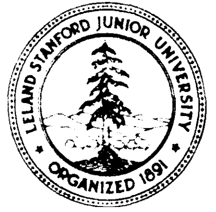
DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES
DEPARTMENT OF ELECTRICAL ENGINEERING
STANFORD UNIVERSITY · STANFORD, CA 94305



SEL-77-039

A SIMULATOR FOR THE EVALUATION OF DIGITAL SYSTEM RELIABILITY

Peter Alan Thompson

Technical Report No. 119

August 1977

This work was supported in part by
NASA Grant NGR-05-020-699, Sup. 1,
National Science Foundation Grant NSF MCS 76-05327, and
Air Force Office of Scientific Research Grant 77-3325.

Acknowledgment: Acknowledgment is made to the
NASA Ames Research Center, Moffett Field, California
for the use of their CDC-7600 computing facility.

SEL-77-039

A SIMULATOR FOR THE EVALUATION OF DIGITAL SYSTEM RELIABILITY

Peter Alan Thompson

Technical Report No. 119

August 1977

DIGITAL SYSTEMS LABORATORY
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California

This work was supported in part by NASA Grant NGR-05-020-699, Sup. 1, National Science Foundation Grant NSF MCS 76-05327, and Air Force Office of Scientific Research Grant 77-3325.

Acknowledgment: Acknowledgment is made to the NASA Ames Research Center, Moffett Field, California for the use of their CDC-7600 computing facility.

A SIMULATOR FOR THE EVALUATION OF DIGITAL SYSTEM RELIABILITY

Peter Alan Thompson

Technical Report No. 119

August 1977

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California

ABSTRACT

This report describes a simulation package designed to evaluate the reliability of digital systems. The simulator can be used to model many different types of systems, at varying levels of detail. The user is given much freedom to use the elements of the model in the way best suited to simulating the operation of a system in the presence of faults. The simulation package then generates random faults in the model, and uses a Monte Carlo analysis to obtain curves of reliability. Three examples are given of simulations of digital systems which have redundancy. The difference between this type of simulation and other simulation techniques is discussed.

INDEX TERMS: Simulation, Reliability, Computer, Digital, Faults, Monte Carlo, Event.

CONTENTS

		Page
	Abstract	i
	Contents	ii
	Illustrations	iii
Section		**Page**
1.	INTRODUCTION	1
2.	SPECIFICATIONS FOR MODEL BUILDING	4
2.1	Model Partitions	4
2.2	Links	6
2.3	Faults	8
2.4	Simulated Units	13
3.	THE SIMULATION PROCESS	20
4.	THREEEXAMPLES	28
4.1	Example a - A General Dual Computer System	29
4.2	Example b -A **TMR Network**	35
4.3	Example c - A Dual Computer System for Navigation	38
5.	COMPARISON WITH OTHER SIMULATORS	49
6.	CONCLUSION AND FUTURE WORK	54
7.	APPENDIX A - SIMULATION EXAMPLES	57
8.	APPENDIX B - RELIABILITY ANALYSIS	70
9.	REFERENCES	73

ILLUSTRATIONS

Figure		Page
1.	Model of a Dual Computer System with Arbiter	5
2.	Definition of a Computer Unit (TYPE 1)	9
3.	Definition of an Arbiter Unit (TYPE 2)	10
4.	Simulation of One Mission	21
5.	One Monte Carlo Simulation Run	26
6.	A Dual Computer System	30
7.	Reliability Curves for Simple Dual Computer System . .	32
8.	Reliability Curves for Complex Dual Computer System .	34
9.	Triple Modular Redundant (TMR) NAND-Gate System . . .	36
10.	Reliability Curves for Simple TMR NAND-Gate System . .	39
11.	Reliability Curves for Complex TMR NAND-Gate System .	40
12	A Dual Computer System for Navigation	42
13.	Unreliability Curve Using Honeywell-316 Computers . .	45
14.	Unreliability Curve Using MSI Computers	46
15.	Unreliability Curves Using Different Repair Rates . .	47
A1.	Convention of Labelling Model Diagrams	59
A2.	Model of a Dual Computer System with Arbiter	60
A3.	Definition of Computer Unit (TYPE 1)	61
A4.	Definition of an Arbiter Unit (TYPE 2)	62
A5.	Definition of a Monitor Unit (TYPE 3)	63
A6.	Data-Storage Subroutine	64
A7.	Formatted Data-Deck for Complex Dual Computer Example.	65
A8.	Model Parameters	66
A9.	Sample Simulation Output	67
A10.	Model for Study of a TMR NAND-Gate Circuit	68
A11.	Model of a Dual Computer System for Navigation	69

1. INTRODUCTION

Any digital system is subject to physical failures, either in the electronic components themselves or in the connections between components. Such faults can cause parts of the system to act in an incorrect manner, and the ability of the entire system to perform its required function is impaired. Redundancy techniques have been devised to allow digital systems to tolerate some failures. However, the evaluation of the efficiency is a difficult problem. For such systems, accurate determination of their reliability function, that is to say, the probability of correct operation as a function of the length of the mission, is difficult to obtain by analytic methods alone. The intricacies of the mechanisms for fault detection and repair along with the diversity of the failure modes and their effects on the overall system do not lend themselves to analytical modeling. Simulation techniques are far better suited to this task. This report will describe a simulator written especially for the evaluation, with respect to reliability, of redundant digital systems.

The use of simulation can be a very powerful technique to evaluate the reliability of computer systems. Analytical reliability modeling usually requires numerous simplifying assumptions ta be made in order to make the mathematics tractable. Simulation allows many of these assumptions to be removed, resulting in a more accurate model of the system behavior. The effects of random faults on the system can be studied more easily and a variety of statistical measures

can be computed from the results of simulator runs, to characterize the system reliability.

A very general purpose simulator has been developed at the Center for Reliable Computing (CRC) at Stanford University. The simulator is general enough to allow any type of hardware configuration to be studied. The actual system configuration is specified by the user, as is the level of detail at which simulation should be performed. The user can select a portion of the system, simulate it in great detail, and then incorporate the results in a higher level simulation of the total system, where the subsystem studied in detail is represented as a single component. In this way, the computer time needed for the total simulation can be reduced.

The simulator consists of two basic parts: the utility package which handles all the details and bookkeeping required in any simulation, and the user-supplied functions which define the model for the system to be simulated. The utility package is the same regardless of the system being simulated, and is written in standard ANSI FORTRAN IV (not extended) so that it can be used on any computer supporting high-level languages. In order to simulate a system, the user must supply to the utility package a description of the system's structure and behavior. The level of detail at which the system is described is completely arbitrary. For each basic component of the system as partitioned by the user, it is necessary to supply to the utility programs a FORTRAN subroutine which describes the behavior of that component in terms of the relation between inputs, outputs, and

the set of faults that the element can suffer. Thus, one component of the model could be anything from a gate to a computer. Then the user specifies how the components are interconnected and what the fault distribution and simulation parameters are, so that the utility package can construct a model of the complete system composed of the predefined components. When the entire model is built, the utility package simulates the model's behavior in response to random faults in such a way as to obtain reliability and performance data for the system. The following sections describe first how a user can use the simulator and secondly the method used by the utility programs to simulate a system for reliability analysis.

A separate report [Thompson, 1977B] supplies all the details necessary to define a model for the simulator. It also discusses various aspects of the simulation process more explicitly than is done here, and includes a complete listing of the FORTRAN source programs of the utility package. Another report [Thompson, 1977A] describes in detail how a complex dual redundant computer system (the same as Example c - section 4.3 of this report) was studied with the simulator.

2. SPECIFICATIONS FOR MODEL BUILDING

2.1 Model Partitions

The system to be simulated is partitioned by the user into subsystems, called units. The way to partition is left completely to the user's choice. For a digital circuit, one unit may represent a single gate or register, while for a multicomputer system, one unit would be a whole computer or bus switch. All the units in the model are further grouped into different types, such that all units of the same type would be considered physically interchangeable in the real system. For example, all units representing a NAND gate can be grouped into one type, but units representing NOR gates must be grouped as a different type because they are not functionally identical to NAND gates. The reason for this grouping is that the user will supply to the simulator a description of the behavior, not of each individual unit, but rather of each type of unit. The same behavior description will be used during simulation for each unit of the same type.

This is illustrated in Figure 1, which gives one of the many possible models for a dual computer system with an arbiter. It has four units (two computers, an arbiter, and a monitor) but only three unit types (computer, arbiter, and monitor). This system is described in greater detail in the first example of Section 4, and a complete definition of the model is included in Appendix A. The units representing the computers and the arbiter will be used to explain the way in which basic elements such as links and faults can

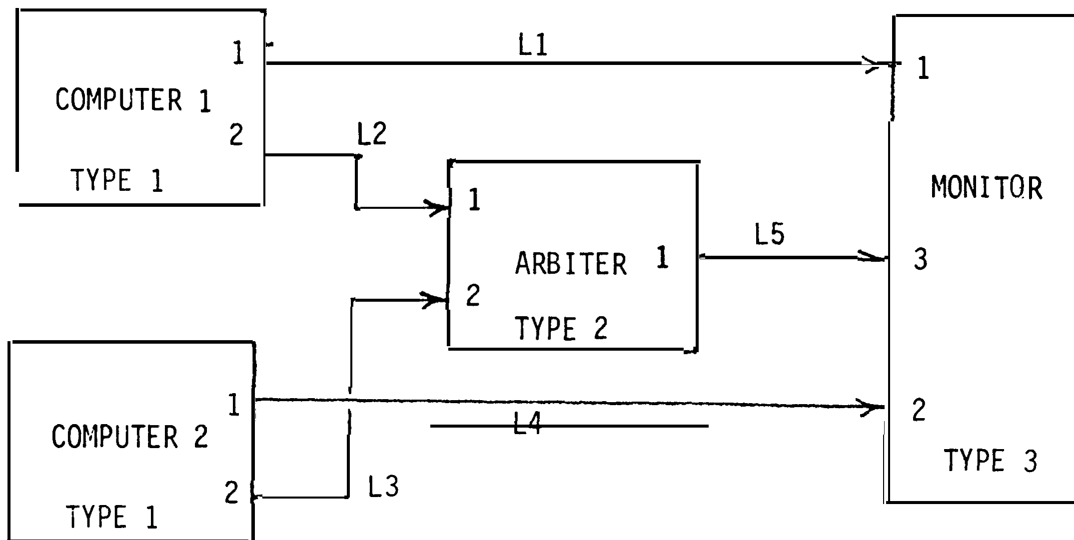


Figure 1. Model of a Dual Computer System with Arbiter.

be used to construct a model. Each computer has two types of outputs, one for computed data and the other to indicate whether or not a physical fault has been detected in that computer. The function of the arbiter is to look at the detection signals from both computers and form an opinion as to which computer has correct data outputs. The selection is characterized by a number (i.e. 1 or 2, the number of the "good" computer unit) and sent to the monitor (which is a unit that does not have any hardware equivalent. It exists only for simulation purposes). For example, suppose the error detection input from computer 1 indicates a fault in computer 1 and the error detection input from computer 2 indicates a no-fault condition in computer 2. Then the arbiter will decide that computer 2 is the correct choice, and will send the number "2" to the monitor.

2.2 Links

In the simulation models, all communication between units is done by links. Each link carries one or several numerical values. The set of these values will be referred to as the vector associated with the link. The model of Figure 1 has five links, labeled L1 through L5, each representing a vector of length one. When defining the model, the user specifies the vector length of each link, and assigns links to the input and output ports of each unit. In general, a link may have any number of sources and destinations. Each unit may have any number of input and output links. As shown in Figure 1, the arbiter has two input links and one output link. Links L2 and L4 are connected to input ports 1 and 2, respectively, and link L5 is connected to output port 1.

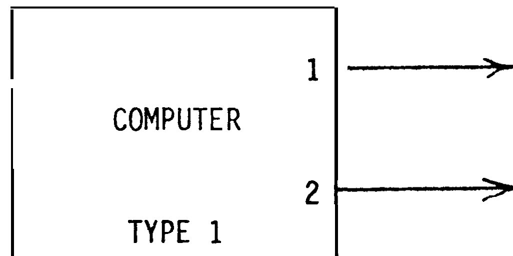
The numbers carried on a link do not necessarily correspond to actual signals in the real system. For example, the arbiter output, carried on link L5, is either the number 1 or the number 2 (referring to the decision of the arbiter). The value does not relate to the method used in the actual hardware to encode and transmit this information. The same is true of all the other links in the model. Links L2 and L4 could indicate the detected/not-detected conditions with any two unequal numbers, as long as the computer units and the arbiter unit agree on what each number means. This is even more clearly illustrated for links L1 and L3, which carry the computed data output from the computer units to the monitor. The monitor looks for a system failure by testing if the arbiter has selected a computer which has incorrect data output. Since the monitor must only know whether the selected data is correct or incorrect, there is no reason for the computer units to actually produce the exact binary or analog signals which would come from the real physical computers. During the simulation process, links L1 and L3 have the value 1 if the corresponding physical signal is correct, and value 0 if it is incorrect. When the user specifies the behavior which takes place inside each unit (described later), the computers, arbiter and monitor are defined so as to agree on the meaning of each possible value on each link. In this way, only the information necessary for reliability evaluation is transferred between units.

It should be emphasized here that the meaning of the numbers carried on each link is completely specified by the person designing

the model. The second and third examples of Section 4 use models in which some links actually do carry binary digits corresponding to physical logic levels in the circuit. Some links in the third example are vectors of length 5, of which three components are binary digits, one component signifies the program state of a computer, and the fifth component indicates one of several degrees of correctness/incorrectness for that computer's output bus. Clearly, the method of specification is general enough to simulate the transfer of both real signals and abstract information between units.

2.3 Faults

Figures 2 and 3 show the basic specifications for unit types 1 and 2. Notice the input and output definitions correspond to the conventions discussed in the preceding paragraphs. Also, notice that each type has several fault-equivalence classes (FECs); unit-type 1 has two FECs and unit-type 2 has one FEC. Each FEC corresponds to those faults in the physical system which would all cause the same effect. For example, all the faults represented by FEC-1 of unit-type 2 (Figure 3) will cause the arbiter selection to be random. Since this is the only FEC for the arbiter, the model assumes that only this kind of fault can occur in an arbiter circuit. Unit-type 1 has two kinds of faults, those which can be detected with special fault-detection hardware, and those which cannot. Faults from either FEC will cause the computer data output (port 1) to be incorrect.



OUTPUT 1 = 0 data output is incorrect.
= 1 data output is correct.

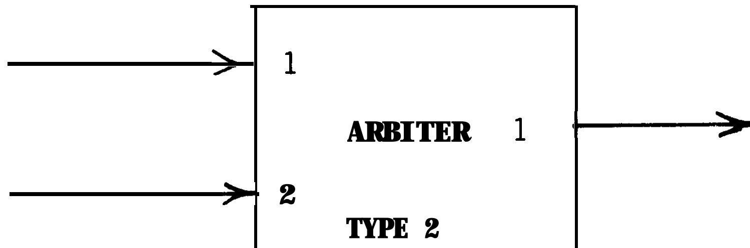
OUTPUT 2 = 0 a fault is detected in this computer.
= 1 no fault is detected in this computer.

FEC 1 = 0 no undetectable fault is active in this computer.
≥ 1 at least one undetectable fault is active in this computer.

FEC 2 = 0 no detectable fault is active in this computer.
≥ 1 at least one detectable fault is active in this computer.

STATE-VARIABLE 1 = 0 computer memory has been contaminated by a fault.
= 1 computer memory is not contaminated.

Figure 2. Definition of a Computer Unit (TYPE 1).



INPUT 1 = 0 a fault is detected in computer 1.
= 1 no fault is detected in computer 1.

INPUT 2 = 0 a fault is detected in computer 2.
= 1 no fault is detected in computer 2.

OUTPUT 1 = 1 arbiter selects computer 1.
= 2 arbiter selects computer 2.

FEC 1 = 0 no arbiter fault is active.
≥ 1 a fault is active which causes the arbiter to always select the same computer, regardless of the values on the input links.

STATE-VARIABLE 1 = 0 no fault was active the last time the arbiter function was evaluated.
= 1 a fault was active last time, and the output was forced to select computer 1.
= 2 a fault was active last time, and the output was forced to select computer 2.
(State-variable 1 is concerned with faults occurring in the arbiter, not faults in the computers.)

Figure 3. Definition of an Arbiter Unit (TYPE 2).

An FEC can be _active_ or _inactive_. An active FEC means that at least one of its faults is active in the physical system. The simulation of one mission will usually begin with all faults in the inactive state, which in the real system would mean that it was thoroughly pretested before the start of the mission. The simulator automatically changes the fault states as the simulated mission "progresses" through time, according to various user specifications concerning the occurrence and duration of faults within each FEC. For each FEC, the simulated time interval spent in each active or inactive state is generated randomly according to a probability distribution supplied by the user. This requires selecting the law of distribution, then setting the parameters used in that kind of distribution. The various types now available include exponential distributions (constant rate), Weibull (ageing), normal, constant, uniform, Pascal, etc. The current version of the simulator allows the user to specify (for each FEC) separate distributions for the active and inactive time intervals. A future version will offer two more distributions to more accurately simulate faults which have a burst-mode characteristic; for example faults due to loose connectors or transmission-line noise. During simulation, the utility package automatically generates the random time intervals and changes the fault states accordingly. In order to allow complete generality in the model, the user may specify the state (active/inactive) to which each FEC is initialized at the beginning of every simulated mission. This provides a way to model systems which are not thoroughly pretested.

