NASA
Technical
Paper
2738

September 1987

A Technique for Evaluating the Application of the Pin-Level Stuck-At Fault Model to VLSI Circuits

Daniel L. Palumbo
and George B. Finelli

NASA
Technical
Paper
2738

1987

A Technique for Evaluating the Application of the Pin-Level Stuck-At Fault Model to VLSI Circuits

Daniel L. Palumbo
and George B. Finelli

Langley Research Center
Hampton, Virginia

National Aeronautics
and Space Administration

**Scientific and Technical
Information Office**

Identification of commercial products in this report is included only to adequately describe the equipment and does not constitute an official endorsement, either expressed or implied, of such products by the National Aeronautics and Space Administration.

Contents

Summary

This paper describes a technique by which a researcher can evaluate the capability of the pin-level stuck-at fault model to represent true error behavior in very large scale integrated (VLSI) digital circuits. Accurate fault models are required to conduct the experimentation recommended by an earlier study of proposed validation methodologies for highly reliable fault-tolerant computers (e.g., computers with a probability of failure of 10^{-9} for a 10-hour mission). The validation experiments are designed to measure the recovery process parameters of fault-tolerant computers.

The quality of the pin-level stuck-at fault model is assessed by comparison of the error behavior which results from faults applied at the pins of a VLSI circuit with the error behavior produced by faults applied to gates internal to the VLSI circuit. The internal, gate-level faults are assumed to produce "true" error behavior. Error behavior is observed at the pins of the VLSI circuit, in this case a "virtual microprocessor." In the presence of internal faults, the error behavior at the pins of the virtual microprocessor is more dynamic than static.

To study the dynamic error behavior, a hypothesis is put forth that pin-level stuck-at faults produce error behavior that is similar to the error behavior caused by internal device failures. The hypothesis is tested with Wilcoxon's rank-sum test. It is found that this technique is tractable and has the potential to produce meaningful results. Using a sample data set, a set of bar charts are derived which show the tendency to reject the hypothesis. Many of the pin-level faults exhibit very little rejection. However, in some test cases, the results strongly suggest rejection, especially in the modeling of the duration of errors. A firm conclusion cannot be drawn because of the preliminary nature of the sample data. The results do imply that the application of the pin-level stuck-at fault model requires careful consideration. Additional experimentation is needed to confirm the use of the model before it can be used with confidence when validating highly reliable digital systems.

Introduction

The ability to simulate faults in digital circuitry is an important issue in a proposed methodology for validating the failure recovery process in fault-tolerant computers (ref. 1). A common method of simulating a fault in a physical circuit is fault injection on the pins of the circuit components (ref. 2). Typically, the injected fault takes the form of a "stuck-at" fault (ref. 3). The accuracy with which an injected fault models its physical counterpart is important if confidence in the validation process is to be established. When high reliability is specified, such as a probability of failure of 10^{-9} for a 10-hour mission, high confidence levels and, therefore, accurate fault models are required. Reference 1 cautions that the pin-level stuck-at fault model may not be adequate for circuits composed of very large scale integrated (VLSI) components. A technique by which the capability of the pin-level stuck-at fault modeling can be evaluated is the subject of this paper.

As background, the fault recovery process is defined and validation methods for the recovery process are described. The objective and approach of this work are then stated, followed by a detailed description of the experiment, the results, and conclusions.

Background

Fault-tolerant computers which are based solely on redundant hardware and voting cannot meet high-reliability requirements unless the failure rates currently obtainable are reduced by an order of magnitude (ref. 4). Today's highly reliable fault-tolerant system must incorporate a recovery process to remove and, possibly, replace faulty components. There are two important aspects of the recovery process of a fault-tolerant system that can be exercised by fault injection: the ability to detect the fault and the ability to reconfigure successfully after the fault has been detected.

The detection process. The presence of a fault in a system is detected when erroneous system behavior is recognized by a system monitor (voter, watchdog timer, etc.). However, the existence of a fault does not guarantee that the system will exhibit erroneous behavior. The errors produced by the faulty component must be propagated to the system monitor level.

Two additional factors complicate fault detection: different components may produce similar error behavior; and detected errors may be transient, that is, not the result of hard faults. When presented with ambiguous error behavior, isolation procedures (such as a specific series of tests) locate the faulty component. Verification procedures filter out transient errors by logging the errors until, based on some heuristic, a hard fault is declared. In this study, isolation and verification are considered part of the detection process. Both isolation and verification require extra data, and therefore time, before the recovery process can begin a reconfiguration.

The detection process can thus be divided into three tasks: error propagation, fault isolation, and fault-type verification. The time spent performing each of these tasks depends a great deal on which

component is faulty in the circuit. Yet these times are more directly dependent on the error behavior observed by the monitor. The error behavior is not solely a product of hardware characteristics but depends to a large degree on the input presented to the hardware (e.g., software).

Reconfiguration. A successful recovery results in the logical restructuring of the system around the fault, that is, reconfiguration. Previous studies have shown that the reconfiguration time is not as dependent on the fault present in the circuit as is the detection time (ref. 2).

Reconfiguration can be accomplished in hardware or software. With hardware reconfiguration, extra switching circuitry is added to enable the removal and replacement of elements which contain faulty components. The switching circuitry can be installed at different levels. The Fault-Tolerant Multiprocessor (FTMP), for example, reconfigures at the processor-memory-bus interfaces (ref. 4). The Fault-Tolerant Processor (FTP) switches entire computers into and out of the computer interstage (ref. 5).

Software reconfiguration is accomplished by rewriting system data structures which control communication and scheduling procedures. The faulty processors remain connected to the intercomputer network, but their outputs are ignored during communication and voting and tasks are not allocated to them during scheduling. A good example of this paradigm is the Software Implemented Fault Tolerance (SIFT) computer (ref. 6).

The validation methodology. Two parameters can be used to characterize the recovery process. They are the percentage of faults for which the recovery process performs the correct action (in the presence of the fault and its resulting errors) and the time taken to complete the process. The first parameter will be referred to as the coverage of the recovery process; the second will be referred to as the total recovery time.

In an effort to define a validation methodology which could determine whether a candidate fault-tolerant computer meets the requirement for life-critical digital avionics (i.e., with a probability of catastrophic failure of 10^{-9} for a 10-hour mission), a previous study has suggested that recovery parameters can be obtained by applying pin-level stuck-at faults to the fault-tolerant computer during carefully controlled experiments (ref. 1). The parameters are then inserted into models of the failure-recovery process (Markov models or Markov model derivatives) to calculate the probability of failure. One such model is described in appendix A.

With the assumption that the processor failure rate can be obtained from sources such as MIL-HDBK 217D (ref. 7), all that remains to complete the model is to derive a value for the recovery rate. As mentioned above, it has been suggested that the recovery rate be obtained through experimental measurements.

The recovery rate measurement must be accurate enough to establish confidence in the computation of the probability of failure. As explained previously, the detection time is a part of the recovery time and depends on the error behavior generated by the fault. Therefore, the degree of experimental accuracy will depend on how well the fault injection stimulus reproduces true erroneous behavior in the target system.

