testing" as in Figure 4.

In general, emulation can be used to generate repeated trials of "emulated" systems from which failure ratios and histograms can be tabulated for analysis -- hence, aggregate behavior models are verified and parameter values estimated with some measure of confidence (in a statistical sense). Clearly, assumptions about the manners and rates of occurrence of failures and flaws must still be made in order to introduce these last into the emulations. However, while the credibility of precise assumptions will still be questionable, it should be possible to develop credibly pessimistic assumptions to attempt to demonstrate that particular fault tolerant system designs exceed the reliability requirement.

While, also, in general, the use of emulation to perform such "pseudo-testing" is limited by the efficiency (i.e., computation speed) of the emulation technique and equipment, it appears reasonable to state that it is less restricted than in the case of digital simulation. Given the previously described need and difficulty of establishing the reliability of the fault tolerant avionic computer systems of interest, emulation techniques merit further investigation.

SAMPLE EXPERIMENT

Scope

An effort of limited scale was undertaken in order to determine whether or not an emulation scheme could be devised which would be sufficiently efficient to support systems of meaningful sizes and complexities, and to demonstrate that such a scheme could be implemented in a manner convenient for analysis purposes by users not well versed, if at all, in the emulation scheme itself. As a demonstration, a sample analysis bearing upon reliability of fault tolerant systems was chosen.

The effort was experimental; time and effort were expended searching out efficient implementations and superior microprogramming capabilities to support the implementations. Consequently no commitment to any specific microprogrammable hardware was desirable initially. The experiment was performed on a large, general purpose computer whose underlying microcode was sacrosanct. For this reason emulation was really simulated. This last level of complication can be accounted for by introducing a time scale factor; it is otherwise ignored here. While some variant emulation algorithms which have been conceived have not yet been implemented and examined, the effort has provided a basis for selecting microprogrammable hardware.
for further studies. Here, however, the experiment is discussed merely to illustrate an actual, rather than speculative, application of emulation to reliability analysis.

**Emulation Technique**

The scheme selected consists of an algorithm generated independently of any target computer. Descriptions of particular systems to be emulated are provided to the algorithm at the time of operation. The method is referred to as "table-driven" in contrast to a "compilation" method in which a hardware description is input to a hardware description language "compiler" which generates a computer program to emulate one specifically defined computer. The table-driven method was chosen because it was believed to facilitate the infusion of failures and to provide better visibility to a user. That is, the target hardware is visible as a distinct entity at emulation time rather than being dispersed and buried inside the workings of an emulation program, and failures and faults can be added and removed without altering the cyclic nature of the algorithm.

From a user's viewpoint, the emulation is visualized as the repeated transformations of two variables. One variable, \( S_n \), describes the structure of the system at time step \( n \). The variable is essentially a matrix which identifies the interconnections among the logic elements in a system, and also identifies the functional behavior of each element. The most primitive element permitted is a generalized gate to which constant behavior characteristics (neither correct nor faulty to the emulation algorithm) are attached. More complex elements such as flip-flops and tristate devices are also permitted, if desired, as primitive elements to be manipulated as indivisible entities by the emulation algorithm. For example, a logic element \( X \) might have been identified to act as a four (4) input NAND gate driving six (6) other identified elements and supposed to have an irregular input-to-output signal propagation time. Hence, \( S_n \) is effectively a time-varying, annotated logic diagram.

A second variable, \( V_n \), is a vector containing the output values, at time step \( n \), of each of the logic elements defined in \( S_n \). Target software corresponds to a subset of this variable, viz., those values corresponding to logic elements defining some of the emulated system's memory.

A third auxiliary variable, \( F_n \), can be visualized as a source of external perturbations into the emulated system — affecting \( S_n \), \( V_n \), or both. As currently implemented, this variable is generated separately from the others in order to increase the speed of the emulation computations. It represents the source of random failures, flaws, and anomalies at either preselect or random times and control over the emulation process.

The emulation algorithm, a time invariant transformation, is a collection of techniques (so-called "selective trace", linked lists, data compression, parallel processing — untested because of the limitations of the general computers previously mentioned —, event scheduling) consistent with a model of the behavior of a "generalized" logic element over an arbitrary time step.

**SAMPLE ANALYSIS: LATENT FAILURES**

The experimental analysis performed was a study of the efficacy of five (5) particular algorithms, each with a different instruction mix, as detectors of component "stuck-at" faults (i.e., latent failures) in a particular "play" system. The analysis is documented in detail in (Nagel, P., 1978).