For the examples shown in Figures 2 and 3, when we refer to one FEC, it is implicitly assumed that there could be many different faults in the FEC. Since the term "single fault" has many meanings in the literature, this report will use single fault to imply division of the FEC into the greatest number of parts such that any single part alone can cause the effect related to that FEC. A single fault is then that part of the FEC (even though in the hardware such a single fault may be a multiple point failure). For digital circuits a single fault might be an integrated-circuit pin stuck-at-1 or stuck-at-0, and an FEC would include all such pin failures having the same effect on the circuit's operation. On the other hand, an FEC might consist of only one single fault, such as the failure of a power-supply connector.

The number of single faults in an FEC is the multiplicity of that FEC. If an FEC has a multiplicity greater than one, each single fault must have a constant failure and repair rate, to allow efficient generation of state transitions for the FEC. The FEC then has a number of active states equal to its multiplicity; and its state is determined by the total number of single faults assumed to be present at that time in the real system. In this way, the program accurately simulates the simultaneous occurrence of a large number of physical faults.

Returning to the example of Figure 1, suppose computer 1 has 500 integrated circuit pins whose failures can be detected by the

fault-detection hardware, and 50 pins whose failures cannot be detected. A failure of any one of the 550 pins will cause incorrect computer data output. The user just specifies multiplicities of 50 and 500 for FEC 1 and 2, respectively, and assigns Poisson distributions for failure and repair rates of a single pin to each FEC. Each FEC will be inactive only when none of its faults are active.

The various parameters discussed above are specified by the user just before the start of simulation. The length of link vectors, initial link values, connections of links to units, and the fault generation parameters must be supplied to the simulation package in a deck of punched cards, which has a standardized format for all the required information. A sample deck for the model of Figure 1 is shown in Appendix A, Figure A7.

2.4 Simulated Units

As discussed in the example of Figure 1, each simulated unit must perform a specific operation. A computer, for instance, must test the state of its FECs and change the values of its data and fault-detection outputs accordingly. The user defines these operations by supplying a short FORTRAN subroutine for each type of unit. These type subroutines are called from the main simulator program during the simulation process, when there are changes in any of the parameters which might affect the output of the units. Each subroutine has access during the simulation (through a predefined COMMON area) to the values on the links at its input ports, to the states of its FECs (whether active or inactive),

and to other parameters. The form of the COMMON statement is standardized, so that access will be similar for every type subroutine. It should be noted that the subroutines do not require any information about unit interconnections or link numbers, because the utility package takes care of updating the COMMON area for all the values present on the links. As far as the user is concerned, there are two arrays, with preassigned names, acting as the input and output ports. The subroutine is also given a preassigned index pointer to test the entries in an array for information about fault-states in the unit. By means of this preassigned pointer, the subroutine for one type of unit can distinguish between the array entries for all the units of the same type. The subroutines for each type of unit in Figure 1 are listed in Appendix A, Figures A3, A4, and A5.

The specifications of the computer and arbiter units, Figures 2 and 3, include some _state-variables_ for each unit. State-variables and FEC states are different: the former is only for the convenience of the user in the programming of type subroutines, while the latter carries all the information concerning the faults. The state-variables are only tested and assigned values by the type subroutines which the user supplies, and are used to save information between successive calls to the subroutine for each unit. Access by each type subroutine to its state-variables is achieved by the same pointer as for the FEC array.

A good illustration of how a state-variable is used is shown in the definition of the arbiter, Figure 3. When an FEC first becomes active in the arbiter, its subroutine determines whether the fault forced the decision of the arbiter towards computer 1 or computer 2. The arbiter output should indicate the same selection as long as the FEC remains active. The arbiter subroutine uses a state-variable to indicate whether or not the FEC was active during the previous call and its effect, so that the subroutine will not repeat the analysis of the fault. It should be noted that the "states" of the simulated arbiter do not necessarily correspond to the states of the physical arbiter, even though the user has that option. A simulation state-variable may hold the value stored in a physical register or flip-flop, for instance, or it may indicate the overall condition of a software process for a computer. The value stored in each state-variable at the beginning of a simulation is specified by the user in the specially-formatted card deck. Although the examples only have one state-variable each, in general any unit can have any number of state-variables.

When a unit's fault state or input link value changes, the main simulation program sets the index pointer for that unit and calls the subroutine for that type. The subroutine tests the input values, fault entries, and old state of that unit to derive the new state values, which are stored in the state-variable array. All inputs, faults, and new state values are then tested to determine the final values for the unit outputs, and these are stored in the pre-defined output buffer array. Its task completed, the subroutine then does a RETURN to the main program which called it.

The model simulates the time required for information to propagate through a unit to its output, and considers the transmission delay of all links to be zero. Each output port of each unit is assigned its own time delay probability distribution by the user. Before the utility package calls the subroutine for a particular unit, the simulated time delay for each output is randomly generated according to its distribution, and the time delays are stored in a COMMON array which can be assessed and changed by the type subroutine. The subroutine then has the option of deciding whether to use these values or to change them. The computer unit defined in Figure 2 illustrates the use of this feature. An active fault may cause an 'error' signal to appear as the value at output port 1 after a time interval described by the Pascal distribution (to take care of the latency problem), but when the fault changes back to the inactive state, the corresponding value on the output link should immediately revert to 'no error'. The user would then assign the Pascal distribution to that output. The utility program generates a value T for the delay every time it is required. When the type subroutine finds the fault active, then the time-delay will be left set, but when the fault is inactive. the subroutine replaces that value with 0.0 to make the output change immediately. This method allows complete generality in specifying time delays through each component of the network.

It is possible that a variable such as a time delay would always have a constant value during any one mission, but that the value would

be randomly distributed between missions. The simulation package allows the user to choose, separately for each unit output in the model, whether the time delay is of this **type.** The proper values are computed automatically by the simulation program.

As mentioned above, the utility programs must occasionally generate a random number from various probability distributions. This same facility is made available to the user as a special utility program which can be called from any of the type subroutines. This program is called with a variable that selects the type of distribution, followed by the numerical parameters required by that distribution. Successive calls to the program return independent random variables. The type subroutines can thus obtain random time delays for the unit outputs, or can test the random values for probabilistic branching in the program. Where it is not efficient *or* desirable to simulate exactly the behavior of complex systems, the user may choose to model the statistical properties of some components with appropriate random variables.

The function of each element in the simulation is completely left to the user who is building the model for a real system. This provides a great degree of both generality and freedom while allowing the model to be very specific at selected points. The value on a data link may signify exact binary digits, a strobe signal, or an indication of correctness/incorrectness, etc. At the gate level, a link may be three-valued to signify high, low, and undefined logic levels; and at a computer level, a link may be many-valued to signify

various modes of incorrectness. Similarly, the specifics about state variables and fault states are left to the wish of the user. Because the utility package does all the fault generation and transfer of link values, the amount of effort required to accurately model system components is reduced. All the work put forth by the user is strictly concentrated on model building (and not model running). The level of specification is completely arbitrary and need not be homogeneous so that the same model may focus on some parts of the system in great detail while treating less interesting parts of the system in more general terms. The user is not forced to exactly define parts of the system he is not interested in at the time, and the simulator does not waste time and money simulating those parts in detail. When one section of the system has been thoroughly studied, that part of the model can be replaced by more simple units which accurately reflect the reliability aspects of that section.

To use the simulator, the user supplies two card decks to the utility package. One deck includes the subroutines which perform the functions of each type of component. The second deck is punched in a special format, and sets the link vector lengths, link initial values, types of each unit, and which links are connected to which input and output ports for each unit. For each unit, it sets the initial state variables, initial fault states, fault multiplicities, fault probability distributions, and output time-delay distributions. The second deck also provides some special parameters such as the initial state of the system (described in the next section).

Since all units of the same type have the same type subroutine, extensive use of the simulator would lead to the formation of a library of subroutines which would represent all the various types of components used to design the real system. This way, the most time consuming part of defining the model, writing the subroutines, would be done only once for each different kind of component. Also, the designer can change the specification of all units of the same type in one step by altering or replacing the subroutine for that type.

3. THE SIMULATION PROCESS

The simulation package uses a standard Monte Carlo analysis to determine the reliability characteristics of the model. This means that it simulates a large number of missions to find the probability that the mission does not 'fail' before a certain time has elapsed. The events which occur are likely to be different for different missions, because they are randomly generated by the simulation program according to probability distributions specified by the user.

The simulation of one mission is event-driven and asynchronous. An event is a change of some element in the model, usually either a change of fault state or a change of a link value. All events occur at a specific simulated time which is completely determined when the event is generated. Throughout a mission there will always be events in the system which are waiting to take effect because the simulation has not yet progressed to the simulated times at which those changes occur. These are called future events. When a future event is generated, it is inserted into a next-event list, which is ordered from top to bottom according to the time at which each event will occur. A typical entry of the list would consist of a time, link number, vector of new values for that link, and a number identifying this entry as a link-change type of event. A fault-change type of event would include a unit number, FEC number, and new fault state instead of the link parameters.

Refer to Figure 4 for a flow-chart description of the simulation process. The simulation of one mission begins by initializing all

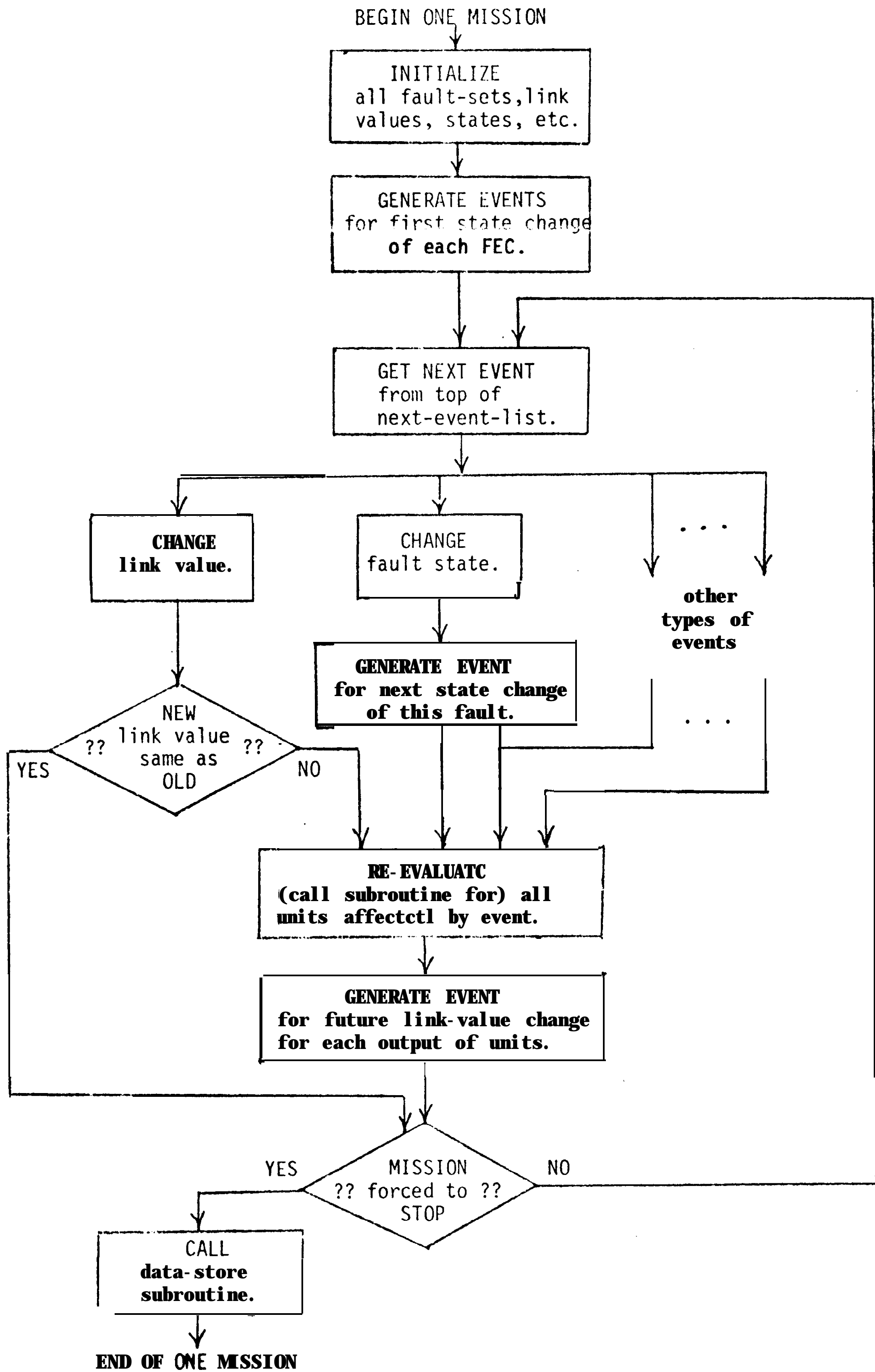


Figure 4. Simulation of One Mission.

fault states, link values, and unit state-variables to the values specified by the user. The utility package then generates one event for each FEC in the model, using the probability distributions assigned to them. These events, which typically are the first transitions to the active state for each FEC, are stored in the next-event list. The process then continually goes through a cycle which always begins by getting the next-event off the top of the list. The "current simulated time" always jumps to the time of this next-event, and the event action is processed immediately. When the event is completed, the simulator returns to the beginning of the cycle for the next event.

Usually, the processing of one event will cause other events to be generated and inserted into the next-event list. When a link value changes, all the units which have that link as an input are re-evaluated. For a unit to be re-evaluated means that the main program sets up the input buffer and calls the subroutine for that unit. When the type subroutine returns to the utility program, each output link from that unit is stored in the next-event list (this, because of the time delay) for a future change of the link value. The times of these future events are computed by adding the current simulated time to the output time delays, and the future link values are those left in the output buffer by the type sub-. routine. When a fault-change event is processed, it causes the re-evaluation only of the unit in which that fault-set is located. Also, the utility program will automatically generate another (future)

to know when to load all the new input data into its state variables. This feature facilitates the modeling of synchronous digital circuits.

It is possible that a generated event such as the change of a link value could make obsolete a future event waiting in the **next-** event list (for example another change in the same link). This is due to the randomness of the output time delays for each unit. The utility program automatically solves this problem by searching the next-event list for link-change events which are superceded by a more recently generated event. If an old event of this type is found, it is deleted from the list without being processed. A typical situation where this is required occurs when an FEC becomes inactive in a unit while the output of that unit is scheduled to become erroneous at some later time. Then the future event, the transition from correct to erroneous value, will be removed from the next-event list. This process allows a correct simulation of a system in which future events and time delays are randomly generated.

At some point during the simulation of a mission, one of the type subroutines supplied by the user must decide that the mission has failed, using some predefined failure criterion. A utility program is available which can be called by any type subroutine; when called, it forces the mission to stop at the beginning of the next event cycle.

As the mission progresses, the type subroutines must be able to store data pertaining to reliability (such as the simulated time before mission failure) for the purpose of statistical evaluation

fault-change event which will cause the next change of state for that FEC. When an event is processed, it causes a re-evaluation of units only if the event actually induces a real change in the system. For instance, if the new values for a link are identical to the old values, then the connected units will not be re-evaluated. Thus, a single change anywhere in the model will only propagate where it causes other changes to occur, and when the model reaches a "steady-state", no more events will be generated. This means that the entire model doesn't have to be simulated in response to one change somewhere in the system.

Some units will have input links which, when their values change, should not force a re-evaluation of the unit. This usually occurs when there is a synchronizing link which acts like a clock signal in a digital circuit; that is, the circuit does not respond to changes on the other input lines until the clock lines change. When specifying the link interconnections, the user can distinguish, separately for each input, whether or not a link value change on that input will force a reevaluation of that unit. The usefulness of this feature is illustrated by a unit representing a clocked register, which stores the binary digits present at its data inputs only when the clock input makes a low-to-high level transition. The model is defined such that only the input link carrying the clock signal will force a reevaluation of the unit. The subroutine must only test for the clock to be high

of the system after a large number of missions have been simulated. The simulator package has a standardized method to achieve this. It is done by allowing the user to supply one subroutine which will record the data from each mission, then evaluate the stored data at the end of the simulation run. This subroutine will be called by the the utility package before the first mission and immediately after each mission so the user can initialize tables and save vital data. It is also possible to call this subroutine from any of the type subroutines to record relevant data during simulation of any mission. This is shown in Figures 4 and 5. An index value is sent to the subroutine each time it is called to indicate the specific purpose of that call. When any subroutine determines that enough missions have been simulated for the desired accuracy of the results, it calls another auxiliary routine supplied in the utility package; this other routine forces the simulator to stop after the end of the current mission. After the last mission, the data gathering subroutine is called one more time with an index value which tells it to evaluate the accumulated numbers and plot a reliability curve for the system.