Injecting a fault at the pin level simulates a condition in which the error has already propagated from the lower-level device. Because of this, pin-level fault injections are not suitable for measuring the error propagation time component of the detection process. Thus, it is assumed that pin-level fault injections will be used primarily in experiments designed to measure the remaining time components of the detection process. These processes (i.e., the fault isolation and verification processes) have error behavior symptoms as their input. The fault-injection model must produce error symptoms similar to true error behavior to be effective. To determine the applicability of the pin-level fault model, the technique described in this paper produces a measure of the similarity between true error behavior and error behavior created by fault injection. To demonstrate the technique, it is assumed that if dissimilar error behavior is observed on fewer than 10 percent of the pins of the VLSI circuit under test, then acceptable experimental accuracy will result. The 10-percent limit is inferred from the discussion in appendix A, which concludes that it is necessary to measure experimental parameters to within an order of magnitude of the correct value.

Objective and Approach

The objective of this work is to define a technique by which insight can be gained into the ability of the pin-level stuck-at fault model to simulate true error behavior in VLSI digital circuits. To this end, the error behavior caused by pin-level stuck-at faults will be compared with true erroneous behavior in VLSI digital circuits. The stuck-at fault model has been accepted as a good model of gate-level fault behavior (ref. 3). Transferring the stuck-at model to the pins of an integrated circuit presents little difficulty for small-scale integrated (SSI) circuits and for some medium-scale integrated (MSI) circuits because of

the proximity of the gates to the pins. However, the pin-level stuck-at model may not apply well to VLSI circuits. With a VLSI circuit, the pins of the integrated circuit are more at the level of the system monitor than at the level of the faulty device (consider the self-checking dual processor in fig. 1). At the pins of a VLSI circuit, the objective is to model error behavior rather than fault behavior.

To compare the error behavior generated with pin-level stuck-at faults to true error behavior, it is necessary to obtain samples of both. Obtaining samples of a VLSI chip error behavior when subjected to pin-level faults is straightforward. Acquiring the response of the same chip to internal faults is more difficult and requires some type of simulation of the chip. The approach used in this experiment was to create a "virtual microprocessor" from an existing SSI-MSI processor by drawing a boundary around the components that would be found inside a chip if indeed the processor were a chip. The resulting virtual microprocessors (Vmp) consists of 48 pins of data, address, and control. The Vmp allows faults to be injected on its internal devices while its behavior is recorded at its boundaries.

Once samples of the error behavior are obtained, two types of analysis are performed. First, the tendency towards static error behavior is derived. Static error behavior is defined as a condition where a pin does not change state because of a fault within the circuit. This condition closely resembles a pin-level stuck-at fault. The second analysis examines the dynamic error behavior of the pins (i.e., errors are present and the pin is changing state). The dynamic error behavior of internal and pin-level faults are compared by formulating the hypothesis that their respective data samples are taken from the same general error behavior distribution. If this hypothesis is rejected, then it will be concluded that pin-level stuck-at faults are "not very good" at simulating true error behavior in VLSI circuits.

Experiment Definition

The definition of the experiment is divided into three sections. The first section describes the data that were gathered and the analysis that was performed. The second section defines assumptions made which support the experiment approach and analysis. The third section contains the details of the experiment configuration and procedure and is found in appendix B.

Experiment Data and Analysis

To produce data descriptive of error behavior, a faulted system is compared with a fault-free version of the system. In this investigation, two types of error behavior are analyzed, static and dynamic.

Static error behavior. If a pin does not change state during a test, it is said to exhibit static behavior. If the test is conducted in a fault-free condition, then this is normal static behavior. If a pin that was not normally static becomes static during a fault injection test, or if a pin that was normally static remains static but in an inverted state, the pin is said to exhibit static error behavior. A pin-level stuck-at fault of the correct polarity is indiscernible from a pin exhibiting static error behavior.

One evaluator of static error behavior is the percentage of pins of the Vmp that exhibit static error behavior during faulted runs. A large percentage of pins exhibiting static error behavior would indicate that the pin-level stuck-at model has some justification for VLSI circuits. However, a small percentage of pins exhibiting static error behavior is not sufficient evidence to conclude that the pin-level stuck-at model is inaccurate. The dynamic error behavior must be considered also.

Dynamic error behavior. Dynamic error behavior is defined by two values, the time between errors and the duration of errors. Figure 2 illustrates how these values are derived. The first trace in figure 2 is of a Vmp pin during a fault-free test. The second trace is of the same pin, but now the effect of an internal fault is manifested as different behavior. The third trace is derived as the exclusive-or of the first two traces and represents the error behavior of the second trace. In other words, when the error trace is "high" the value of the pin during the faulted run is opposite that of the fault-free run. The time between errors (TBE) is defined as the time elapsed between rising edges of error pulses. The duration of errors (DE) is the length of the error pulses.

A pin can exhibit both static and dynamic error behavior. If an induced stuck-at fault is applied to a pin which behaves dynamically during the fault-free run, it will behave statically (i.e., it will not change state during the test). However, from the definition of dynamic errors, TBE and DE characteristics can be observed.

Any single internal fault which exhibits dynamic error behavior will produce samples of TBE and DE. The TBE and DE data can be thought of as samples from an error behavior population. Thus, both the pin-level stuck-at faults and the internal faults can be associated with general, but possibly different, error behavior populations. If the error behavior population associated with pin-level stuck-at faults can be shown to be different from the error behavior population of the internal faults, one could conclude

that the pin-level stuck-at fault does not produce error behavior similar to that produced by internal faults.

Statistical hypothesis testing can be used to determine whether two samples were taken from the same population. The strength of hypothesis testing is in rejecting the hypothesis; this is called a significant result. If the hypothesis is not rejected, all that can be said is that for this particular case the null hypothesis holds. If the hypothesis of identical populations is rejected, it can be concluded that the null hypothesis does not hold for this particular case and, therefore, does not hold in general.

Wilcoxon's rank-sum test. Wilcoxon's rank-sum test (also known as the Mann-Whitney test) can be used to test the hypothesis of identical populations (ref. 8). Given two data samples, x_i and y_j, the rank-sum method tests for a shift in the samples which would indicate that the samples came from different populations. In our case, for example, the x_i are samples of error behavior taken during internal fault tests and the y_j are samples taken from pin-level fault tests. A test statistic w is derived for x_i and y_j by summing the ranks assigned to the members of each sample after x_i and y_j are merged and ordered. A function W is calculated by constructing the distribution of all possible orderings of two samples. Values for W are found in textbooks. (See ref. 8, table A.5 for an example.) The decision rule for the test can be stated as follows. Reject the null hypothesis (H_0) at level of significance a if the test statistic w is greater than the $(1-a)/2$ quantile of W. For example, if a significance level of 0.95 is desired and the two samples have sizes of 3 and 6, the $(1-a)/2$ quantile of W is 22. (See table A.5 of ref. 8.) If $w > 23$, then H_0 is rejected. The IMSL subroutine NRWST is used to perform this test on the data (ref. 9).

Assumptions

The assumptions stated in the following sections describe models of behavior which together comprise the foundation for the experimental approach and analysis. Two models are used during the experimentation. One simulates VLSI circuit function and the other fault behavior. A third model describes the characteristics of the data to which the Wilcoxon test are applied. A final model defines what is meant by an error.