The "play" target computer was originally generated (i.e., defined at the gate logic level) as a vehicle for checking out the initial and modified versions of the emulation algorithms, and for demonstrating the ability of support software, a hardware description language translator and meta-assembler, for regenerating target software, to respond semiautomatically to hardware design changes. The "play" computer has a memory of 8192, 16 bit wide words, a CPU with a count of approximately 2000 gate equivalents, and a single input-output register/port. The logic is arbitrarily assigned to four (4) hypothetical chips: a "clock" chip, an "adder" chip, an "op-decode" chip, and a miscellaneous odds and ends chip. The instruction set contains about a dozen basic instructions.

The emulated system trails were simple. The five algorithms, ranging in length from about a dozen instructions to several hundreds, were repeatedly executed, with randomly selected initial data, and randomly selected faults of random components. Distributions of time from fault occurrence to fault detection (i.e., fault latency duration) were generated. Two analyses of the sort that would be of interest in studies of fault tolerant systems were made. For one, the observed distributions were fitted against commonly used mathematical models, e.g., exponentials, as would be done in order to determine models and parameter values for use in reliability analysis programs. The results, of course, are not significant, owing to the fanciful nature of the input data; still, it is interesting that the distributions were best fit by models of balls selected at random from urns. Another result, that the distributions each exhibited different nonzero probabilities of never detecting the faults, was predictable, but only an experiment of this nature could determine the differences in magnitude. A second effort was a search for correlations among the distinguishable characteristics of the algorithms and the distributions. The only significant correlation found was between instruction mix and detection probability. Here too, because of the nature of the target system, the magnitudes of the correlations can only be considered fanciful. But the concept is useful in considering characteristics which should be avoided in algorithms whose function is to reconfigure a system after a failure has been detected.

**CONCLUSION**

A case has been made for the use of emulation techniques as a needed adjunct to reliability analysis models for highly reliable avionic computer systems. Although no conclusion about the technique's
eventual usefulness is yet warranted, in light of its apparent usefulness as a failure-modes effects analysis tool and the promise and potential rewards of its use for probability distribution uses, further development and investigation of the technique appears warranted and is being pursued by the NASA.

REFERENCES


FAA FAR part 33, paragraph 25.1309(b), dated 5 August 1979.


### TABLE 1

<table>
<thead>
<tr>
<th>r</th>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>.83</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.08</td>
<td>.58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.28</td>
<td>.78</td>
<td>.45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1.45</td>
<td>.95</td>
<td>.62</td>
<td>.37</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1.59</td>
<td>1.09</td>
<td>.76</td>
<td>.51</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.72</td>
<td>1.22</td>
<td>.89</td>
<td>.64</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>1.00</td>
<td>.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>.85</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ratio of MTTF (r/n system, \( C_i = 1 \)) to MTTF (constituent device).

### TABLE 2

<table>
<thead>
<tr>
<th>r</th>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.6</td>
<td>.55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.46</td>
<td>.45</td>
<td>.43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.38</td>
<td>.37</td>
<td>.37</td>
<td>.35</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.31</td>
<td>.31</td>
<td>.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>.27</td>
<td>.27</td>
<td>.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>.21</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ratio of MTTF (r/n system, \( C_1 = 0.9, C_{i\neq 1} = 0.1 \)) to MTTF (constituent device).

3-6-8
Figure 1 — r-out-of-n system with coverage parameters

Figure 2(a) — Fault Tolerant Software — Figure 2(b)

Figure 3 — r-out-of-n system with fault tolerant software
FIGURE 4. EMULATION "PSEUDO-TESTING" ACTIVITY

- Possible design mods to correct unbearable flaws

- Requirement mods

- System description

- Test designs

- H/W Logic

- Algorithms

- Compiler

- Assembler

- Translator

- Fault generator

- Input generator

- H/W Logic

- Object code

- Emulator

- Diagnostic observer

- Continuous reporting

- Output

Diagnostics
FIGURE 5. - FAILURE DISTRIBUTIONS WITH SAME MTTF.
This paper proposes that emulation techniques can be a solution to a difficulty arising in the analysis of the reliability of highly reliable computer systems for future commercial aircraft, and thus should warrant investigation and development.

The paper first establishes the difficulty, viz., the lack of credible precision in reliability estimates obtained by analytical modeling techniques. The difficulty is shown to be an unavoidable consequence of: (1) a high reliability requirement so demanding as to make system evaluation by use testing infeasible, (2) a complex system design technique, fault tolerance, (3) system reliability dominated by errors due to flaws in the system definition, and (4) elaborate analytical modeling techniques whose precision outputs are quite sensitive to errors of approximation in their input data.

Next, the technique of emulation is described, indicating how its input is a simple description of the logical structure of a system and its output is the consequent behavior. Use of emulation techniques is discussed for "pseudo-testing" systems to evaluate bounds on the parameter values needed for the analytical techniques.
End of Document