Most applications of the simulator will require the same type of data storage and reliability analysis. A general purpose version of this subroutine has been developed which will record the times-to-failure for each mission and print out a listing of reliability vs. time for the system, including the boundaries of the 80% and 90% confidence intervals on reliability at each time coordinate. This service facilitates the basic purpose of the simulator, namely, reliability evaluation, but still allows the user to collect and

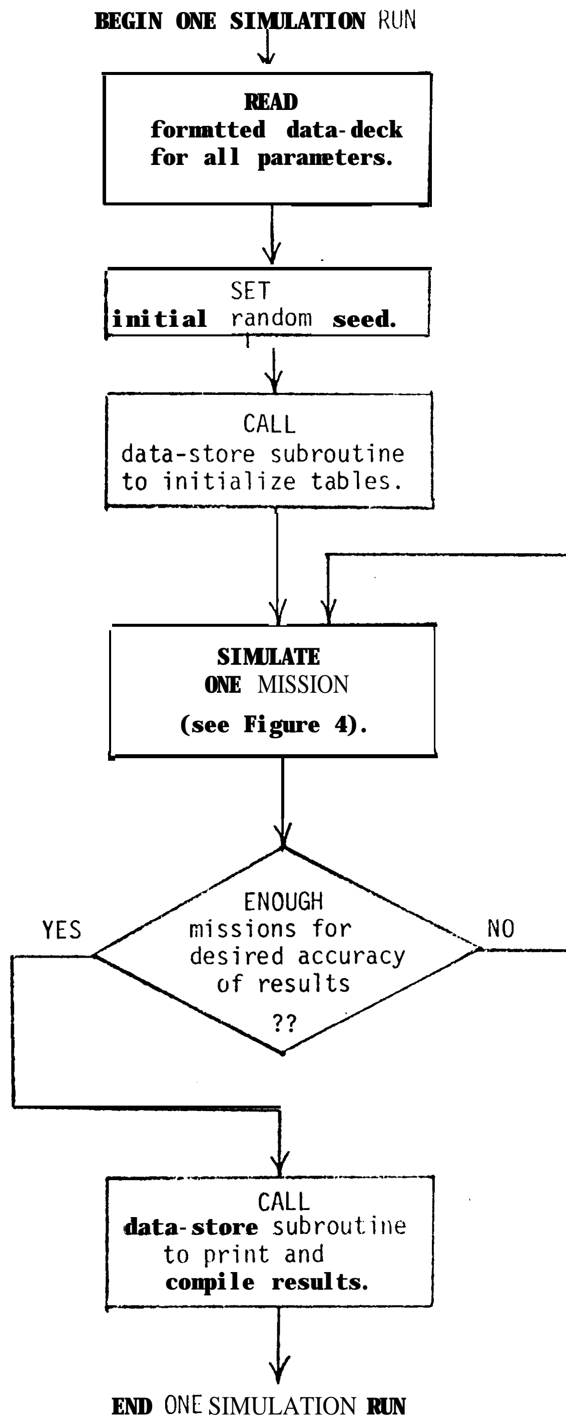


Figure 5. One Monte Carlo Simulation Run.

compile any kind of statistics relating to the system behavior.

In the previous section we mentioned the specially formatted data deck in which the user specified numerous parameters for the model. Also included in this deck are various parameters relating to the entire simulation process, such as the starting and maximum simulated times for each mission, the maximum number of missions to simulate, the random seed for the pseudorandom number generator, and selection of different output operations. The utility programs have options to automatically print a complete trace of all events and event-list changes for each mission as an aid to help the user debug the model. Also, by allowing the user to specify the random seed for the random number generator (rather than using the time of day, for instance), the results of any simulation can be repetitively obtained so that debugging or validation of the model is made easier.

The formatted data deck may also specify any events the user wishes to insert into the next-event list at the beginning of each mission. These may be any one of the four types of events which may occur in the model, and must include exact values for each parameter of the event. This feature is useful when investigating the behavior of the model in specific situations. The user may force **FECs** to change state in a particular sequence, fur example, or could test the response of the model to changes of link values which represent primary inputs to the system. This ability, coupled with the option to print listings of all events, forms a powerful debugging tool for the person who designs the model.

4. THREE EXAMPLES

This section describes three examples in which the simulator was used to evaluate the reliability of a redundant system. Examples a and b include a simulation of a system with simple enough features to enable it to be evaluated analytically, thus demonstrating the validity of the simulation package. Example c compares simulation results with the results of an analysis in which it was necessary to make simplifying assumptions. All three examples also include the simulation of a nontrivial system which would be too complex to handle mathematically, demonstrating the ability of the simulator to obtain results not easily derivable with current analytic techniques.

Examples a and b show plots with a range of reliability from 0.0 to 1.0. Most digital systems, however, are designed to have a reliability of no less than 0.95 during their entire period of use. The simulator can produce useful results in the higher ranges of reliability, as demonstrated in example c, for which the reliability never falls below 0.9999980. For very high reliabilities, the simulator program requires a slight modification (described in another report[Thompson, 1977A]) of its fault generation section in order to be cost-effective for the required accuracy. The range used for examples a and b demonstrate that the simulator results are verified analytically over all values of reliability.

4.1 Example a - A General Dual Computer System

Figure 6 shows a dual computer system with an arbiter. This example is defined in great detail in Appendix A, so in the following, only the basic simulation model is described.

In this dual system, two identical computers run concurrently, executing the same programs. In the simulation, each computer has two output links, one indicating whether the data output is correct or incorrect, and the other indicating whether a fault has been detected or not. The fault-detection outputs both go to the arbiter unit, which uses that information to make a selection of which of the two computers has correct data output. The arbiter always chooses one or the other, and the mission is assumed to be failed as soon as the arbiter selects a data output which is incorrect. Referring to Figure A2 in Appendix A, the arbiter selection link and both data output links are inputs to a fourth unit which acts as a monitor for the mission. The monitor stores times-to-failure in the data-storage subroutine and decides when the mission should stop.

The complete model includes faults in the arbiter, and both detectable and undetectable faults in each computer. Different random delays may be assigned to the computer outputs to simulate the time interval between the change of state of fault-sets (from inactive to active) and the first corresponding change which appears at the unit's output. The time delays for the data and detection outputs are called, respectively, the error-latency and detection-latency. Thus, if the error-latency of a computer fault is less

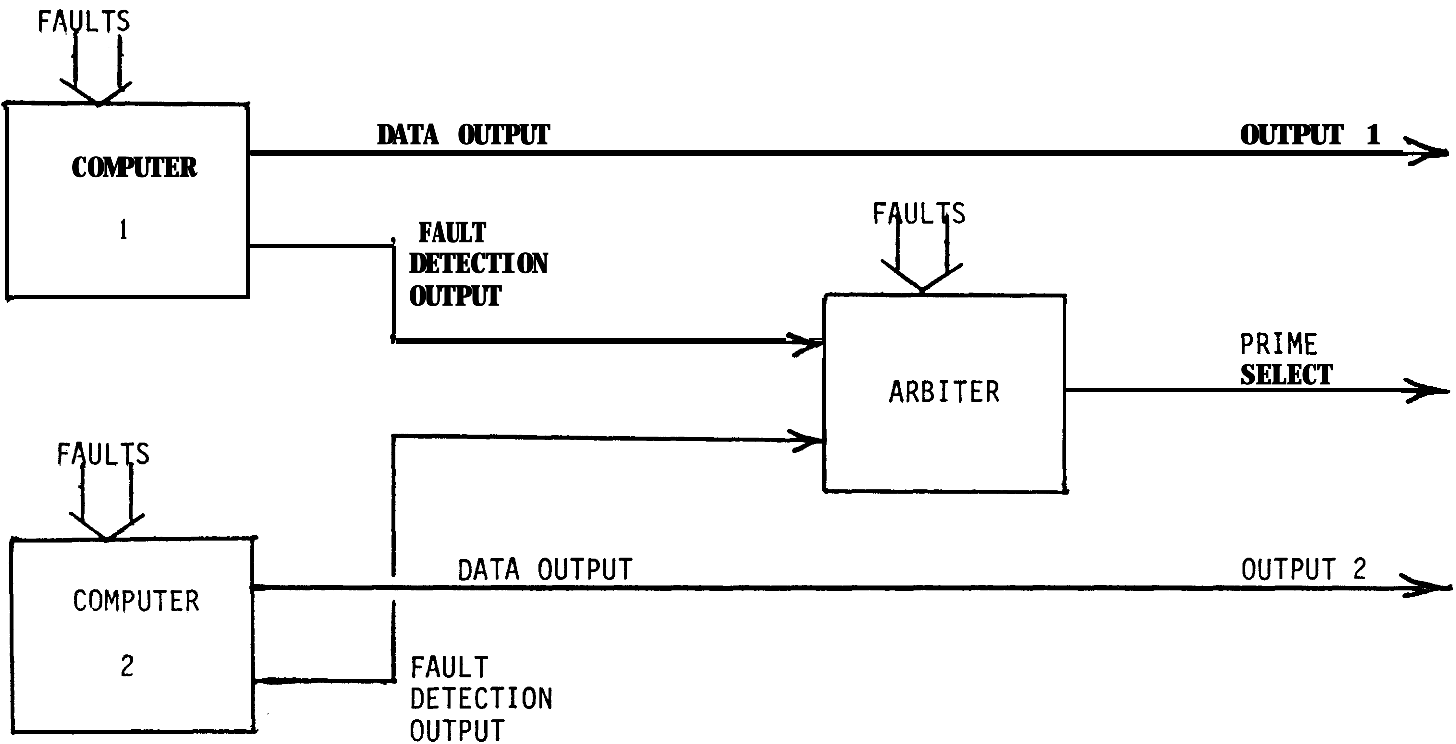


Figure 6. A Dual Computer System.

than the detection-latency, the data output will become erroneous before the existence of this fault is indicated to the arbiter. In the simple case (Figure 7), both latencies are constant 0.0, but in the complex case (Figure 8), they are randomly generated from a Pascal probability distribution. The arbiter randomly selects one computer when the fault detection inputs are ambiguous, or when a fault is active within the arbiter unit itself.

The simple case of the dual system will consider the arbiter to be perfect, i.e. have a zero failure rate. The nondetectable faults within the computers also have zero failure rates. The arrival times for detectable faults in each computer have an exponential distribution to simulate a constant failure rate λ. For the sake of the experiment, λ was assigned the value 10^{-4} failures per time unit, but the meaning of one time unit in terms of hours, days, minutes, etc. is left to the interpretation of the user, as long as all parameters are interpreted consistently. All faults are permanent faults and the latency times are zero. The system with these parameters is modeled analytically in Appendix B. Figure 7 shows plots of the reliability as a function of time for both the analytical analysis, as well as the results determined by the simulator. The analytically computed curves are shown as a solid line. We see that the dual configuration shows a significantly higher reliability than a single computer for this set of parameter values. It can also be seen that the analytical curves correspond very closely with the simulation plots.

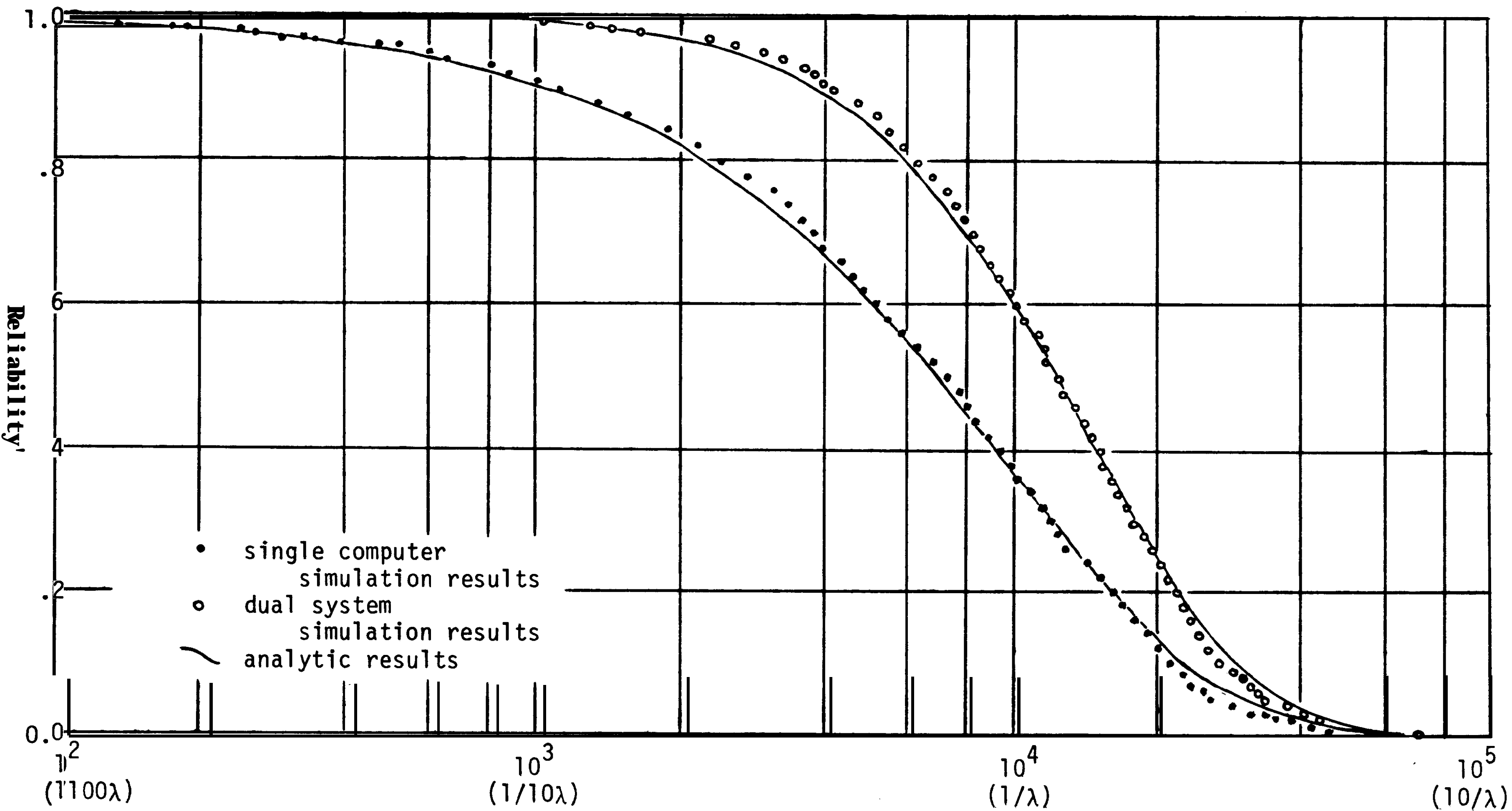


Figure 7. Reliability Curves for Simple Dual Computer System.

No undetectable computer faults or arbiter faults.

Detectable faults: arrivals exponentially distributed, $\lambda = 10^{-4}$ failures/time unit.
All faults permanent (zero rate of repair).
Zero time latencies for fault detection and errors.

The differences between the plots arise because of the loss of accuracy resulting from the fact that only a finite number of missions have been simulated (500 in this case). Increasing the number of missions will improve the accuracy.

A second more complex characteristic of the system has its reliability plots indicated in Figure 8. This sytem has both undetectable and detectable faults possible in the computers. There are 500 detectable and 50 undetectable faults possible in each computer. The arrival times of the faults are exponentially distributed as before, but they are different for the two classes of failures. The duration of the faults is also exponentially distributed; they have error and detection latencies which are Pascal distributions. The arbiter, in addition, is not perfect but has a single fault which is Weibull distributed. This system is obviously too complex to be able to be modeled analytically. The simulator results are shown in Figure 8. In this case, we see that the dual system is actually less reliable than a single computer. This can be attributed to the facts that the arbiter was not a fault-free unit, that undetectable faults could occur in the computers, and that the detection latency (i.e., the time between the occurrence of a fault and the time that it is detected) was usually larger than the error latency for the faults in the computers. By varying these parameters, one could see how each contributes to the system unreliability, and for what values the dual system becomes less reliable than the single computer.