VLSI circuit model: the virtual microprocessor. As previously stated, the purpose of the experimentation is to obtain the error behavior characteristics, that is, samples of time between error (TBE) and duration of error (DE) at the pins of a VLSI circuit. For the best results, the errors should be produced by the types of faults which would occur naturally within the VLSI circuit. Except for the case wherein hundreds of VLSI circuits are available for destructive fault-injection testing, the VLSI circuit must be simulated at a level consistent with the fault model that will be applied to the circuit.

A VLSI circuit simulation was created by considering a Bendix BDX-930 processor as a 48-pin virtual microprocessor (Vmp). (See section entitled *The Vmp* in appendix B.) The BDX-930 is a 16-bit processor constructed mostly of bipolar SSI and MSI logic. The 48 pins of the Vmp are mapped to 48 signals within the BDX-930 processor. The 48 signals correspond to 16 memory address lines (the MAR bus), 16 data lines (the DAT bus), and 16 miscellaneous signals.

Because the circuit simulation is a processor, the simulation must be executing a program to produce results and, therefore, errors. The program chosen produces a Fibonacci series. (See section entitled *The Fibonacci Series* in appendix B.)

The following assumptions support the use of the Vmp as a VLSI circuit simulator.

1. *The BDX-930 is a modern processor of adequate complexity so that if it were produced as a single chip, it would be considered to be a VLSI device.*

In support of this assumption, consider that the BDX-930 has a microprogrammed pipeline architecture and contains 5000 to 6000 gates. Although the BDX-930 is not a state-of-the-art processor and the gate count is less than the 10 000 usually attributed to VLSI circuits, the authors believe this to be a fair assumption.

2. *The Fibonacci series program represents typical program execution.*

Fibonacci series calculation has been used in past studies (refs. 10 and 11) to represent typical program behavior. The program used in this test was written to reflect as wide an instruction mix as possible. (See appendix B.) If the data gathered with the Fibonacci program cause the H_0 hypothesis to be rejected, it can be expected that H_0 will be rejected for most programs. This is believed to be true because of the simplicity and directness of the Fibonacci program. However, the converse is definitely not true. If H_0 is not rejected, very little can be said about the behavior expected from other programs. (This assumption may be weak. See section entitled *The Fibonacci Series* for a discussion.)

3. *The Vmp allows sufficient access to its internal devices for fault injection.*

The Vmp contains approximately 100 integrated circuits, most with either 14 or 16 pins. If the pins of the integrated circuits are considered to be connected to devices that would actually be inside the Vmp, then there is access to about 30 percent of the devices internal to the Vmp (1800 pins out of an estimated 5400 gates). The implication of the assumption is that access to this subset of devices is adequate if H_0 is rejected. During actual testing, a small sample was taken from this subset.

Fault model: the stuck-at. The stuck-at fault model is derived from the tendency of a switch to fail either stuck open or stuck closed (a transistor in a digital circuit acts as a switch). Other failure modes can be modeled as special cases of stuck-ats. For example, an intermittent contact can be modeled as a recurring, short-term stuck open failure. The emphasis here is placed on the binary nature of the device and the fact that the device has a tendency to fail to one state or the other.

As mentioned above, there is access to approximately 1800 devices within the Vmp. The stuck-at fault model was chosen to represent the failure modes of those devices. The decision to use the stuck-at model is based on the following assumptions.

4. *The accessible devices of the Vmp are logic gates.*

5. *The stuck-at fault model is adequate for gate-level injection.*

It may at first seem unwise to test the validity of the pin-level stuck-at fault model under an assumption that the stuck-at model is adequate. However, the important issue is the scope of the stuck-at model. Given that the stuck-at model faithfully represents the failure modes of a logic gate, the question is whether the model can be extended to cover the error behavior at the boundary of a large circuit of gates.

Data model. The Wilcoxon test compares two data samples to test the hypothesis that the samples were taken from the same population. One set of samples, taken as the true error behavior, is acquired from the 48 pins of the Vmp with internal stuck-at faults. The second set of samples is taken with one or more pins of the Vmp stuck at zero or stuck at one. The true error behavior samples are compared with samples for both pin-level stuck at one and stuck at zero to test whether they came from the same general error behavior population.

Given that two data sets are obtained with m samples in x and n samples in y (x being the true error behavior and y being the pin-level stuck-at behavior), application of the Wilcoxon test to x and y implies the following assumptions.

6. $x_i = e_i$ $(i = 1, 2, \ldots, m)$ *and* $y_j = e_j + D$ $(j = m + 1, m + 2, \ldots, m + n)$ *where* x *and* y *are observable,* e_j *are unobservable, and* D *is an unknown shift in* y.

7. *The* N *e-values* $(N = m + n)$ *are mutually independent.*

8. *The e-values are sampled from the same continuous population.*

With respect to this model, the hypotheses of the Wilcoxon test can be stated as

$$H_0 : D = 0$$
$$H_1 : D \neq 0$$

Assumption 6 proposes that x and y differ by a constant offset D. If H_0 is rejected (i.e., it is found that $D = 0$), then a shift will be found in the data and we will know that x and y were not taken from the same population.

Assumption 7 may be used with confidence. Precautions were taken to ensure that the system was reloaded and initialized to the same state prior to each fault-injection test. (See section entitled *The Experiment Procedure* in appendix B.) However, attempts for a complete system initialization may have fallen short. See section entitled *The Fibonacci Series* in appendix B for a discussion.

The error model. To correctly interpret an analysis of error behavior, it is necessary to know what is meant by an error. Consider the block diagram in figure 1. The system described by the block diagram can be considered to be a tightly coupled self-checking dual processor. The system monitor (comparator) compares the pin-level state of processor A to that of processor B. If either processor deviates from the other, the monitor signals an error.

The block diagram can also be used to describe the error analysis procedure used in this study. For example, let the block labeled processor A represent data acquired during the fault-free run and the block labeled processor B be data acquired during fault-injection tests. Thus, the system monitor becomes the analysis procedure which steps through the data from the fault-free and faulted files and signals an error upon miscomparison. However, the process of comparing these two files is complicated by two factors. First, the processor does not gate data onto its buses every clock cycle. Between bus transactions the state of the bus is undefined, and therefore a comparison is meaningless. Second, the faulted processor, given that it might follow a different instruction path, might not gate data onto the bus on the same cycle as in the fault-free run. The following two assumptions serve to eliminate this ambiguity.

9. *The state of the pins is observed when the fault-free run has data enabled onto its buses.*

10. *An error has occurred when the state of a pin in a faulted run is observed to differ from the state of the corresponding pin in the fault-free run.*

The scope of the error analysis has been limited to the behavior of the memory and data buses.

Results

A total of 136 fault injections were applied to signals both internal to (112) and at the pin-level boundary of (24) the Vmp. The faulted behavior of the Vmp was observed at the pin-level boundary and recorded in what will be referred to as the "results" files. (See appendix B.) The 68 signals to which the stuck-at one and stuck-at zero faults were applied were chosen from a signal list of the BDX-930. Care was taken to ensure that injections were applied to each chip in the BDX-930. Of the 136 fault injections, 92 produced error behavior. The following sections present the static and dynamic analysis of these errors.

Results of Static Error Analysis

As explained in the section entitled *Static Error Behavior*, there are two kinds of static pin behavior: pins which are normally static in the fault-free run and pins which become static due to the presence of a fault. A total of 15 pins were found to be normally static in the fault-free run.