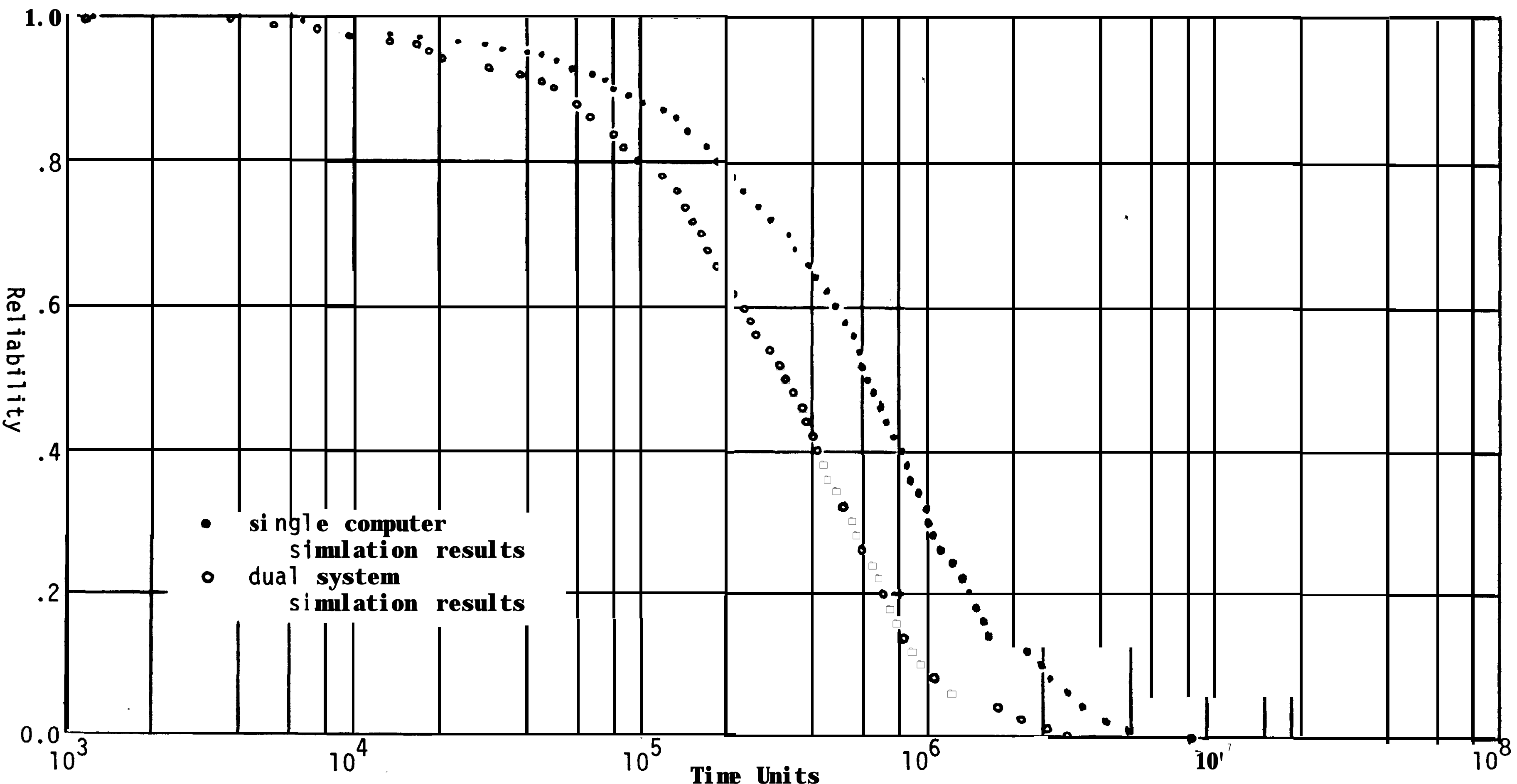


Figure 8. Reliability Curves for Complex Dual Computer System

50 undetectable and 500 detectable faults in each computer.

Detectable faults: arrival exponential, $\lambda_o = 10^{-7}$ failures/time unit,
duration exponential, $\lambda_D = 10^{-4}$ repairs/time unit.

Undetectable faults: arrival exponential, $\lambda_o = 10^{-6}$ fai;ures/time unit,
duration exponential, $\lambda_D = 10^{-4}$ re airs/time unit.

Arbiter fault (single): arrival exponential, $\lambda_o = 0.5 \times 10^{-4}$ failures/time unit,
duration Weibull, $\lambda_D = 10^{-4}$ repairs/time unit, ageing factor 1.1.

Error latencies are Pascal, probability of error is 0.05 for each 10.0 time units,
Fault detection latencies are Pascal, probability of detection is 0.03 each 10.0 time units,

4.2 Example b - A TMR Network

A triple-modular redundant **(TMR)** system is shown in Figure 9. Three identical modules' outputs are fed to a voter unit which outputs a value based on the majority of the input values. The modules will be assumed to be NAND gates with two inputs. The faults that can occur are stuck-at-one (s-a-l) faults or stuck-at-zero (s-a-O) faults both on the inputs to the NAND gates and the outputs. The voter unit is assumed to be fault-free. An exponential probability distribution characterizes the occurrence of the faults, and a Weibull distribution describes the duration of transient faults. The latency times of the faults are again described by a Pascal distribution. The parameters of the gate input faults can be made to be different from the gate output faults. In addition, the frequency of occurrence of s-a-l faults can be made to differ from that of s-a-0 faults.

Four different reliability measures were obtained from each simulation of this system; the probability that a specific pin on a gate would fail, the probability that a NAND module would have at least one failed pin, the probability that at least two out of three modules have failed, and the probability that the voter output is incorrect. Standard analysis of TMR systems assumes that when at least two modules are failed, the system has failed. This is not always correct in a system such as TMR NAND gates, because if one module's output is stuck-at-l and another module's output is stuck-at-O, they will compensate for each other at the voter.

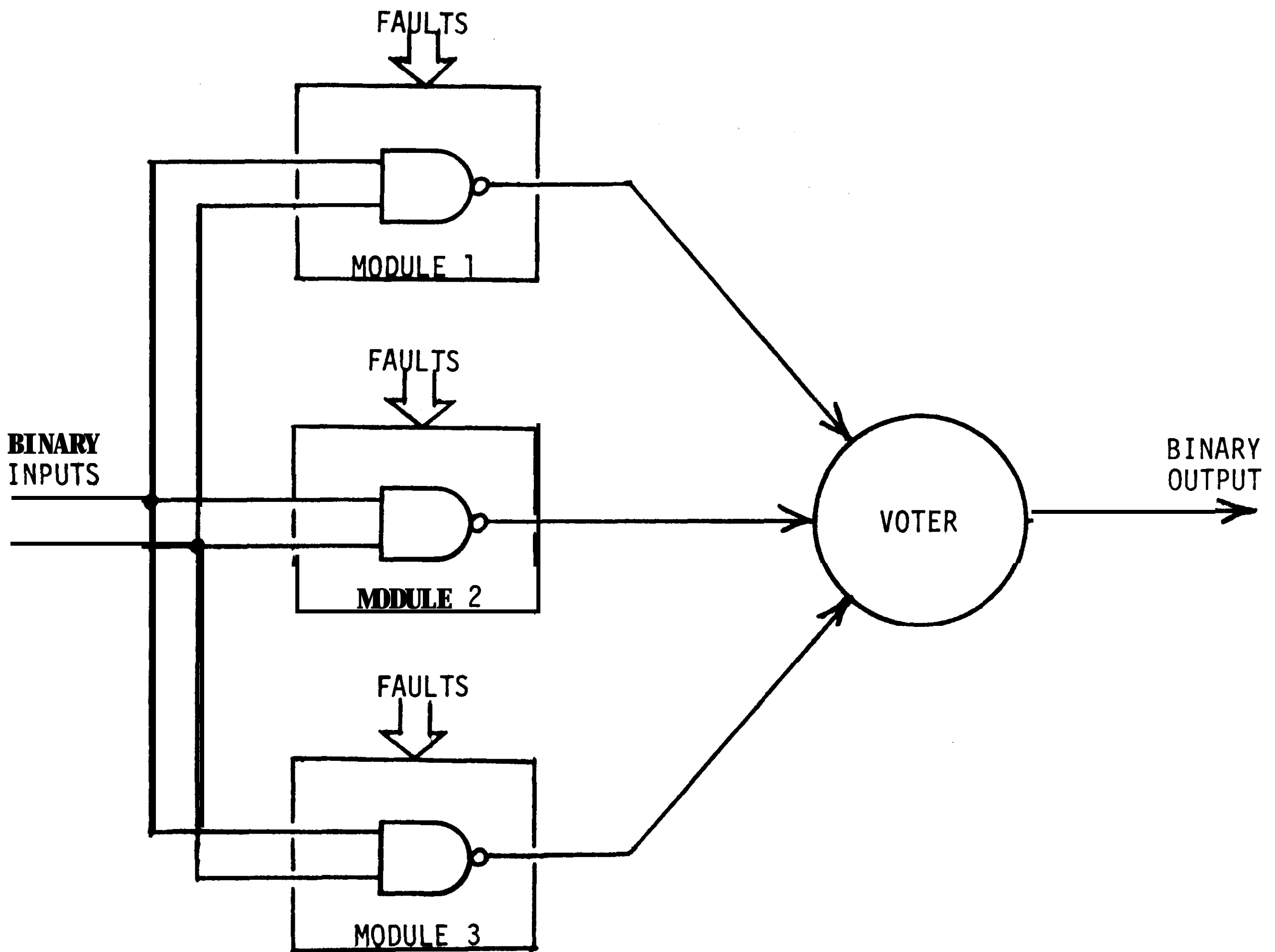


Figure 9, Triple Modular Redundant (TMR) NAND-Gate System

Thus, when compensating failures are taken into account, the circuit may still perform the correct function even though two modules have failed. The first three reliability measures are easily derived from analytical techniques in Appendix B, but the actual circuit reliability is very difficult to obtain analytically due to the problem of compensating failures [Siewiorek, 1971].

Although this circuit could have been simulated with only four units, it was decided to use many units, each of which performed a very simple function. The final model is shown in Appendix A, Figure A10, and consists of twenty units and thirty links. There were five unit types: NAND gates, threshold gates, comparators, negative-edge detectors, and stuck-at-X units. Faults only occurred in the stuck-at-X units, which were linked in series with each NAND gate input and output. The subroutine describing each unit type was very simple to write, and demonstrates the way in which a library for standard components can be formed for modeling a large digital system. The relatively large number of units and links had no noticeable effect on the efficiency of the simulation, because the simulation process only deals with the parts of the system which are caused to change by the occurrence of a fault.

As before, we will consider a simple case of the system which can be modeled analytically, and comparison will be made with the results obtained using the simulator. The simple system considers only permanent faults with exponential distribution for their occurrence. The latency time of the faults is zero, and s-a-1 faults are

as equally likely to occur as s-a-0 faults. The reliability curves obtained through both analytical and simulation methods are shown in Figure 10.

The solid lines indicate the analytically computed results. Once again, there is a close correspondence between the analytic and simulation results. The modular TMR (2-out-of-3) curve shows an improvement in reliability over a single module only when the reliability of both is greater than 0.5. The actual functional reliability, which takes into account all compensating failures, shows that the standard TMR analysis is very pessimistic. In fact, the TMR circuit is better than a single gate over the entire range of reliability.

The more complex example considers all faults to be transient faults with occurrence exponentially distributed, and duration Weibull distributed. The probability distributions of s-a-l faults differ from those of s-a-0 faults and the error-latency times are nonzero. Figure 11 shows the reliability of the TMR system with no compensating failures considered and when compensating failures are considered. These curves are very easy to obtain using the simulator, because changing a parameter value only requires repunching a card in the formatted data deck.

4.3 Example c - A Dual Computer System for Navigation

The simulation package was used to evaluate a dual computer system which was designed and built at the Charles Stark Draper Laboratories at M.I.T. [Ressler, 1973]. The system was intended to provide reliable navigation computing for airborne applications.

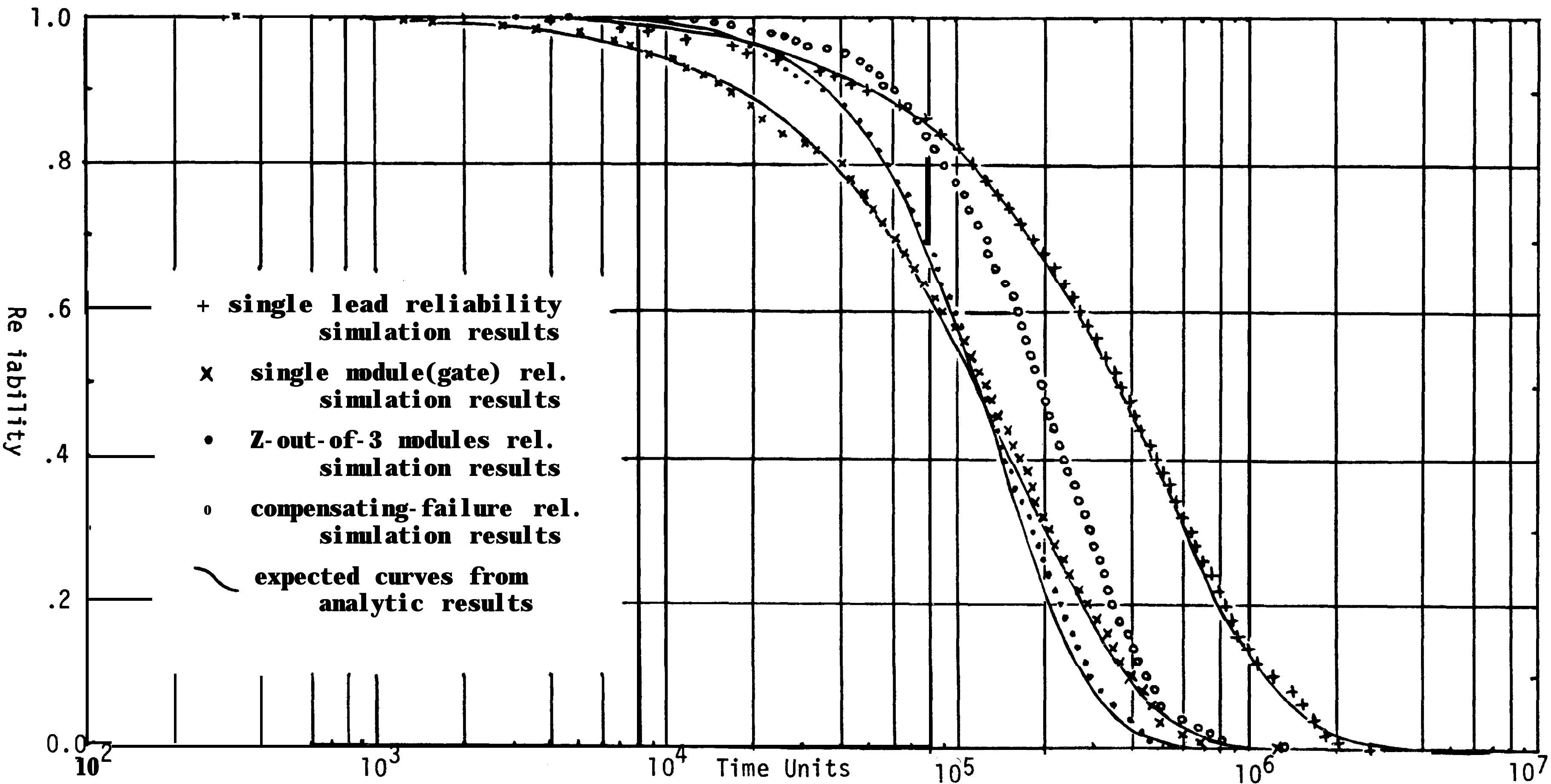


Figure 10. Reliability Curves for Simple TMR NAND-Gate System

All stuck-at-0 and stuck-at-1 faults for inputs and outputs are exponentially distributed, $\lambda_0 = 10^{-6}$ failures/time unit.
All faults are permanent (zero repair rate).
Zero error latency for all faults.

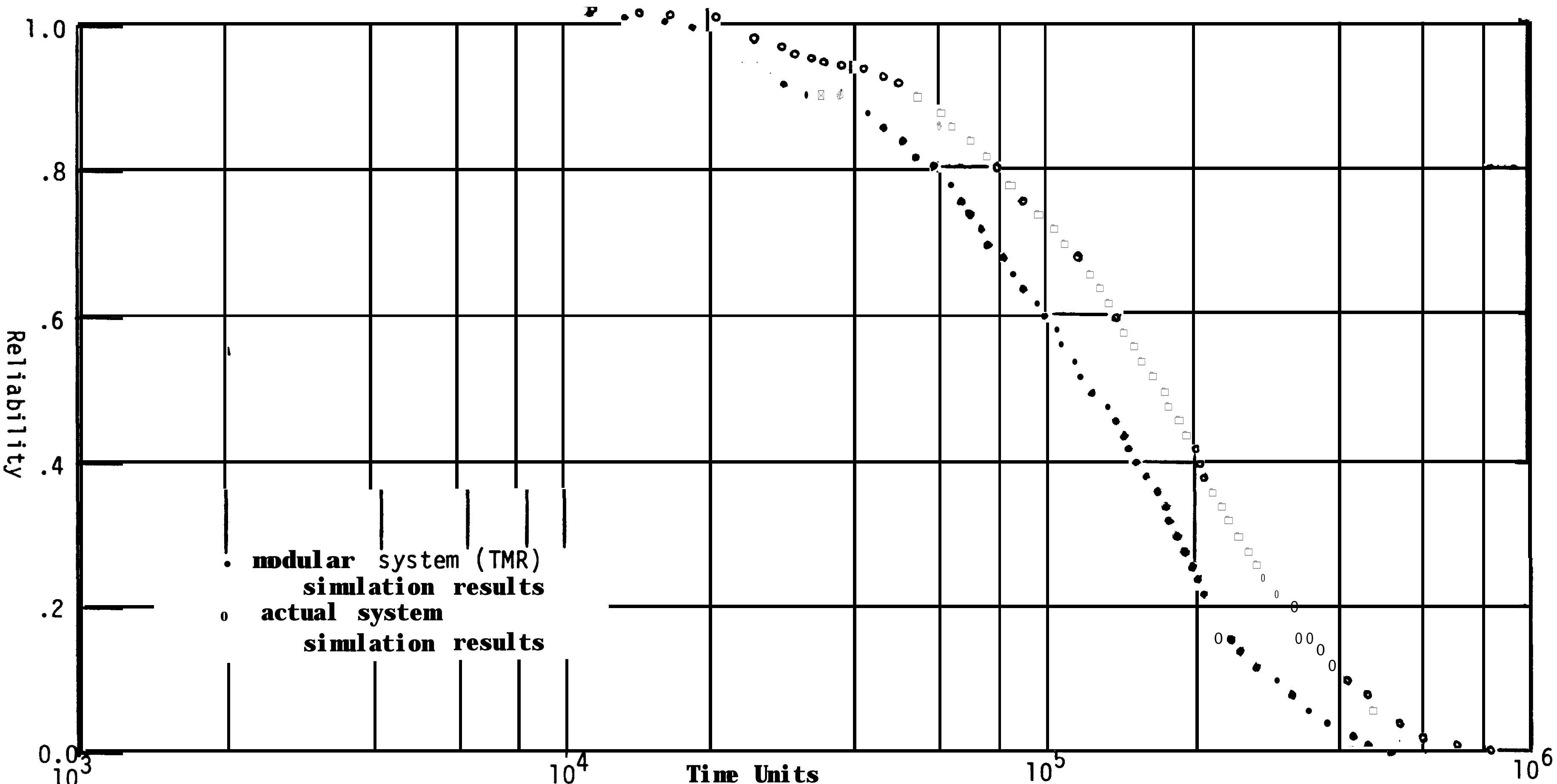


Figure 11. Reliablity Curves for Complex TMR NAND-Gate System

Inputs **stuck-at-0: arrival** exponential, $\lambda_o=1.5\times10^{-6}$ **failures/time unit,**
duration Weibull, $\lambda_D=10^{-6}$ **repairs/time unit, ageing factor 1.1.**

Inputs **stuck-at-1: arrival** exponential, $\lambda_o=0.5\times10^{-6}$ **failures/time unit,**
duration Weibull, $\lambda_D=10^{-6}$ **repairs/time unit, ageing factor 1.1.**

Outputs **stuck-at-0: arrival** exponential, $\lambda_o=0.5\times10^{-6}$ **failures/time unit,**
duration Weibull, $\lambda_D=10^{-6}$ **repairs/time unit, ageing factor 1.1.**

Outputs stuck-at-1:arrival exponential, $\lambda_o=1.5\times10^{-6}$ **failures/time unit,**
duration Weibull, $\lambda_D=10^{-6}$ **repairs/time unit, ageing factor 1.1.**
Inputs error latencies Pascal, prob. of error is 0.2 for each 1000.0 time units.
Outputs error latencies Pascal, prob. of error is 0.5 for each 1000.0 time units.