A total of 92 faults which were applied to the Vmp produced errors. During each fault, all 48 pins of the Vmp were tested, yielding 4416 pin-tests (48 pins * 92 test). Of the 92 faults, 62 created static error behavior among 1 or more of the 48 pins. In 4416 pin-tests, only 204 pins (≈ 5 percent) demonstrated static error behavior. However, 219 normally static pin-tests became nonstatic. The application of stuck-at faults actually decreased static behavior. This finding, while not totally unexpected, leads to the analysis of the dynamic error behavior.

Results of Dynamic Error Analysis

To perform the dynamic error analysis, the 48 pins of the Vmp are broken down into 3 groups of 16: the MAR bus, the DAT bus, and the signals. Each pin-test yielded time-between-errors (TBE) and duration-of-errors (DE) data. These data, which quantify dynamic error behavior, are derived from an analysis that is based on the definition of an error as stated in assumptions 9 and 10. A program was written which cycled through the fault-free and results files as if they were a tightly synchronized self-checking pair. However the fault-free run was the master. Values from the fault-free and results files were compared only when a bus transaction was initiated in the fault-free run. The analysis presented will be limited to the MAR and DAT groups because the two buses represent general behavior rather than unique functionality, represented by the signals.

Figures 3 to 6 are histograms of the TBE and DE for the MAR bus and the DAT bus. Each histogram shows data summed over all 16 pins in a group for all 92 fault-injection tests which produced error behavior, thus representing 1472 pin-tests. The number of data points in each histogram depends on the error activity; that is, more samples of TBE and DE are obtained if, on average, the TBE and DE are small compared with the sample period. Figures 3 and 4 show the number of errors that produced TBE's ranging from 1 to 350 cycles (1 cycle is 125 ns); figures 5 and 6 show similar information for the DE data. For example, in figure 3 a peak of 1900 errors occur with a TBE of approximately 15 cycles.

If the data are further broken down according to the type of operation (input, output, or instruction fetch) taking place on the bus at the time of the error, different error behavior can occur. Input and instruction fetch operations were found to be similar to each other and to the combined TBE data as plotted in figure 4. Output operations have a different characteristic, as shown in figure 7. The plots of the TBE during output operations demonstrate the type of error behavior that a monitor which votes output data will encounter.

The data presented so far illustrate gross error behavior in that the samples taken on all pins for all fault-injection tests were summed. These data cannot be subjected to a hypothesis test without additional assumptions which state that the individual samples are from the same population and, therefore, can be summed. Figure 8 is an example of data from a single internal fault-injection test. The TBE samples for each pin on the DAT bus are represented in "box plot" format.

In the box plot format, the width of the box is proportional to the number of points in each sample. The lower perimeter of the box marks the 25th percentile; the upper perimeter is the 75th percentile. The line through the box is the median. The bars off the ends of each box mark the farthest points within the fences, which are 1.5 box lengths below the 25th and above the 75th percentiles. The crosses mark points outside the fences (called outliers). Finally, the little squares within the x's in them are the means. As can be seen, the box plot format condenses the error behavior of an entire group into a single plot without loss of information.

Comparison of Internal and Pin-Level Fault Behavior

The error behavior of the system in response to internal faults is compared with the error behavior in response to pin-level faults. In the case of the Vmp, pin-level faults are considered to be those faults which occur at pins located at the boundary of the Vmp (more specifically, the DAT and MAR buses). Figure 9 is the box plot of the TBE on the DAT bus which occurred in response to a stuck-at one fault applied to the pin-level signal DAT00.

Comparison of figures 8 and 9 shows that pin-level data (fig. 9) can produce error behavior similar to that which occurs during internal faults (fig. 8). This is not always the case. Figure 10 is an example of different error behavior caused by an internal fault. The Wilcoxon rank-sum test is used to obtain a measure of how well the pin-level fault data match the internal fault data. For each pin, two samples are compared, and the hypothesis of identical populations is tested. For any single fault-injection test, 32 hypothesis tests are performed on both the TBE and DE data. (There are 16 pins each on the DAT and MAR buses.) A summary measure of the ability of the pin-level fault to model the internal fault is presumed to be the percentage of pins which reject the hypothesis of identical populations. A large percentage of rejection indicates a small likelihood that the internal faults are modeled well.

Figures 11 to 20 are bar charts summarizing the results of the hypothesis testing for the TBE and DE data which resulted from stuck-ats applied to signals DAT00, DAT07, DAT15, MAR00, and MAR07. The results of stuck-at one and stuck-at zero faults were found to be similar and therefore are combined. A total of 224 tests are possible, resulting from a stuck-at one pin-level fault versus 112 internal faults plus a stuck-at zero on the same pin versus 112 internal faults. Because some of the internal faults produced insufficient data for a test, the bar charts display the results of about 150 tests. These displays can be best described by considering a specific example.

Figure 11 shows the results of comparing the stuck-at faults injected on DAT00 to all internal faults. The first bar of figure 11 represents the number of tests which rejected the hypothesis on less than 10 percent of the pins. Remember, there are 16 data bus pins and 16 memory address bus pins. According to figure 11, about 115 of the tests resulted in rejection of the null hypothesis on 0 to 10 percent of the 32 pins.

All the TBE results are alike, with most falling into the 10-percent rejection region. The DE results are more weighted at 40 percent and above than the TBE results. A noteworthy example of this is the DAT07 DE graphs (fig. 17).

Discussion

The dynamic error behavior resulting from internal and pin-level stuck-at faults has been characterized by the time between errors (TBE) and the duration of errors (DE). The similarity of the two sample types (pin-level vs internal faults) was tested with Wilcoxon's rank-sum test. It has been found that single, pin-level stuck-at faults produce dynamic error behavior characteristics which are similar to single internal faults in many cases. A closer look into the underlying mechanism which causes the error behavior provides a plausible explanation for this similarity.

Two types of aberrant processor behavior are defined. A crash is said to occur when the processor jumps outside the program limits. Skewing occurs when one or more instructions use one or more extra clock cycles to execute than normal. When skewing occurs, even though the processor may follow normal program flow, error behavior similar to a crash may be observed. This is because the comparison of the two runs (faulted and fault free) which produces the errors is synchronized by the fault-free run and, therefore, a single clock cycle of skew in the faulted run produces a great deal of "crashlike" errors.

Out of the 80 internal faults which produced errors, 34 caused a crash. Of the 46 remaining noncrashed runs, 30 were skewed. Thus, 80 percent of the internal faults either crashed or were skewed and are therefore likely to exhibit the same error behavior. Of the 10 pin-level faults tested which produced errors, 100 percent either crashed or were skewed. This puts all the tests of pin-level faults into the same behavioral category as a large majority of internal faults.

Another interesting observation is that the TBE samples are more likely to be similar than the DE samples. This may be due to a correlation between the TBE and the average instruction execution time. The TBE histograms (figs. 3 and 4) show that most TBE's lie between 10 and 20 clock cycles (125 ns per cycle). This matches well the average instruction execution time of the Vmp of 1 to 2 μs and implies that once an error goes away it is likely to return in one or two instructions. The DE histogram in figure 5 does not exhibit this characteristic. The histogram peaks at 1 or 2 cycles and drops to zero at 30 cycles.

Besides the suggested correlation of TBE to instruction execution times, the similarity in behavior of TBE sample is related to three of the assumptions.

Assumption 2 states that the Fibonacci program represents typical program behavior. This is true until the processor crashes. From that point on, the behavior is unrelated to the Fibonacci program but to some pattern in a large block of undefined memory. Consistent error behavior may be related to the presence of a large block of undefined memory through one of two mechanisms. The randomness of the undefined memory may yield consistent error behavior. Given a completely random memory pattern, similar behavior would result no matter where the processor executed in memory. Alternatively, after a few fault injections, the undefined block of memory may tend toward the same constant pattern. If, for example, a crash tends to write zeros in memory (which are no-operation instructions in the Vmp), then, after a few crashes, the once undefined block of memory is now well defined as a long string of no-operation instructions. This mechanism violates the independence assumption. If the test program had been larger, fewer jumps outside its range could have occurred. If the program resided in read only memory, modification of executable code could not occur. These additional factors might have produced entirely different behavior than that observed with the Fibonacci program.

Assumption 9 states that the comparison of the test and fault-free files is to be synchronized with the execution of the fault-free file. Thus, an error cannot occur unless the fault-free run initiates a bus transaction. This type of analysis selectively deletes a great deal of error behavior. Performing a comparison whenever either of the two runs initiated a transaction may have produced different error behavior than that observed.

The end result of this abundance of similar error behavior is that the test hypothesis of identical populations is not overwhelmingly rejected. With the DAT00 data used as an example and the assumption of a desired rejection limit of 10 percent or less, consider that less than 25 percent of the fault-injection tests analyzed for TBE exceed the 10-percent limit. Analysis of the DE data, often a better discriminator, results in about 55 percent of the fault-injection tests exceeding the 10-percent limit. The pin-level fault model might be rejected based on the poor performance exhibited by the DAT00 DE data alone. However, when the results of testing pin MAR07 are considered, both TBE and DE analysis have less than 15 percent of the tests exceeding the 10-percent rejection limit. Clearly the pin-level stuck-at fault model has some modeling capability.

Because of the specific attributes of this experiment, the finding of similar error behavior between pin-level and internal faults cannot be applied in general. The amount of rejection observed is adequate to raise serious doubts as to the usefulness of pin-level fault injection in the validation of highly reliable systems where accurate fault models are necessary.

Conclusion

A technique has been described by which the modeling capability of the pin-level stuck-at fault can be assessed. The technique has been shown to be tractable and has the potential to produce meaningful results. The technique was applied to data acquired during a small-scale fault-injection experiment. Conclusions drawn from these data are thus limited in scope. However, it was found that a pin-level stuck-at fault can generate error behavior very similar to that produced by internal faults. This occurs mainly when the faults (both internal and pin-level) result in a processor crash. Thus it seems that the pin-level stuck-at fault may be used to study a system's response to a large class of faults which are known to cause a processor to crash. However, this leaves many other fault classes which are not modeled well by the pin-level fault. It is recommended that, if the pin-level stuck-at fault model is being considered, additional experimentation be performed to establish when it can be used with confidence. In particular, these experiments should precede any attempt to use the model when validating highly reliable digital systems.

The value of this work can be seen as twofold. It provides an analysis of the pin-level stuck-at fault modeling capability in very large scale integrated (VLSI) circuitry. If a researcher requires a high degree of accuracy over all fault classes, the pin-level stuck-at fault may be discarded based on these results. However, given the need for a more definitive result, this work also establishes an experimental procedure and analysis techniques to be used in future experimental efforts of this kind.

NASA Langley Research Center
Hampton, Virginia 23665-5225
June 25, 1987

Appendix A
Model of Failure-Recovery Process

Consider the model of a failure arrival-recovery process shown in figure 21. The recovery process, which is assumed to have perfect coverage, is modeled with three types of states. State G is the good, or nonfaulty, state. A single processor fails with rate λ to state A, or the active fault state. From state A, the process continues to state R, the recovered state. The rate δ at which the system executes the recovery process is the inverse of the total recovery time. The values G, A, and R denote the number of processors in each state. The state of the process as a whole is defined by the state vector (G, A, R).

System failure is calculated by summing the probabilities of the following two events:

1. All processors have failed, $G = 0$ (lack of spares).
2. The voter has been defeated, $A > G$ (recovery too slow).

The accuracy required in estimating recovery rate can be illustrated by the following example. Given a quad redundant computer (i.e., a four-processor configuration) which executes the recovery process described by the model in figure 21, the following list of probabilities and associated recovery rates has been derived with the SURE modeling tool (ref. 12). Each processor is assumed to have a mean time between failures (MTBF) of 20 000 hours and, therefore, a failure rate λ of 5.0×10^{-5} per hour.

Recovery rate, δ, per hour	Probability of failure
100.0	3.00×10^{-9}
1 000.0	.88
10 000.0	.53

It can be seen from these data that the recovery rate must be known to be within an order of magnitude of 1000 per hour to give reasonable confidence that the system exhibits a probability of failure on the order of 10^{-9}.

If an experimental procedure is to be used to obtain the recovery process coverage parameter, a much higher degree of accuracy must be achieved. It can be shown through use of a sensitivity analysis similar to the one above that the coverage parameter must be greater than 0.9999999. This value is equivalent to a coverage failure once every 10^7 trials. Observing such a rare event is beyond the capability of most experimental procedures. Fault-injection experiments will most likely not be used to obtain this parameter.

Appendix B

Experiment Configuration and Procedure

The experiment procedure can be divided into the following four steps:

1. Initialize system.
2. Observe system state.
3. Stimulate system.
4. Record system response.

This series of steps is repeated for each run. System initialization prior to each run ensures independence as required by assumption 7. The system is the Vmp performing a Fibonacci series calculation. The stimulus is provided by fault injection. The state of the Vmp is observed at its 48-pin boundary.

The system configuration upon which the experiment was performed consists of the following four components:

1. A system emulation-simulation (the Vmp)
2. A representative work load (the Fibonacci series)
3. A stimulus (fault injection)
4. A system monitor (a logic analyzer)

The following sections describe these four components, the combined configuration, and the procedure which controlled the experiment.

The Vmp

As mentioned in the section entitled *VLSI Circuit Model: The Virtual Microprocessor*, the Vmp is constructed from a Bendix BDX-930 processor. The BDX-930 is a 16-bit pipelined processor with a minimum instruction execution time of 250 ns obtained with the nominal 16 Mhz oscillator. At the heart of the BDX-930 are four bit-slice processors. The bit-slice processors contain the arithmetic and logic unit (ALU) and 16 general purpose registers. Surrounding the bit-slice processor chips are approximately 100 SSI and MSI circuits. These peripheral circuits perform the following functions (see fig. 22):

1. Micro code sequence control
2. Pipelined instruction decoding
3. Address processing
4. Data path buffering
5. Status registers
6. Timing and control

The signals chosen to represent the 48 pins of the Vmp are listed in table I along with their associated BDX-930 chip and pin number designation. The 48 signals are divided into 16 data bus signals (DAT00 to DAT15), 16 memory address signals (MAR00* to MAR15*), and 16 miscellaneous signals. The 16 miscellaneous signals can be classified as follows:

Clock: ABUF*
Power-on reset: POS and PON
CPU status: FOV, IND, LINK, PFEIN, FLAG1, and FLAG2
Memory arbitration: MM*, MEM1, MEM2, WE1, WE2, CDE01, and CDE02

The BDX-930 used in the experiment is part of the Software Implemented Fault Tolerance (SIFT) computer system, which is currently undergoing evaluation in the Langley Avionics Integration Research Laboratory (AIRLAB). Access to and control of the BDX-930 was obtained through the SIFT interface available in AIRLAB. Further information on the interface of SIFT, BDX-930, and AIRLAB can be obtained in reference 13.