The architecture, shown in Figure 12, uses two Honeywell-316 mini-computers to do concurrent calculations from the raw sensor data. The updated navigational position is sent through channel transmit and receive circuits to a display at the pilot's console. An extensive variety of hardware and software fault-detection mechanisms continually test all parts of each half of the system, and all the test and opinion outputs are gathered at the arbiter circuit for a final decision regarding which data display is correct. The two computers can communicate with each other to compare output data, or to transfer status words during the recovery of one computer from a transient fault. As in example a, the mission is assumed to fail when the arbiter selects an output which displays incorrect data.

The actual link/unit model of this system is shown in Appendix A, Figure All. The computers were divided into functional s&systems so as to accurately simulate the effect of a fault within that part of the computer. By studying the structure of the Honeywell-316 computer, a complex representation of its behavior in the presence of faults was derived. The model included abnormal program execution, loss of navigational status words, loss of recovery or diagnostic software, physical failures in the timing generator, absorbing and non-absorbing failed states for the program counter, etc. In some cases, the system chooses from a variety of possible effects with a probability distribution adjusted for the fault states, and in other cases the fault state completely determines the effect. Due to the way in which the computer was divided, it was possible to later

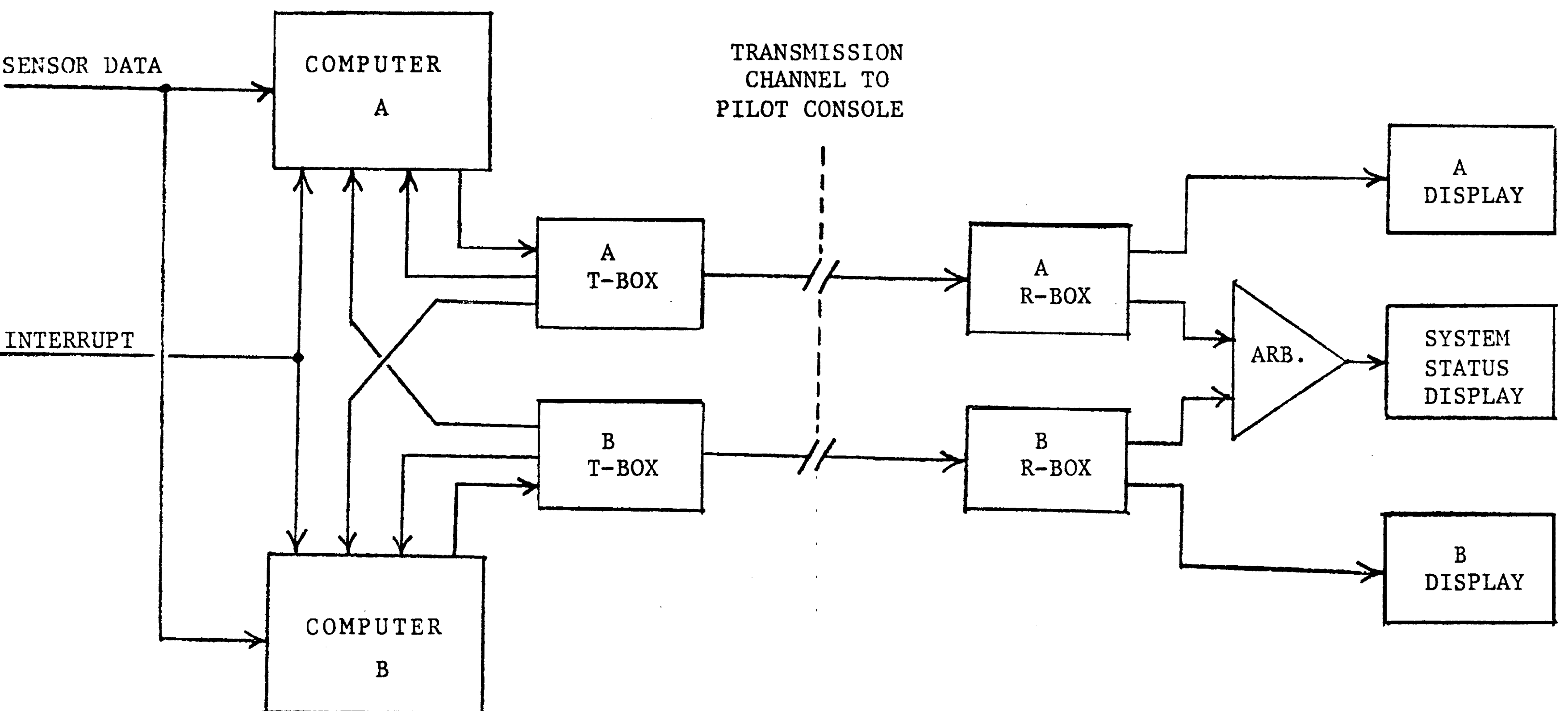


Figure 12. A Dual Computer System for Navigation.

replace the Honeywell-316 with a model for a more state-of-the-art MSI minicomputer simply by changing various parameters in the formatted data deck. It should be emphasized that the meaning of the link values dealt with the state of correctness/incorrectness transferred in the real system between one circuit and another. The simulation did not execute a navigation program on the model, and no navigational data was transferred on any link, rather the system as a whole propagated between its parts the effects of particular faults.

The interface, channel transmission, and fault-detection hardware was modeled to the extent that the effects of faults could be accurately simulated. Faults were modeled which affected only the data and not certain test signals, or only the test signals and the cross-communication, etc. The simulator differentiated between faults which would make a fault-detector be stuck-at-good from those which would make it stuck-at-bad, and took into account the masking of one such fault by another fault.

The final model is composed of twenty-seven units, representing fifteen unit types. These are connected by fifty-seven links, most of which are actually vectors of length up to six. The entire model used sixty FECs, representing approximately 24,600 single integrated circuit pin faults for the system using Honeywell-316 computers. Since the maximum mission time was ten hours at a very high reliability, the desired accuracy implied a very large number of simulated missions. So as to reduce the

computer time needed to do the simulation, the fault-event generator was extended to take advantage of the memoryless property of the exponential fault distribution. This option, coupled with the ability to pre-test the initial fault events to find only those missions which have potential to fail, allowed an effective simulation of a large number of missions at a very reasonable cost. For example, an effective simulation of 1.1 billion ten-hour missions required about twenty minutes of execution time on a CDC-7600 computer, obtaining unreliability measures on the order of 10^{-7}.

Figure 13 compares the analytic curve* for unreliability with that obtained using simulation for a system which has Honeywell-316 computers. The values of λ_O and λ_D are the rates of failure and repair for each integrated-circuit pin in the system. The analytic results are seen to be pessimistic (have a higher unreliability) due to the inability of the mathematical model to accurately handle arbiter and fault-detector failures. In Figure 14, the Honeywells are replaced by a state-of-the-art MSI minicomputer, which lowers the overall pin count from 24,600 to 5,800, so the arbiter and fault-detector failures contribute a larger proportion to the overall reliability. This increases the discrepancy between the analytic and simulation results, as seen in the figure. The simulation runs also found that about 10 per cent of the mission failures for the MSI version occurred when one of the two sets of data output was still correct, as opposed to only 2 per cent for the Honeywell-316 version. Again, this is due to failures occurring in the fault-detectors and the arbiter. Figure 15 demonstrates a study made of

*[Thompson, 1977A] contains more information on the analysis used.

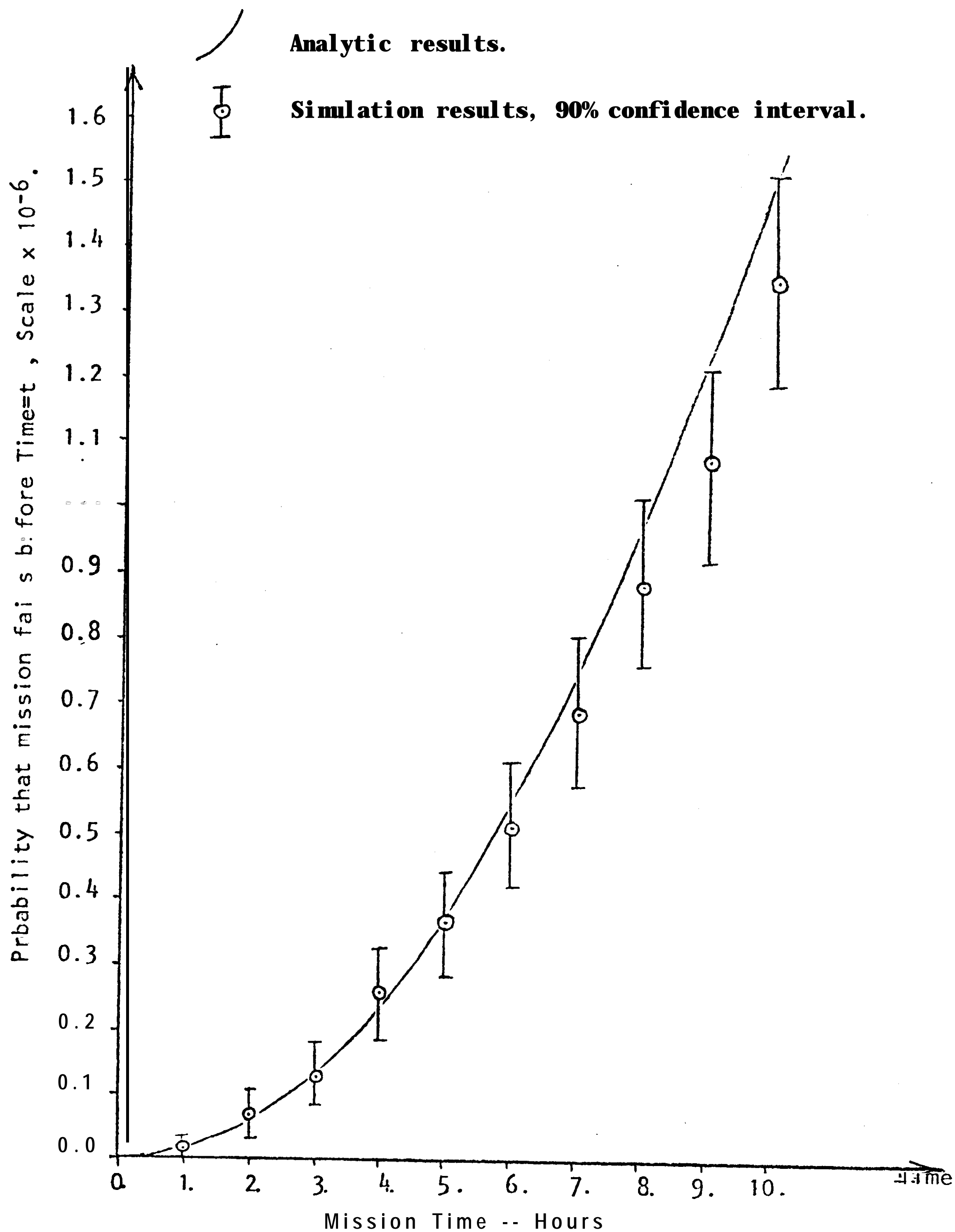


Figure 13. Unreliability Curve Using Honeywell-316 **Computers.**
All single faults have arrival exponential, $\lambda_o=10^{-8}$ **failures/hour, and duration exponential,** $\lambda_b=0.5$ **repairs/hour. The simulation** run effectively included 1.4×10^{8} **ten-hour missions, of which** 194 failed **before the end of ten hours. The vertical axis measures** Unreliability, **which is defined to be** 1.0-(Reliability).

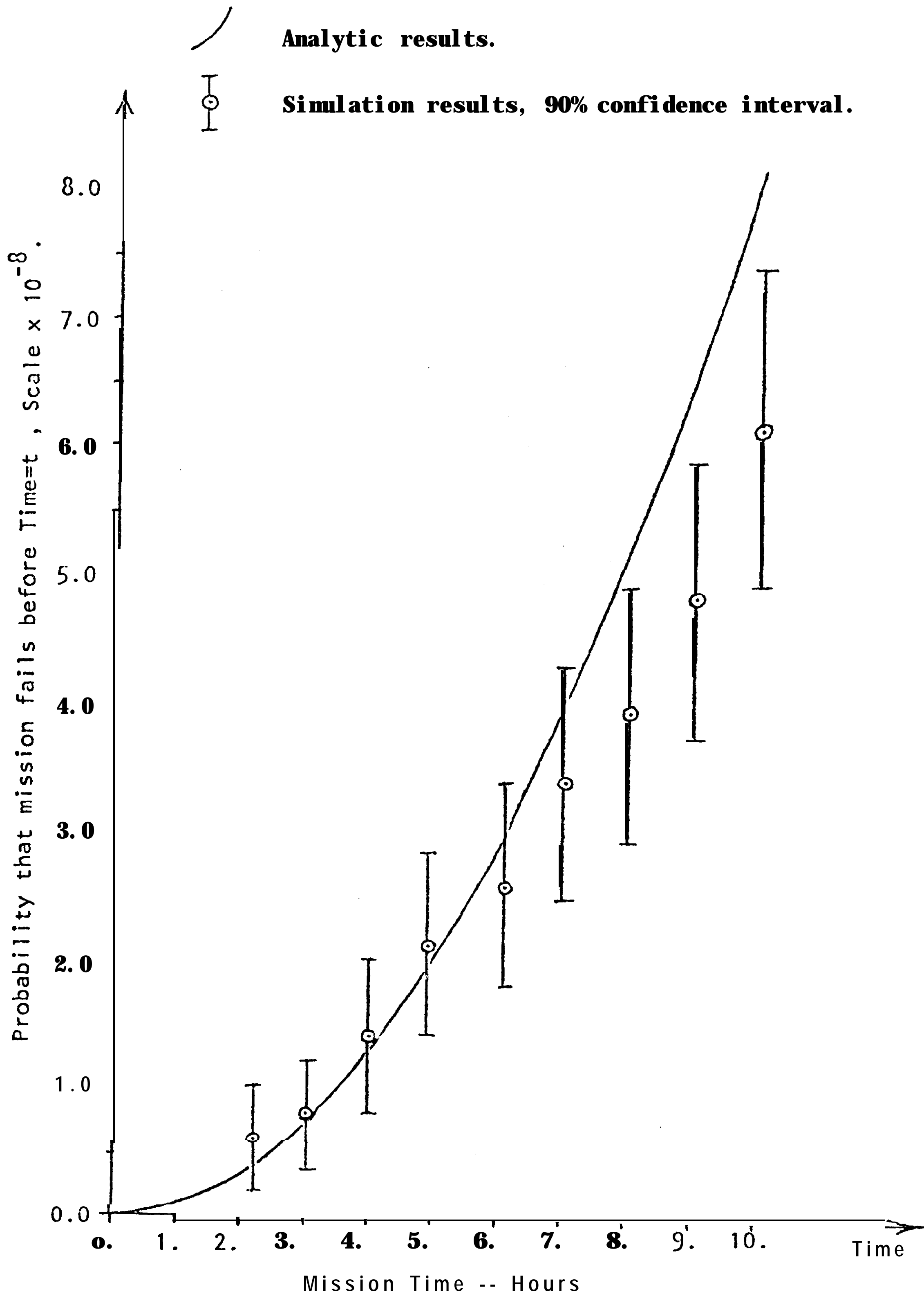
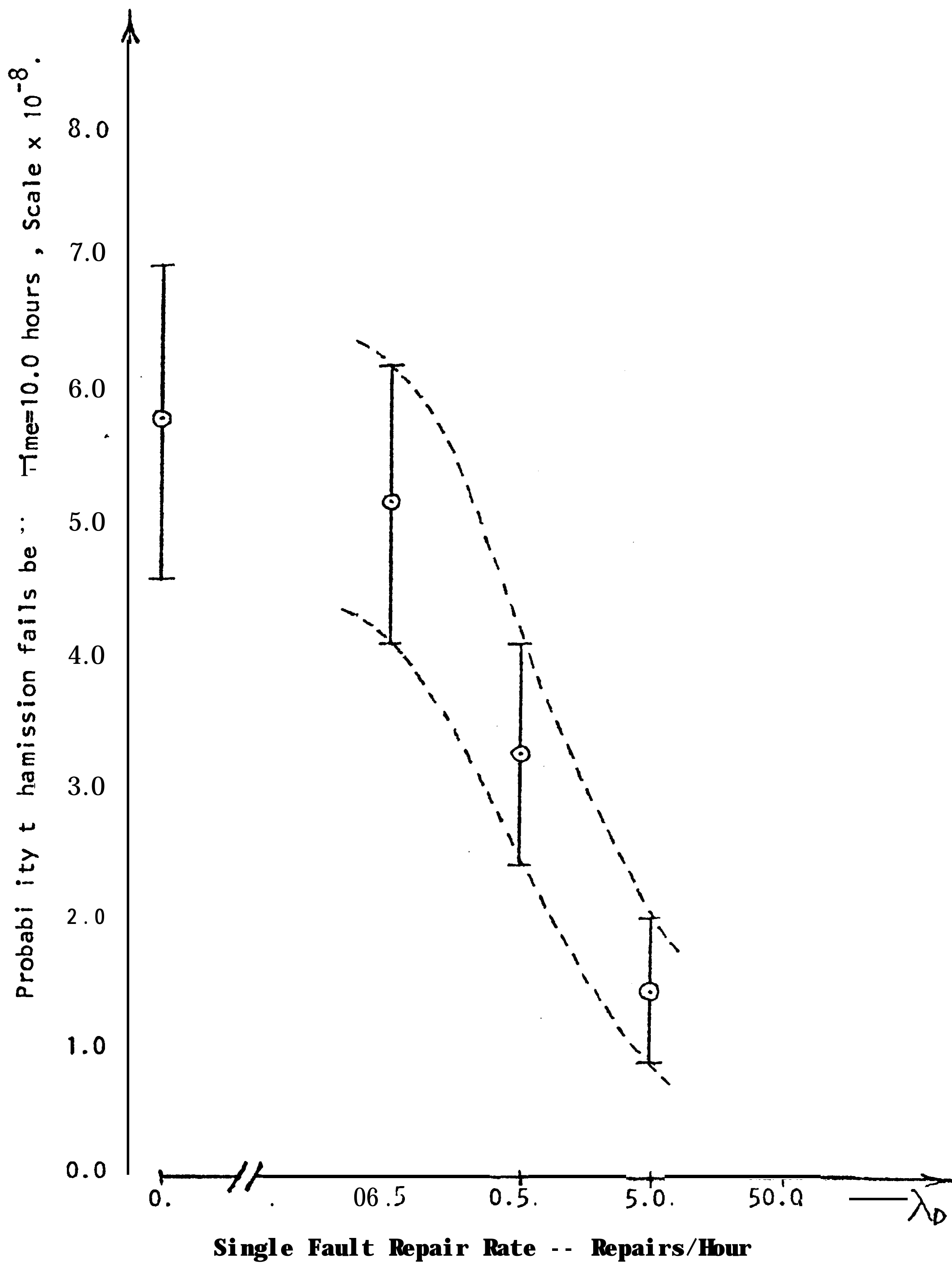


Figure 14. Unreliability Curve Using MSI Computers.
All single faults have arrival exponential, $\lambda_0=10^{-8}$ failures/hour, and duration exponential, $\lambda_D=0.5$ repairs/hour. The simulation run effectively included 1.1×10^9 ten-hour missions, of which 69 failed before the end of ten hours.



Single Fault Repair Rate -- Repairs/Hour

Figure 15. Unreliability Curves Using Different Repair Rates.

All single faults have arrival exponential, $\lambda_0=10^{-8}$ failures/hour, and repair rates as shown on the horizontal axis. The vertical axis measures unreliability of the system at the end of one ten-hour mission. Each simulation run effectively included about 1.2×10^9 ten-hour missions; the 90% confidence intervals are drawn for each point.

the repair rate (λ_D) for pin faults. The simulator was also used to test specific changes in the circuit relating to the use of computer self-diagnosis for recovery. The ability to test incremental design changes was very useful, because it was used to validate or invalidate conjectures regarding improvements in the system design from the standpoint of reliability. The complete definition of the model used for this example is provided in another report [Thompson, 1977A].

5. COMPARISON WITH OTHER SIMULATORS

There are four basic types of general simulators geared to the study of computer systems. The first, typified by GPSS and SIMSCRIPT [Fishman, 1973], simulates process control networks with transactions flowing between processes. This level of modeling is not particularly suited to the evaluation of reliability for a physical system, and the lack of certain features, such as floating point numbers for GPSS, impose restrictions on the person defining the model. The other three types of simulation can be more easily used to study reliability.

The method of simulation used by CAST [Conn, 1974] is typical of the second type. The system is modeled as a set of connected modules, where each module is one of several kinds of predefined functional subsystems. Each represents a basic part, such as a. CPU, Memory, Input, etc. The function performed by each module is defined in the simulator itself; the user can assign values to the parameters of each module for the type of error detection, fault distributions, modular fault recovery, number of spares, and other parameters relating to the module as a whole. During simulation, each module passes through states such as operational, detected failure, undetected failure, recovered, etc. One simulator [Masreliez and Bjurman, 1976] uses Markov models to determine the state behavior of modules, and uses dependency trees to group modules. The effect of one module's state on the behavior of the system is completely determined by the module's position in a fixed configuration. The system configuration

is limited to modules in either series or parallel combinations with possible spares. Some simulators of this type, and specifically CAST, are only used to supply coverage and other parameters to an analytic model, rather than obtain the system reliability directly.

The third type of simulator could also be called an emulator, because the actual binary levels in a digital circuit are copied while faults are injected into the network. A good example is the ELSA program [Beaufils et al., 1974], which emulates the exact operation of specific integrated circuits to the extent that the execution of software on those circuits is identical to that in the physical system. ELSA can deterministically or randomly inject single and multiple stuck-at faults into the logic while emulating the execution of a real program on that logic, then classify the various fault combinations according to their effect on the network as a whole. The faults can be permanent or transient. A variety of useful parameters are measured, such as the detection-latency, probabilities of detecting a transient fault, probability that a transient fault affects the network, etc. The large number of digital components in a computer system prohibit the use of this type of simulator for the purpose of obtaining reliability curves directly. Usually each subsystem is emulated separately in detail, and their measured parameters are incorporated into another simulator or analytic model which considers each subsystem as a black box. Emulation at the integrated circuit level is almost always very costly in terms of

man-hours and computer time. Most circuit emulators reduce the computer time considerably by writing the emulation program in assembly language, thus taking advantage of the computer's word structure to effectively do many emulations in parallel.

The simulator described in this report is a representative of the fourth basic type. Unlike the other types, it cannot be labeled as either low-level or high-level, because the person defining the model always decides the meaning of each element in the system. Theoretically, this simulator could successfully copy any model studied on any of the other three simulator types and obtain the same results. In practice, this could not be done in some cases, due to the much better efficiency with which the other types can simulate certain specific kinds of systems. By limiting their range of possible models, each type of simulator can predefine various aspects of the overall specification and make available to the user a set of facilities which will always be useful (or required) for all the models within that range. For some models it is more efficient, in terms of programming hours and computer time, to use one of the specialized simulation programs. Also, the way in which a user defines a model for this simulator relates more to a physical system than a set of abstract processes, and thus would appeal more to a design engineer than to a computer scientist. A person who is more comfortable with program structures would tend to think of hardware systems in software terms, and would be more inclined to use GPSS or SIMSCRIPT.

The strongest feature about this simulator is its complete

generality. As the examples in Section 4 show, it can be used to simulate at a high 'black-box' level as well as at the more detailed digital circuit level. The third example simulated the behavior at the gate level, register level, and complex system level all in the same model. The functional definition of any unit can be arbitrarily simple or arbitrarily complex, and the way in which units are interconnected has no restriction. Since the simulator is event-driven, both synchronous and asynchronous networks can be modeled. The meaning of a fault is defined by the user. These characteristics are not found in the other simulators. The third type of simulator only has stuck-at faults on package leads, and never gets above the level of synchronous digital circuits. The second type of simulator cannot change the function of any module or the basic form of interconnection without extensively revising the simulator package, and does not actually simulate circuit-level operations. In some cases, the other types do not obtain reliability curves, but simply derive parameters to use in another simulator or analytic model. Thus, the simulator in this report is of a very general nature in that it can be used to obtain the reliability curves of a system as well as be geared to evaluate coverage factors and their sensitivity to various physical parameters.

It would be useful to take advantage of the best features of each type of simulation in the same program package. In particular, the fourth type of simulator often requires an accurate description of the faults which would occur in a unit representing a complex

digital system. A simulator such as ELSA could provide such data more efficiently than the general simulator itself; in fact, the way in which that unit interacts with other units in the general simulator would determine how the ELSA simulator categorizes the **FECs** for that unit. Alternatively, a type subroutine might call another subroutine which efficiently emulates a digital circuit at the gate level. The ELSA simulator could also use the general simulator to handle asynchronous circuit elements, or to provide more complex units to drive the gate-level network. The generality of the package described in this report makes possible the efficient coordination required to simulate several levels of detail simultaneously.

6. CONCLUSION AND FUTURE WORK

A very general purpose simulator for the evaluation of digital system reliability has been developed. The requirements of specifying a model for a system maximizes the freedom of the user to innovate, and allows him to simulate a hardware system at any arbitrary level of detail and complexity. Various facilities are made available to the user to allow definition of a model with randomly generated faults, probabilistic branching, and other standard elements. Some problems which typically arise in accurate simulation models, such as superceded events and synchronizing unit inputs, are automatically dealt with by the simulation package. The person specifying the model has a variety of options from which to choose for fault distributions, time delay distributions, and initialized events. The printed output options and random seed specification facilitate debugging the model. The simulation program and completed model are portable to any computer supporting standard FORTRAN.

The simulator has been successfully used to evaluate the reliability of three widely different digital circuits, representing a very wide range of circuit detail and general application. The computer time required for the simulations was reasonable for the desired accuracy of results.

When compared with other types of simulators, this simulator is found to provide a greater degree of flexibility in specifying the model. For specific types of models, other types of simulators

may be more efficient. The possibility of using a highly efficient circuit emulator in coordination with the general simulator has been discussed.

Future development of the simulation package could include introducing a second level of specification for interconnected units. The user will be able to define a block-type as a network of units, then will able to build a larger system using many blocks of various types. This allows a higher level of nesting similar to the relationship between units and unit-types, and should facilitate the design of more accurate models.

Future development could also include a more comprehensive data-gathering/compiling routine, and more standardized choices regarding the pre-testing of fault events at the beginning of each mission. A routine which plots reliability curves would also aid the user of the package.

The current version allows such a general specification of any system that there are many different ways in which one system can be modeled. Different designers will almost never choose the same unit divisions or link connections even for the same circuit. This is useful at times because different model designs may study different aspects of the same hardware, but it is not always clear which type of model provides the most complete, efficient study of reliability parameters. Study must be done to formulate rules of how best to partition and simulate a complex system so that the elements pertaining to reliability accurately reflect those in the

physical hardware. These rules could also provide a framework of standardization leading to a more systematic certification of a system's reliability.

7. APPENDIX A - SIMULATION EXAMPLES

This appendix includes a complete definition of the model used for Example a, A General Dual Computer System. Diagrams of the units and links used for Examples b and c are also provided.

Figure Al shows the convention used by the author to label various characteristics of a unit when put in a model diagram. It should be noted that input ports are distinct from output ports.

Figure A2 shows the unit/link model for Example a. Figures A3, A4, A5 and A6 list the user-supplied subroutines, with a table of what the various elements mean. The value of UNIT is pre-assigned by the simulation program before the type subroutines are called. The other names are:

IN(•,•)	Input buffer area.
OUTT(•,•)	Output buffer area.
FAULT(•,UNIT)	Fault state.
UV(•,UNIT)	Unit state-variables.
URAND(1)	Returns a uniform (0,1) random value.
TIME	Current simulated time.
MSTOP(TIME)	Force simulation of this mission to stop.
UTD(•)	Time delay for output port.

Figure A7 shows a sample formatted data deck for the model (the one used for the graph of Figure 8) and Figure A8 is the computer listing from that deck. Figure A9 is part of the simulation output for the graph of Figure 7.

Figure A10 shows the link/unit diagram for the model used in Example b, and Figure All shows the model used in Example c.

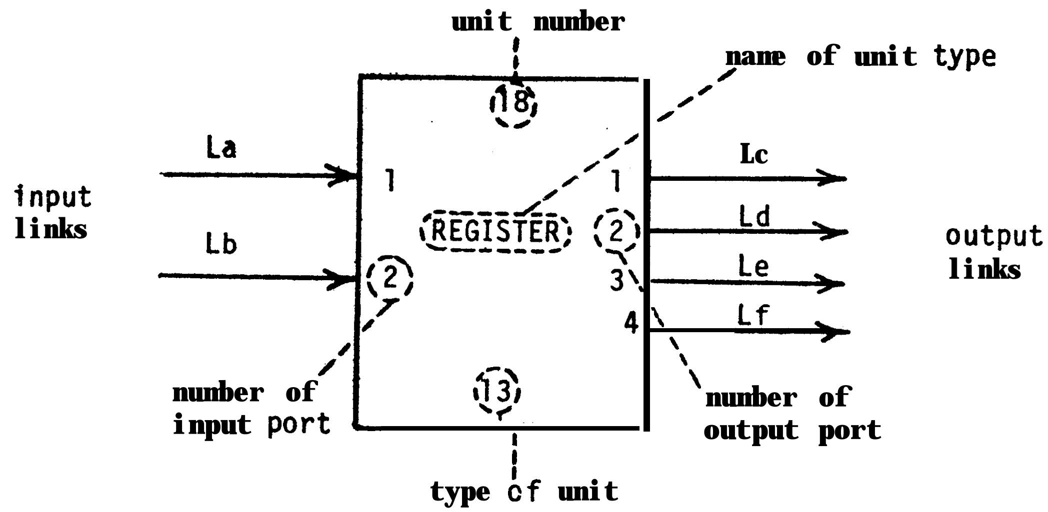


Fig. A1. Convention of Labelling Model Diagrams.

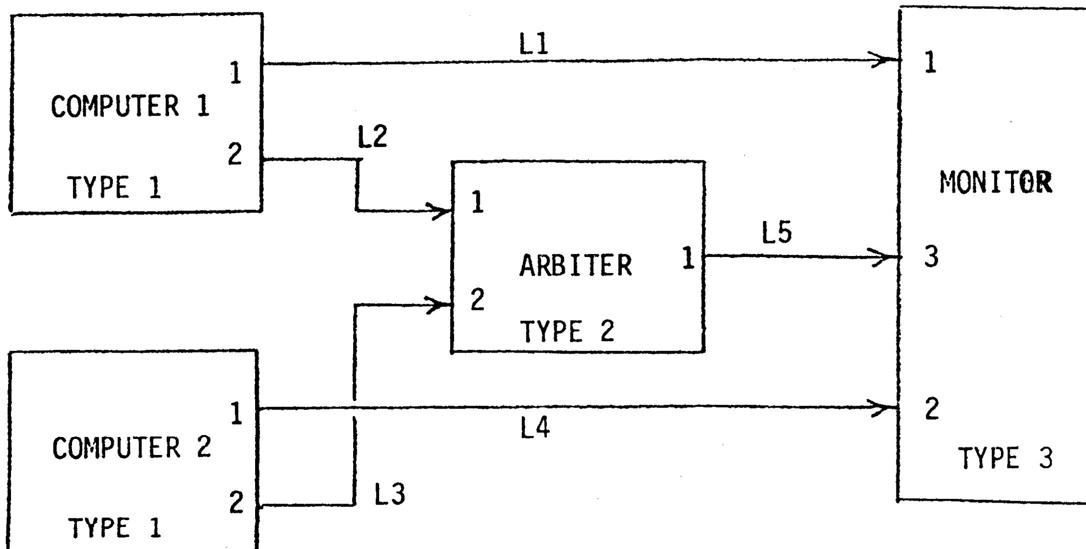


Figure A2. Model of a Dual Computer System with Arbiter.

```
      SUBROUTINE TYPE1
C COMPUTER UNIT WITH TWO FAULTS AND IMPERFECT ERROR DETECTION
      COMMON/USER/ IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)
     1             ,TIME,LIMIT,TITLE(20),SYSIN,SYSOUT,DATOUT
     2             ,MSSION,NMSSN,ISEED,IPRINT(5),UNIT
      REAL IN,OUTT,UV,UTD,TIME,LIMIT,TITLE
      INTEGER FAULT,SYSIN,SYSOUT,DATOUT,MSSION,NMSSN,ISEED,IPRINT,UNIT
C  ASSIGN DEFAULT OUTPUTS
      OUTT(1,1)=1.0
      OUTT(1,2)=1.0
C  TEST IF SECOND FAULT IS ACTIVE
      IF(FAULT(2,UNIT).EQ.0) GO TO 100
      UV(1,UNIT)=0.0
      OUTT(1,1)=0.0
      OUTT(1,2)=0.0
      GO TO 1000
C  TEST IF FIRST FAULT IS ACTIVE
  100 IF(FAULT(1,UNIT).EQ.0) GO TO 200
      UV(1,UNIT)=0.0
      OUTT(1,1)=0.0
      GO TO 1000
C TEST IF COMPUTER MEMORY HAS BEEN CONTAMINATED IN THE PAST
  200 IF(UV(1,UNIT).EQ.0) OUTT(1,1)=0.0
C IF ERROR ISN'T PROPOGATING TO OUTPUT SET TIME DELAYS TO ZERO
 1000 IF(OUTT(1,1).EQ.1.0)UTD(1)=0.0
      IF(OUTT(1,2).EQ.1.0)UTD(2)=0.0
C  ADJUST ERROR DETECTION OUTPUT FOR DELAY THROUGH ARBITER
      UTD(2)=UTD(2)-1.0
      RETURN
      END
```

OUTPUT 1 = 0 data output is incorrect.
= 1 data output is correct.