The Fibonacci Series

A program which produces a Fibonacci series was chosen to exercise the processor during the fault-injection tests. The elements j_i of a Fibonacci series are computed by the following sum:

$$j_i = j_{i-1} + j_{i-2} \ (i = 3, 4, ..., n)$$

where initially

$$j_1 = 1 \text{ and } j_2 = 2$$

This algorithm can be coded efficiently in assembler code. However, what is desired is not an efficient, compact program, but a program which represents a good instruction mix. For example, to introduce stack manipulation instructions, the algorithm was coded as a recursive Pascal procedure. The Pascal program was compiled into BDX-930 assembler language (see appendix C) using SIFT's Pascal cross assembler. A halt instruction was inserted in the assembler code between calls to the Fibonacci procedure to provide a checkpoint in the test when the logic analyzer limited memory could be stored in a data set. A data set would then contain an integral number of iterations. In the final program, the results of one iteration fit in each data set. The remainder of the data set was filled with the results of executing a sequence of instructions which contained operations such as shifting, integer multiplication and division, and testing and branching.

Because the error behavior exhibited by a faulty processor is the product of the physical nature of the fault and the instruction codes executed by the processor, the results of this study may be critically dependent on the test program. If the processor takes

a wild branch out of the expected instruction stream because of the fault, the error behavior becomes dependent on the state of accessible memory. In this study, the Fibonacci program used about 100 words of the 32 K words available in the BDX-930. Although the program was reloaded for each test, no provision was made to set the unused memory to an initial state. This oversight may contribute to a lack of independence between each test.

The Fault Injector

The In-Circuit Fault Injector (ICFI) available in AIRLAB was used to apply the stuck-at one and stuck-at zero faults to the pins of the integrated circuits within the Vmp. The ICFI can be connected directly to the subject pin without extending the integrated circuit from the printed circuit board. The ICFI can be controlled manually from a front panel or remotely as part of the SIFT AIRLAB interface. The ICFI also has an external trigger available. The external trigger provides an experimenter with the flexibility of being able to set the fault parameters from the host computer while fault activation is controlled by an external signal from the experiment apparatus. This was the mode used during the fault-injection tests with the external trigger controlled by the logic analyzer.

The Logic Analyzer

The logic analyzer has 6 channels of 25-Mhz acquisition and 16 channels of pattern generation data. All 48 pins of the Vmp were acquired (see table I), plus the 7 test signals shown in table II.

Signal A* is an 8-Mhz clock which was used as the logic analyzer external clock. The 512-word memory capacity of the logic analyzer thus provided a 64-ms data acquisition window.

The signals TDR*, TIB*, and EOUT* define the operation occurring on the DAT bus. The TDR* is asserted during data input, TIB* is asserted during instruction input, and EOUT* is asserted during data output.

Signal HLTLP, which indicates whether the BDX-930 is running or halted, was used to activate a procedure in the logic analyzer pattern generator upon BDX-930 start-up. The procedure consisted of a wait-for-interrupt loop in the mainline and a simple interrupt handler which asserted one bit of the pattern generator output. This bit was used to trigger the ICFI.

The remaining signals were unused.

The logic analyzer provides local offline storage of its setup on cassette tape, thus ensuring an identical setup for each test session.

The Experiment Configuration

Figure 23 illustrates the experiment configuration. Of the 55 signals acquired by the logic analyzer, all but 2 (MAR15* and LINK) are available on the BDX-930 backplane. A total of 38 signals (MAR00* to MAR15*, DAT00 to DAT15, FOV, IND, LINK, PFEIN, FLAG1, and FLAG2) are located on the BDX-930 CPU board. The remaining 10 Vmp signals are found on the timing and control board. All seven test signals are on the CPU board. The BDX-930 was removed from the SIFT test stand to allow access to the backplane. The BDX-930 was still controllable from the SIFT host environment through an extension cable which connected the BDX-930 to the SIFT test stand. A separate adapter provided power (28 V dc and 110 V ac at 400 Hz).

One bit of the logic analyzer pattern generator output was connected to the external trigger of the ICFI. An interrupt procedure in the pattern generator which asserted this bit was enabled by the BDX-930 HLTLP signal, that is, when the BDX-930 was started. The pattern generator signal activated the fault, which had been programmed remotely into the ICFI. The ICFI fault-injection probe was in turn connected to a pin of the BDX-930. Table III is a list of the pins tested during this experiment.

Finally, the logic analyzer connector was connected to a port on the host computer. The logic analyzer was controlled entirely through the host computer. Acquisition memory was recovered over the RS232 link and saved in individual data files.

The Experiment Procedure

With the assumption that the experiment configuration is as described in the previous section, the operator begins the experiment by entering the SIFT environment on the host computer and defining the fault model for this test session with the SIFT FAULT command. Under the control of a command procedure, the test session occurs in three phases.

The first phase is initialization. The operator is asked for the fault number with which to start the test session. Of approximately 1800 pins in the BDX-930, 68 were chosen for testing and placed in a fault definition file. (See table III.) An effort was made to sample pins from most of the integrated circuits of the BDX-930. The ordering of the pins in the table gives each fault an implied number. Notice in table III that the odd-numbered faults are stuck-at one faults and the even-numbered faults are

stuck-at zero faults. During testing, only one fault type is used at a time; that is, all the stuck-at one faults are done, followed by the stuck-at zero faults. This is done to reduce operator input and, therefore, the chance of error. If the testing occurred with alternating stuck-at one and stuck-at zero faults, the operator would have to modify the fault model with the SIFT FAULT command before each test.

Once the fault number is entered, the second phase of the test session begins. The ICFI is remotely programmed according to the fault model definition. The fault description (signal name, board name, chip number, and pin number) associated with the fault number is displayed on the operator's terminal. The operator then places the fault-injection probe on this pin.

When the probe is set, the operator resumes the procedure by entering the name of the results file. File names are constructed according to the following convention:

/Board Name/Chip Number/Pin Number/Fault Type/.DAT.

For example, the results of testing fault number 1 (SRAM15, stuck-at one) will be stored in file CPU1107S1.DAT. (See table III.) The results of fault number 86 (QII, stuck-at zero) will be stored in file TC3206S0.DAT. The term CPU stands for the central processing unit board and TC stands for the timing and control board.

This completes the operator input. Operator intervention has been required for 4 items: entering the fault type, fault number, and results file name, and positioning the probe.

The second phase of the command procedure ends with the following sequence: The Fibonacci program is loaded into the test processor and started. The processor initializes and comes to a programmed halt just prior to entering the first Fibonacci iteration. (See appendix C for program listing.) A remote halt command ensures that the processor does not restart.

In the final phase of the test session, the processor executes a series of Fibonacci iterations. The data from each iteration is captured, compared with the fault-free file, and stored. If no errors are found, another iteration is performed. Up to 20 iterations are processed.