OUTPUT 2 = 0 a fault is detected in this computer.
= 1 no fault is detected in this computer.

FEC 1 = 0 no undetectable fault is active in this computer.
≥ 1 at least one undetectable fault is active in this computer.

FEC 2 = 0 no detectable fault is active in this computer.
≥ 1 at least one detectable fault is active in this computer.

STATE-VARIABLE 1 = 0 computer memory has been contaminated by a fault.
= 1 computer memory is not contaminated.

Figure A3. Definition of Computer Unit (TYPE 1).

```
      SUBROUTINE TYPE2
C ARBITER WITH ONE FAULT WHICH CAUSES RANDOM OUTPUT CHOICE
      COMMON/USER/ IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)
     1             ,TIME,LIMIT,TITLE(20),SYSIN,SYSOUT,DATOUT
     2             ,MSSION,NMSSN,ISEED,IPRINT(5),UNIT
      REAL IN,OUTT,UV,UTD,TIME,LIMIT,TITLE
      INTEGER FAULT,SYSIN,SYSOUT,DATOUT,MSSION,NMSSN,ISEED,IPRINT,UNIT
C  MAKE A RANDOM CHOICE
      OUTT(1,1)=1.0
      IF(UV(1,UNIT).LT.URAND(1)) OUTT(1,1)=2.0
C  TEST IF INPUTS DETERMINE OUTPUT, SET OUTPUT ACCORDINGLY
      IF((IN(1,1).EQ.1.0).AND.(IN(1,2).EQ.0.0)) OUTT(1,1)=1.0
      IF((IN(1,1).EQ.0.0).AND.(IN(1,2).EQ.1.0)) OUTT(1,1)=2.0
C  TEST IF FAULT IS CURRENTLY ACTIVE
      IF(FAULT(1,UNIT).EQ.0) GO TO 700
C  TEST IF ACTIVE FAULT HAS ALREADY BEEN DETECTED
      IF(UV(2,UNIT).GT.0.0) GO TO 600
C  FLAG ACTIVE FAULT DETECTED, MAKE RANDOM CHOICE FOR OUTPUT
      UV(2,UNIT)=1.0
      IF(URAND(1).GT.0.5) UV(2,UNIT)=2.0
  600 OUTT(1,1)=UV(2,UNIT)
      GO TO 1000
C  FLAG  FAULT IS NOW INACTIVE
  700 UV(2,UNIT)=0.0
 1000 RETURN
      END
```

INPUT 1 = 0 a fault is detected in computer 1.
= 1 no fault is detected in computer 1.

INPUT 2 = 0 a fault is detected in computer 2.
= 1 no fault is detected in computer 2.

OUTPUT 1 = 1 arbiter selects computer 1.
= 2 arbiter selects computer 2.

FEC 1 = 0 no arbiter fault is active.
≥ 1 a fault is active which causes the arbiter to always select the same computer, regardless of the values on the input links.

STATE-VARIABLE 1 = 0 no arbiter fault was active the last time the arbiter function was evaluated.
= 1 an arbiter fault was active last time, forcing the output to select computer 1.
= 2 an arbiter fault was active last time, forcing the output to select computer 2.
(State-variable 1 is affected only by faults occurring in the arbiter circuit, not by faults occurring in the computer units.)

Figure A4. Definition of an Arbiter Unit (TYPE 2).

```
      SUBROUTINE TYPE3
C MONITOR WHICH STORES TIMES-TO-FAILURE AND STOPS MISSION
      COMMON/USER/ IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)
     1             ,TIME,LIMIT,TITLE(20),SYSIN,SYSOUT,DATOUT
     2             ,MSSION,NMSSN,ISEED,IPRINT(5),UNIT
      REAL IN,OUTT,UV,UTD,TIME,LIMIT,TITLE
      INTEGER FAULT,SYSIN,SYSOUT,DATOUT,MSSION,NMSSN,ISEED,IPRINT,UNIT
C  TEST IF COMPUTER 1 HAS FAILED YET
      IF((UV(1,UNIT).EQ.0.0).OR.(IN(1,1).EQ.1.0)) GO TO 100
      CALL STASH(TIME,1)
      UV(1,UNIT)=0.0
C TEST IF COMPUTER 2 HAS FAILED YET
  100 IF((UV(2,UNIT).EQ.0.0).OR.(IN(1,2).EQ.1.0)) GO TO 200
      CALL STASH(TIME,2)
      UV(2,UNIT)=0.0
C  TEST IF MISSION IS FAILED
  200 IF(UV(3,UNIT).EQ.0.0) GO TO 300
      IF((IN(1,3).EQ.1.0).AND.(IN(1,1).EQ.1.0)) GO TO 300
      IF((IN(1,3).EQ.2.0).AND.(IN(1,2).EQ.1.0)) GO TO 300
      UV(3,UNIT)=0.0
      CALL STASH(TIME,3)
C WHEN EVERYTHING HAS FAILED AT LEAST ONCE, STOP THE MISSION
  300 IF(UV(1,UNIT)+UV(2,UNIT)+UV(3,UNIT).EQ.0.0) CALL MSTOP(TIME)
      RETURN
      END
```

```
               INPUT 1 = 0 data output of computer 1 is incorrect.
                       = 1 data output of computer 1 is correct.

               INPUT 2 = 0 data output of computer 2 is incorrect.
                       = 1 data output of computer 2 is correct.

               INPUT 3 = 1 arbiter decides that computer 1 has
                           correct output.
                       = 2 arbiter decides that computer 2 has
                           correct output.

      STATE-VARIABLE 1 = 0 data output of computer 1 has been
                           incorrect previously.
                       = 1 data output of computer 1 has never
                           been incorrect.

      STATE-VARIABLE 2 = 0 data output of computer 2 has been
                           incorrect previously.
                       = 1 data output of computer 2 has never
                           been incorrect.

      STATE-VARIABLE 3 = 0 arbiter has selected incorrect data
                           previously.
                       = 1 arbiter has not yet selected incorrect data.
```

Figure A5. Definition of a Monitor Unit (TYPE 3).

```
      SUBROUTINE STASH(R,I)
C DATA COLLECTION AND ANALYSIS
      COMMON/USER/ IN(8,8),OUTT(8,8),UV(8,40),UTD(8),FAULT(24,40)
     1             ,TIME,LIMIT,TITLE(20),SYSIN,SYSOUT,DATOUT
     2             ,MSSION,NMSSN,ISEED,IPRINT(5),UNIT
      REAL IN,OUTT,UV,UTD,TIME,LIMIT,TITLE
      INTEGER FAULT,SYSIN,SYSOUT,DATOUT,MSSION,NMSSN,ISEED,IPRINT,UNIT
      DIMENSION TSAVE(500,6),ITS(6)
      DATA IR,IC,ITS,TSAVE/500,6,6*0,3000*0.0/
      IF (I.NE.-1) GO TO 100
C INITIALIZE ARRAY IN WHICH TIMES WILL BE SAVED
      DO 10 K=1,IC
      DO 10 J=1,IR
   10 TSAVE(J,K)=LIMIT
      GO TO 1000
  100 IF (I.LT.1) GO TO 200
C STORE TIME TO FAILURE AT END OF A MISSION
      TSAVE(MSSION,I)=R
      ITS(I)=NMSSN
      GO TO 1000
  200 IF (I.NE.0) GO TO 1000
C CALL OUTPUT ROUTINE TO PLOT AND SORT TIMES TO FAILURE
      CALL OUTPUT(TSAVE,ITS,IR,IC)
 1000 RETURN
      END
```

CALLING PARAMETERS:

R = number to record for later analysis, usually the time-to-failure for some part of the system.

I = -1 initialize tables to maximum mission time.
  = 0 sort and plot reliability vs. time.
  = 1 save time at which computer 1 first has incorrect data.
  = 2 save time at which computer 2 first has incorrect data.
  = 3 save time at which arbiter first selects a computer with incorrect data.

Figure A6. Data-Storage Subroutine.

Description	Unit	Line										
GLOBAL SIMULATION PARAMETERS		1.	5	6	6							
		2.	DUAL COMPUTER WITH FAULTY ARBITER (COMPLEX EXAMPLE)									
		3.			0.0			1.E9	500 0 0 0 1 0			12345
FOR EACH LINK, VECTOR LENGTH AND INITIAL LINK VALUE.		4.	1	1			1.0					
		5.	2	1			1.0					
		6.	3	1			1.0					
		7.	4	1			1.0					
		8.	5	1			1.0					
		9.										
FOR EACH UNIT: INPUT LINKS, OUTPUT LINKS, OUTPUT TIME DELAY DISTRIBUTIONS, FAULT-EQ. CLASSES, STATE-VARIABLES. FOR EACH FEC: INITIAL STATE, MULTIPLICITY, OCCURRENCE DISTRIBUTION, DURATION DISTRIBUTION.	UNIT 1	10.	1	1	0	2	2	1				
		11.	1	1	9			.05	10.	0.0		
		12.	2	2	9			.03	10.	0.0		
		13.	1	1	1	0	50					
		14.		1.E-6				0.0	0.0	1.E-4	0.0	0.0
		15.	2	1	1	0	500					
		16.		1.E-7				0.0	0.0	1.E-4	0.0	0.0
		17.			1.0							
	UNIT 2	18.	2	1	0	2	2	1				
		19.	1	3	9			.05	10.	0.0		
		20.	2	4	9			.03	10.	0.0		
		21.	1	1	1	0	50					
		22.		1.E-6				0.0	0.0	1.E-4	0.0	0.0
		23.	2	1	1	0	500					
		24.		1.E-7				0.0	0.0	1.E-4	0.0	0.0
		25.			1.0							
	UNIT 3	26.	3	2	2	1	1	2				
		27.	2	4								
		28.	1	5	6			1.0	0.0	0.0		
		29.	1	1	5	0	1					
		30.		5.E-5				0.0	0.0	1.E-4	1.1	0.0
		31.			0.5			0.0				
	UNIT 4	32.	4	3	3	0	0	3				
		33.	1	3	5							
		34.			1.0			1.0	1.0			
		35.										
(EMPTY) LIST OF PRE-SET EVENTS →		36.										

Figure A7. Formatted Data-Deck for Complex Dual Computer Example.

```
DUAL COMPUTER WITH FAULTY ARBITER (COMPLEX EXAMPLE)

THE LOWER/UPPER TIME LIMITS ARE 0.0          / 0.100000E+10

     500 MISSIONS WITH STARTING SEED                                   12345

LINK     1 IS A VECTOR OF LENGTH     1
           INITIAL VALUES 0.100000E+01
LINK     2 IS A VECTOR OF LENGTH     1
           INITIAL VALUES 0.100000E+01
LINK     3 IS A VECTOR OF LENGTH     1
           INITIAL VALUES 0.100000E+01
LINK     4 IS A VECTOR OF LENGTH     1
           INITIAL VALUES 0.100000E+01
LINK     5 IS A VECTOR OF LENGTH     1
           INITIAL VALUES 0.100000E+01

UNIT     1 IS TYPE     1
  OUTPUT     1 TO LINK     1 WITH TIME DELAY DISTRIBUTION     9
                 TM DLY PARAMS     0.500000E-01    0.100000E+02    0.0
  OUTPUT     2 TO LINK     2 WITH TIME DELAY DISTRIBUTION     9
                 TM DLY PARAMS     0.300000E-01    0.100000E+02    0.0
   FAULT     1  INITIAL STATE      0      MULTIPLICITY        50
                 OCCURANC DIST     1
                 OCCRNC PARAMS     0.100000E-05    0.0             0.0
                 DURATION DIST     1
                 DURATN PARAMS     0.100000E-03    0.0             0.0
   FAULT     2  INITIAL STATE      0      MULTIPLICITY       500
                 OCCURANC DIST     1
                 OCCRNC PARAMS     0.100000E-06    0.0             0.0
                 DURATION DIST     1
                 DURATN PARAMS     0..100000E-03   0.0             0.0
   STVAR     1 HAS VALUE 0.100000E+01

UNIT     2 IS TYPE     1
  OUTPUT     1 TO LINK     3 WITH TIME DELAY DISTRIBUTION     9
                 TM DLY PARAMS     0.500000E-01    0.100000E+02    0.0
  OUTPUT     2 TO LINK     4 WITH TIME DELAY DISTRIBUTION     9
                 TM DLY PARAMS     0.300000E-01    0.100000E+02    0.0
   FAULT     1  INITIAL  STATE    0       MULTIPLICITY        50
                 OCCURANC DIST    1
                 OCCRNC PARAMS     0.100000E-05    0.0             0.0
                 DURATION DIST     1
                 DURATN PARAMS     0.100000E-03    0.0             0.0
   FAULT     2  INITIAL  STATE     0      MULTIPLICITY       500
                 OCCURANC DIST     1
                 OCCRNC PARAMS     0.100000E-06    0.0             0.0
                 DURATION DIST     1
                 DURATN PARAMS     0..100000E-03   0.0             0.0
   STVAR     1 HAS VALUE 0.100000E+01

UNIT     3 IS TYPE     2
   INPUT     1 IS LINK     2
   INPUT     2 IS   LINK   4
  OUTPUT     1 TO LINK     5 WITH TIME DELAY DISTRIBUTION     6
                 TM DLY PARAMS     0.100000E+01    0.0             0.0
   FAULT     1  INITIAL STATE      0      MULTIPLICITY         1
                 OCCURANC DIST     1
                 OCCRNC PARAMS     0.500000E-04    0.0             0.0
                 DURATION DIST     5
                 DURATN PARAMS     0.100000E-03    0.110000E+01    0.0
   STVAR     1 HAS VALUE 0.500000E+00
   STVAR     2 HAS VALUE 0.0

UNIT     4 IS TYPE     3
   INPUT     1 IS LINK     1
   INPUT     2 IS LINK     3
   INPUT     3 IS LINK     5
   STVAR     1 HAS VALUE 0.100000E+01
   STVAR     2 HAS VALUE 0.100000E+01
   STVAR     3 HAS VALUE 0.100000E+01
```

Figure A8. Model Parameters.

	COMPUTER 1			COMPUTER 2			DUAL SYSTEM		
	ANALYTIC REL.	SIMULATION REL.	TIME	ANALYTIC REL.	SIMULATION REL.	TIME	ANALYTIC REL.	SIMULATION REL.	TIME
1	99.9116	99.8000	[illegible]E+01	99.9376	99.8000	.6240E+01	99.6068	99.8000	.6459E+03
2	99.7620	99.6000	.2383E+02	99.8467	99.6000	.1534E+02	99.2938	99.6000	.8773E+03
3	99.5360	99.4000	.4651E+02	99.7711	99.4000	.2292E+02	99.2624	99.4000	[illegible]E+03
4	99.2891	99.2000	.7135E+02	99.7319	99.2000	.2684E+02	99.1654	99.2000	.9580E+03
5	99.1303	99.0000	.8735E+02	99.6936	99.0000	.3068E+02	99.0335	99.0000	.1035E+04
6	98.6874	98.8000	.1321E+03	99.6107	98.8000	.3901E+02	99.0319	98.8000	.1036E+04
7	98.2653	98.6000	.1750E+03	99.5943	98.6000	.4065E+02	98.5343	98.6000	.1290E+04
8	98.1528	98.4000	.1865E+03	99.5885	98.4000	.4123E+02	98.2179	98.4000	.1433E+04
9	97.5902	98.2000	.2439E+03	99.5059	98.2000	.4953E+02	97.9748	98.2000	.1535E+04
10	97.4875	98.0000	.2545E+03	99.3039	98.0000	.6985E+02	97.7330	98.0000	.1632E+04
11	97.2607	97.8000	.2778E+03	99.1017	97.8000	.9024E+02	97.5400	97.8000	.1706E+04
12	97.2040	97.6000	.2836E+03	99.0677	97.6000	.9366E+02	97.3519	97.6000	.1776E+04
13	97.1130	97.4000	.2930E+03	99.0482	97.4000	.9563E+02	97.1055	97.4000	.1865E+04
14	96.7612	97.2000	.3292E+03	98.9059	97.2000	.1100E+03	96.3878	97.2000	.2108E+04
15	96.6208	97.0000	.3438E+03	98.8115	97.0000	.1196E+03	95.8368	97.0000	.2282E+04
16	96.5096	96.8000	.3553E+03	98.7127	96.8000	.1296E+03	95.8053	96.8000	.2292E+04
17	96.1668	96.6000	.3909E+03	98.6831	96.6000	.1326E+03	95.3713	96.6000	[illegible]E+04
18	95.4449	96.4000	.4662E+03	98.4643	96.4000	.1548E+03	95.1818	96.4000	.2478E+04
19	95.[illegible]250	96.2000	.4683E+03	98.0775	96.2000	.1941E+03	94.9976	96.2000	.2532E+04
20	94.9808	96.0000	.5150E+03	97.9674	96.0000	.2054E+03	94.7346	96.0000	.2607E+04
21	94.8050	95.8000	.5335E+03	97.7075	95.8000	.2319E+03	94.6706	95.8000	.2625E+04
22	94.3538	95.6000	.5807E+03	97.3547	95.6000	.2681E+03	94.5853	95.6000	.2649E+04
23	94.2977	95.4000	.5871E+03	97.3117	95.4000	.2725E+03	93.8038	95.4000	.2862E+04
24	94.2454	95.2000	.5927E+03	97.2104	95.2000	.2829E+03	93.6743	95.2000	.2897E+04
25	94.1836	95.0000	.5992E+03	97.0872	95.0000	.2956E+03	93.4652	95.0000	[illegible]E+04
26	94.1298	94.8000	.6050E+03	97.0237	94.8000	.3021E+03	93.3356	94.8000	[illegible]E+04
27	94.0121	94.6000	.6175E+03	96.3466	94.6000	.3722E+03	92.9078	94.6000	.3007E+04
28	93.9614	94.4000	.6229E+03	96.2569	94.4000	.3815E+03	92.6706	94.4000	.3157E+04
29	93.7930	94.2000	.6408E+03	95.8807	94.2000	.4207E+03	92.3412	94.2000	.3240E+04
30	93.7801	94.0000	.6422E+03	95.6263	94.0000	.4472E+03	92.2548	94.0000	.3241E+04
31	93.6116	93.8000	.6602E+03	95.6188	93.8000	.4480E+03	92.0[illegible]	93.8000	[illegible]E+04
32	93.2649	93.6000	.6973E+03	95.1574	93.6000	.4764E+03	91.7360	93.6000	[illegible]E+04
33	92.9482	93.4000	.7313E+03	95.0010	93.4000	.5128E+03	91.3820	93.4000	[illegible]E+04
34	92.8052	93.2000	.7370E+03	94.9585	93.2000	.5173E+03	90.8822	93.2000	.3595E+04
35	92.3553	93.0000	.7953E+03	94.5989	93.0000	.5552E+03	90.6657	93.0000	.3646E+04
36	92.1343	92.8000	.8192E+03	94.5595	92.80	.5594E+03	90.5992	92.8000	.3662E+04
37	92.0309	92.6000	.8305E+03	94.5165	92.60	.5640E+03	90.4251	92.6000	.3702E+04
38	91.6576	92.4000	.8711E+03	94.1517	92.40	.6026E+03	90.4062	92.4000	.3707E+04
39	91.6425	92.2000	.8728E+03	94.0478	92.20	.6137E+03	90.2641	92.2000	[illegible]E+04
40	91.5965	92.0000	.8778E+03	93.8973	92.0000	.6297E+03	90.0187	92.0000	.3797E+04
⋮ etc.	⋮ etc.	⋮ etc.	⋮ etc.	⋮ etc.	⋮ etc.	⋮ etc.	⋮ etc.	⋮ etc.	⋮ etc.

Figure A9. Sample Simulation Output

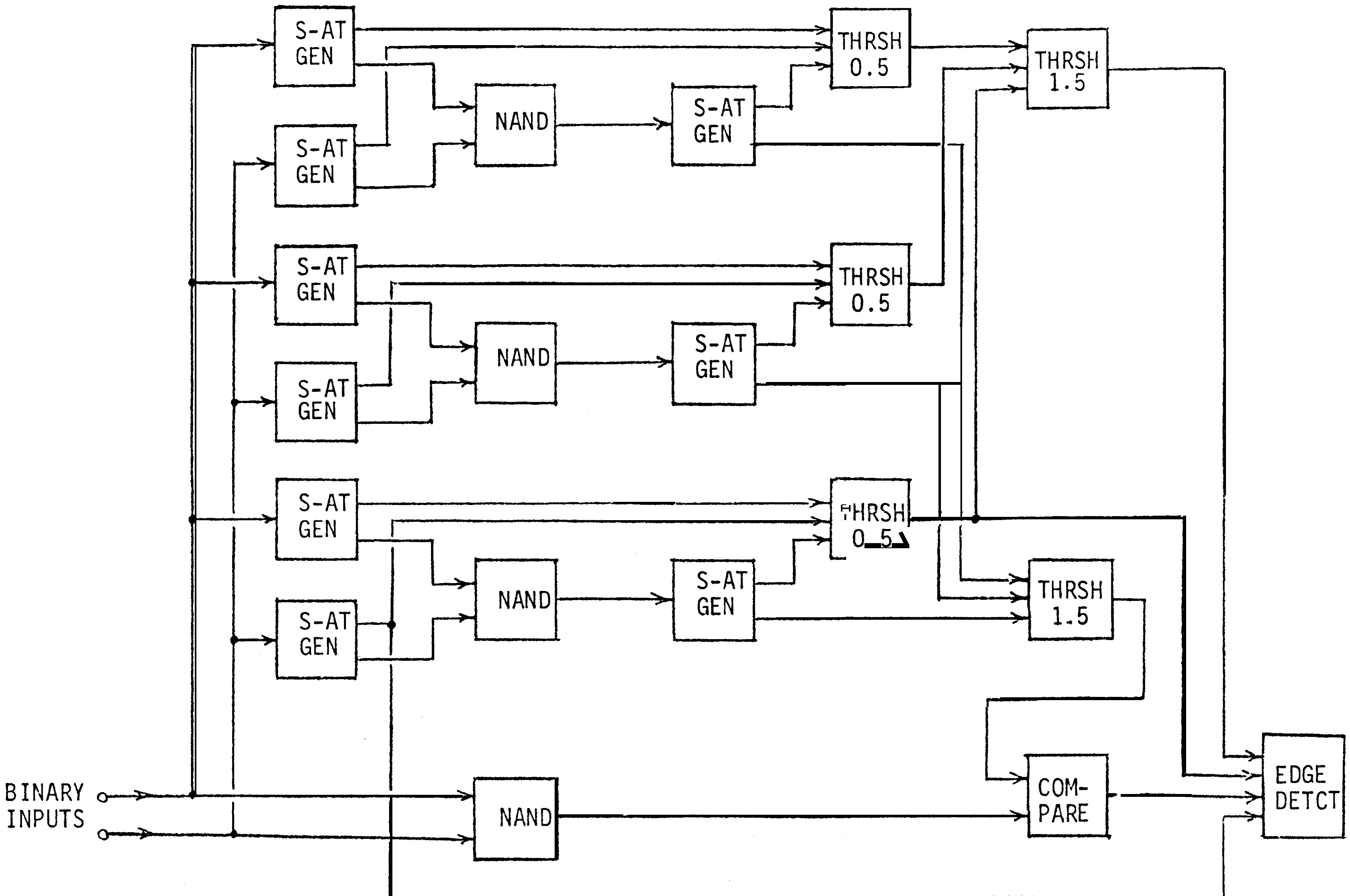


Figure A10. Model for Study of a TMR NAND-Gate Circuit.

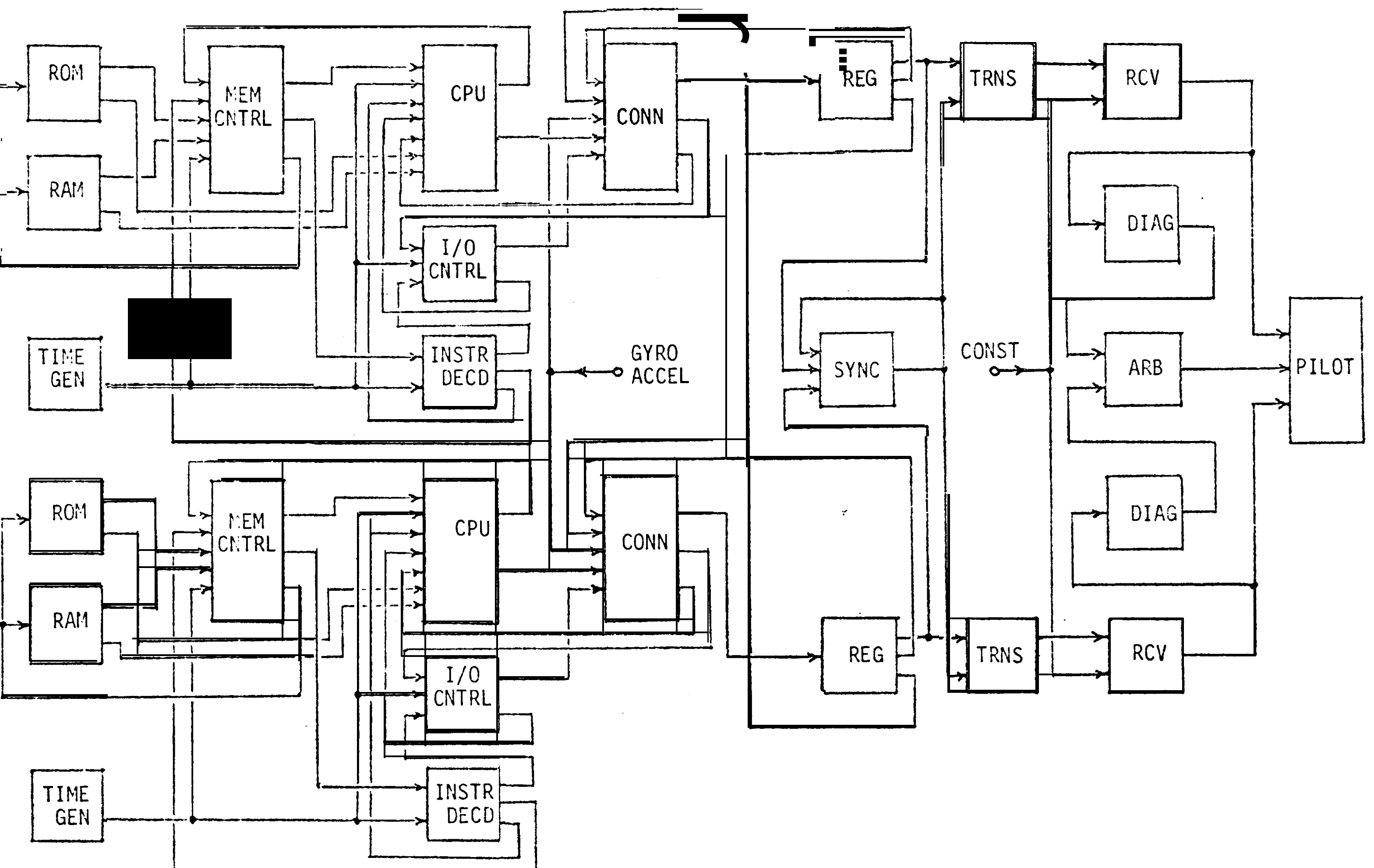


Figure A11. Model of a Dual Computer System for Navigation.

8. APPENDIX B - RELIABILITY ANALYSIS

This appendix derives the analytic reliability model for systems having faults with constant failure rates. The basic model is then extended to obtain the reliability equations used in Examples a and b of Section 4. In the following discussion, all faults are assumed permanent.

Consider a system with one possible fault, where the system fails at the same time that fault becomes active. The time T at which the system fails is a random variable. The reliability, R(t), of the system is a function of the random variable T.

R(t) = probability that the system has not failed before time t, i.e., prob(T>t).

By convention, we always assume that the system is not failed at time t=O, therefore R(0) = 1.

Let the fault have a constant rate of occurrence, expressed as A failures/hours. At time t(hrs.), the failure rate must equal the probability of a failure at that instant, given that the failure has not occurred before time t. The probability of failure at that instant is the negative of the derivative of R(•) evaluated at t. Using Bayes Theorem, the conditional requires only division by R(t). Thus:

$$\lambda = \frac{-R'(t)}{R(t)} \quad \forall\, t>0.$$

With boundary condition $R(0)=1$, this differential equation is directly solved:

$$R(t) = e^{-\lambda t}, \ t > 0.$$

A system with a reliability function of this form is said to have an exponentially distributed fault. All one-fault systems with a constant failure rate λ have this reliability function.

Example a of Section 4 considers a dual computer system, with faults occurring independently in each computer. For the simple case, each computer had only one fault with a constant failure rate λ, and the dual system failed only when both failed. Let $R_s(t)$ be the dual system reliability, and $R_1(t)$ and $R_2(t)$ be the reliabilities of each computer alone. Then we have

$$R_1(t) = R_2(t) = e^{-\lambda t}, \ t > 0.$$

The probability that one computer fails before time t is $1-R_1(t)$, and since they are independent the probability that the dual system fails before time t is

$$1-R_s(t) = [1-R_1(t)]\cdot[1-R_2(t)],$$

so that $$R_s(t) = 2R_1(t) - R:(t),$$

or $$R_s(t) = 2e^{-\lambda t} - e^{-2\lambda t}, \ t > 0.$$

This is the reliability function for the simple case of the dual computer system.

Consider now the case of the TMR NAND-gate circuit. Each gate has three leads (two inputs,one output), and each lead has a

constant failure rate λ. The reliability of one lead is

$$R_{\ell}(t) = e^{-\lambda t},\ t > 0.$$

A gate functions properly only if all three leads function properly. Since the lead failures are independent, we may multiply their probabilities of being fault-free to get the gate reliability:

$$R_G(t) = R_{\ell}^3(t) = e^{-3\lambda t},\ t > 0.$$

A lower bound for the system reliability, $R_s(t)$, can be derived by assuming the final voter output is correct if and only if at least two gates are fault free. By "correct" is meant the TMR circuit performs the same logic function as a single fault-free gate for all input combinations. There are three ways in which two gates can fail while one is fault-free, and one way for all three gates to fail. The probability of system failure before time t is

$$1 - R_s(t) = 3R_G(t)\left[1-R_G(t)\right]^2 + \left[1-R_G(t)\right]^3,$$

so that $R_s(t) = 3R_G^2(t) - 2R_G^3(t)$,

or $R_s(t) = 3e^{-6Xt} - 2e^{-9\lambda t},\ t > 0.$

This is only a lower bound for the true reliability function of the TMR NAND-gate system, because the analysis does not take into account compensating failures (discussed in Section 4).

9. REFERENCES

[Beaufils, et al, 1974] Beaufils, R., J.L. Paul and R. Troy, "Systems d'Evaluation globale de Multiprocesseurs autoréparables", Repport 1 and 2, Contract DRME 73/070, LAAS, University of Toulouse, Toulouse, France, 1974.

[Conn, 1974] Conn, R.B., "Definition and Trade-Off Study of Reconfigurable Airborne Digital Computer System Organizations- Final Report", NASA Contract NAS1-12793, Ultrasystems, Inc., Newport Beach, Ca., Nov. 1974.

[Fishman, 1973] Fishman, G.S., "Concepts and Methods in Discrete Event Digital Simulation", John Wiley & Sons, N.Y., 1973, pp. 98-135.

[Masreliez & Bjurman, 1976] Masreliez, C.J., and B.E. Bjurman, "Fault-Tolerant System Reliability Modeling/ Analysis", Boeing Corp., Seattle, Wn., June, 1976.

[Raytheon, 1974] Raytheon Company, Equipment Development Laboratory', "Reliability Model Derivation of a Fault-Tolerant, Dual, Spare-Switching, Digital Computer System", Final Report, NASA Contract NAS1-12668, Sudbury, Mass., March, 1974.

[Ressler, 1973] Ressler, B.E., 'Design of a Dual Computer Configuration for Redundant Computation', M.S. Thesis, M.I.T., June, 1973.

[Siewiorek, 1971] Siewiorek, D.P., "An Improved Reliability Model for NMR", Technical Report No. 24, Digital Systems Lab., Stanford University, Stanford, Ca., December, 1971.

[Thompson, 1977A] Thompson, P.A., "Using Simulation to Evaluate the Reliability of a Dual Computer System," Technical Report No. 121, Digital Systems Lab., Stanford University, Stanford, Ca., March 1977.

[Thompson, 1977B] Thompson, P.A., "Manual for a General Purpose Simulator Used to Evaluate Reliability of Digital Systems," Technical Report No. 132, Digital Systems Laboratory, Stanford University, Stanford, California, March 1977.