Appendix C
Fibonacci Program Listing

PROGRAM FIB

```
LOC   OBJ       MREF  STMT          SOURCE  STATEMENT
---   -----     ---    - - - - - - - - - - - - - - - - - - - -

                       1  *
                       2  *
                       3  *
                       4          ABS
0100                   5          ORG     100H
0100  0424             6          CONT    ER,1S
0101  9000       1027  7          JU      MAIN$
1000                   8          ORG     1000H
                       9  *
                      10  *
1000  0000            11  A$2     FIX     0
1001  1002            12  A$5     LINK    FIBER
                      13  *
1002  7F03420F        14  FIB     PUSHM   0,3       FIBONACCI PROCEDURE
1004  000F            15          TRA     0,15
1005  59FA            16          LOAD    1,-6,0    =DATA
1006  5AFB            17          LOAD    2,-5,0    =NUM
1007  8F2F            18          IAR     2,-1
1008  46F8       1000 19          CMP     2,A$2          =1
1009  1408       1011 20          JU      A$0
100A  1402       100C 21          JU      A$1
100B  1406       1011 22          JU      A$0
                      23  *
100C  7F01421F        24  A$1     PUSHM   1,2
100E  FF0094F3   1001 25          JSS*    A$5       =FIB
1010  8FFE            26          IAR     15,-2
                      27  *
1011  0812            28  A$0     ADDR    1,2       = ADDRESS DATA + NUM -1
1012  5E00            29          LOAD    2,0,1     = DATA [NUM]
1013  8F11            30          IAR     1,1
1014  5F00            31          LOAD    3,0,1     = DATA [NUM+1]
1015  0823            32          ADDR    2,3
1016  8F1F            33          IAR     1,-1
1017  6E00            34          STO     2,0,1     = → DATA[NUM]
1018  8F12            35          IAR     1,2
1019  6E00            36          STO     2,0,1     = → DATA[NUM+1]
101A  00F0            37          TRA     15,0
101B  7F03410F        38          POPM    0,3
101D  FF001200        39          RPS     0
                      40  *
                      41  * Size: 47
                      42  *
                      43  *
                      44  *
101F  0100            45  A$6     LINK    STAC$
1020  2048            46  A$8     LINK    C1
```

```
LOC  OBJ       MREF STMT       SOURCE STATEMENT
---  -----     ---  ---------- ----------------------------
1021 0001           47   A$9    FIX    1
1022 1047           48   A$10   LINK   DATA
1023 0001           49   A$11   FIX    1
1024 1002           50   A$12   LINK   FIB
1025 77FB           51   A$7    LINK   30715
1026 2049           52   A$13   LINK   C2
                    53   *
                    54          ENTRY  MAIN$     MAINLINE AND INITIALIZATION
1027 54F8   101F    55   MAIN$  LOAD   0,A$6        =STAC$
1028 00F0           56          TRA    15,0
1029 56F8   1021    57          LOAD   2,A$9        =1
102A 55F8   1022    58          LOAD   1,A$10
102B 6E00           59          STO    2,0,1        =DATA
102C 6E01           60          STO    2,1,1        = DATA + 1
102D 1418   1045    61          JU     A$27
102E 7F00421F       62   LOOP   PUSHM  1,1          BEGINNING OF ITERATION
1030 57F3   1023    63          LOAD   3,A$11       =4
1031 7F00423F       64          PUSHM  3,3
1033 FF0094F1 1024  65          JSS*   A$12         =FIBO
                    66   *      IAR    15,-2
1035 0061           67          TRA    6,1
1036 7D741D00       68          LDM    7,11,0,1     SEQUENCE OF INSTRUCTIONS
1038 0267           69          LCM    6,7
1039 0667           70          AND    6,7
103A 8084           71          SLSA   8,4
103B 0B68           72          MPY    6,8
103C 0F86           73          DIV    8,6
103D 0C9B           74          SUBR   9,11
103E 8291   1040    75          SKGT   9,A$25
103F 0CBB           76          CLAO   11,11
1040 0561   1042    77   A$25   DECNE  6,A$26
1041 0000           78          NOP
1042 FF0021AC       79   A$26   DACM   10,12
1044 0813           80          ADDR   1,3
1045 FC00           81   A$27   HALT                END OF 1 ITERATION
1046 14E8   102E    82          JU     LOOP
                    83   *
                    84   * Size: 26
                    85   *
     0100           86   STAC$  EQU    256
1047 0000           87   DATA   BSZ    4096
2047                88   I      RES    1
2048                89   C1     RES    1
2049                90   C2     RES    1
204A                91          END
```

References

1. Gault, James W.; Trivedi, Kishor S.; and Clary, James B., eds.: *Validation Methods Research for Fault-Tolerant Avionics and Control Systems—Working Group Meeting II.* NASA CP-2130, 1980.
2. Lala, Jaynarayan H.; and Smith, T. Basil, III: *Development and Evaluation of a Fault-Tolerant Multiprocessor (FTMP) Computer. Volume III—FTMP Test and Evaluation.* NASA CR-166073, 1983.
3. Siewiorek, Daniel P.; and Lai, Larry Kwok-Woo: Testing of Digital Systems. *Proc. IEEE,* vol. 69, no. 10, Oct. 1981, pp. 1321–1333.
4. Hopkins, Albert L., Jr.; Smith, T. Basil, III; and Lala, Jaynarayan H.: FTMP—A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft. *Proc. IEEE,* vol. 66, no. 10, Oct. 1978, pp. 1221–1239.
5. Smith, T. Basil: Fault Tolerant Processor Concepts and Operation. *The Fourteenth International Conference on Fault-Tolerant Computing—Digest of Papers,* IEEE Catalog No. 84CH2050-3, IEEE Computer Soc. Press, c.1984, pp. 158–163.
6. Goldberg, Jack; Kautz, William H.; Melliar-Smith, P. Michael; Green, Milton W.; Levitt, Karl N.; Schwartz, Richard L.; and Weinstock, Charles B.: *Development and Analysis of the Software Implemented Fault-Tolerance (SIFT) Computer.* NASA CR-172146, 1984.
7. *Military Handbook—Reliability Prediction of Electronic Equipment.* MIL-HDBK-217D, U.S. Dep. of Defense, Jan. 15, 1982. (Supersedes MIL-HDBK-217C, Apr. 9, 1979.)
8. Hollander, Myles; and Wolfe, Douglas A.: *Nonparametric Statistical Methods.* John Wiley & Sons, Inc., c.1973.
9. *User's Manual.* IMSL Library—Problem-Solving Software System for Mathematical and Statistical FORTRAN Programming, Volume 3, Edition 9.2, IMSL LIB-0009, IMSL, Inc., c.1984.
10. McGough, John G.; and Swern, Fred L.: *Measurement of Fault Latency in a Digital Avionic Mini Processor.* NASA CR-3462, 1981.
11. Nagel, Phyllis M.: *Modeling of a Latent Fault Detector in a Digital System.* NASA CR-145371, 1978.
12. Butler, Ricky W.: *The SURE Reliability Analysis Program.* NASA TM-87593, 1986.
13. Green, David F., Jr.; Palumbo, Daniel L.; and Baltrus, Daniel W.: *Software Implemented Fault-Tolerant (SIFT) User's Guide.* NASA TM-86289, 1984.

Table I. The 48 Pins of Virtual Microprocessor

[An asterisk indicates active when signal low]

Signal	Micro pin	Board	Chip	Edge connector
MAR00*	1	CPU	U43 p20	J10 p7B
MAR01*	2		U43 p1	J10 p6B
MAR02*	3		U43 p16	J10 p5B
MAR03*	4		U43 p14	J10 p4B
MAR04*	5		U42 p20	J10 p17B
MAR05*	6		U42 p18	J10 p16B
MAR06*	7		U42 p16	J10 p15B
MAR07*	8		U42 p14	J10 p13B
MAR08*	9		U40 p20	J10 p39B
MAR09*	10		U40 p18	J10 p38B
MAR10*	11		U40 p16	J10 p36B
MAR11*	12		U40 p14	J10 p35B
MAR12*	13		U39 p20	J10 p56B
MAR13*	14		U39 p18	J10 p54B
MAR14*	15		U39 p16	J10 p53B
MAR15*	16		U39 p14	J10 p52B
DAT00	17		U37 p18	J10 p27B
DAT01	18		U37 p17	J10 p9A
DAT02	19		U37 p14	J10 p22B
DAT03	20		U37 p13	J10 p21B
DAT04	21		U37 p8	J10 p11A
DAT05	22		U37 p7	J10 p12A
DAT06	23		U37 p4	J10 p39A
DAT07	24		U37 p3	J10 p13A
DAT08	25		U36 p18	J10 p34B
DAT09	26		U36 p17	J10 p33B
DAT10	27		U36 p14	J10 p34A
DAT11	28		U36 p13	J10 p31B
DAT12	29		U36 p8	J10 p31A
DAT13	30		U36 p7	J10 p33A
DAT14	31		U36 p4	J10 p36A
DAT15	32		U36 p3	J10 p38A
FOV	33		U6 p6	J10 p35A
IND	34		U13 p6	J10 p1A
LINK	35		U13 p8	J10 p2B
PFEIN	36		U6 p8	J10 p5A
FLAG1	37		U7 p6	J10 p8A
FLAG2	38		U7 p8	J10 p10A
POS	39	TC	U20 p6	J9 p21A
PON	40		U19 p12	J9 p26C
MM*	41		U41 p6	J9 p27C
MEM1	42		U58 p9	J9 p52B
MEM2	43		U58 p10	J9 p52C
WE1	44		U58 p7	J9 p44A
WE2	45		U58 p6	J9 p54C
CDE01	46		U46 p10	J9 p49C
CDE02	47		U46 p9	J9 p48C
ABUF*	48		U14 p8	J9 p5B

Table II. Test Signals

[An asterisk indicates active when signal low]

Signal	Board	Pin	Use
A*	CPU	J9 p24B	External clock
AI	↓	J10 p11B	Clock
BBUF		J10 p44A	Clock
HLTLP		J10 p6A	Trigger injection
TDR*		J10 p19B	Input data
TIB*		J10 p18B	Instruction
EOUT*		J10 p29B	Output data

Table III. List of Pins Tested

[An asterisk indicates active when signal low]

No.	Signal	Processor	Board	Chip	Pin	Fault type
1	SRAM15	Proc7	CPU	U11	p7	SA1
2	SRAM15	↓	↓	U11	p7	SA0
3	SQ00			U11	p9	SA1
4	SQ00			U11	p9	SA0
5	LINK			U13	p8	SA1
6	LINK			U13	p8	SA0
7	IND			U13	p6	SA1
8	IND			U13	p6	SA0
9	RPTOV*			U28	p15	SA1
10	RPTOV*			U28	p15	SA0
11	QBIT			U12	p6	SA1
12	QBIT			U12	p6	SA0
13	CON			U12	p8	SA1
14	CON			U12	p8	SA0
15	TIB*			U20	p6	SA1
16	TIB*			U20	p6	SA0
17	TDR*			U20	p11	SA1
18	TDR*			U20	p11	SA0
19	IR01			U35	p3	SA1
20	IR01			U35	p3	SA0
21	SRAM00			U35	p16	SA1
22	SRAM00			U35	p16	SA0
23	C0UT			U35	p33	SA1
24	C0UT			U35	p33	SA0
25	SPA0			U35	p4	SA1
26	SPA0			U35	p4	SA0
27	U10			U35	p12	SA1
28	U10			U35	p12	SA0
29	DAT15			U30	p2	SA1
30	DAT15			U30	p2	SA0
31	DAT07			U31	p2	SA1
32	DAT07			U31	p2	SA0
33	DAT00			U31	p19	SA1
34	DAT00			U31	p19	SA0
35	Y00			U34	p19	SA1
36	Y00			U34	p19	SA0
37	D00			U37	p19	SA1
38	D00			U37	p19	SA0
39	MAR15*			U39	p14	SA1
40	MAR15*			U39	p14	SA0
41	MAR07*			U42	p14	SA1
42	MAR07*			U42	p14	SA0
43	MAR00*			U43	p20	SA1
44	MAR00*			U43	p20	SA0
45	UMA1			U45	p7	SA1
46	UMA1	↓	↓	U45	p7	SA0

Table III. Continued

No.	Signal	Processor	Board	Chip	Pin	Fault type
47	UMA0	Proc7	CPU	U45	p6	SA1
48	UMA0	↓	↓	U45	p6	SA0
49	Y15	↓	↓	U33	p2	SA1
50	Y15	↓	↓	U33	p2	SA0
51	ESTRT*	↓	↓	U70	p2	SA1
52	ESTRT*	↓	↓	U70	p2	SA0
53	EMSB*	↓	↓	U70	p5	SA1
54	EMSB*	↓	↓	U70	p5	SA0
55	ESPC*	↓	↓	U70	p3	SA1
56	ESPC*	↓	↓	U70	p3	SA0
57	EBCH*	↓	↓	U70	p1	SA1
58	EBCH*	↓	↓	U70	p1	SA0
59	ETIR*	↓	↓	U70	p4	SA1
60	ETIR*	↓	↓	U70	p4	SA0
61	ELSB*	↓	↓	U61	p1	SA1
62	ELSB*	↓	↓	U61	p1	SA0
63	HALTM*	↓	↓	U64	p6	SA1
64	HALTM*	↓	↓	U64	p6	SA0
65	IAM*	↓	↓	U62	p3	SA1
66	IAM*	↓	↓	U62	p3	SA0
67	IRS	↓	↓	U09	p2	SA1
68	IRS	↓	↓	U09	p2	SA0
69	HLTLP	↓	↓	U65	p12	SA1
70	HLTLP	↓	↓	U65	p12	SA0
71	U30	↓	↓	U67	p15	SA1
72	U30	↓	↓	U67	p15	SA0
73	U54	↓	↓	U68	p2	SA1
74	U54	↓	↓	U68	p2	SA0
75	FOV	↓	↓	U06	p6	SA1
76	FOV	↓	↓	U06	p6	SA0
77	FLAG2	↓	↓	U71	p8	SA1
78	FLAG2	↓	↓	U71	p8	SA0
79	FLAG1	↓	↓	U71	p6	SA1
80	FLAG1	↓	↓	U71	p6	SA0
81	SPB0	↓	↓	U27	p6	SA1
82	SPB0	↓	↓	U27	p6	SA0
83	QIO*	↓	TC	U32	p3	SA1
84	QIO*	↓	↓	U32	p3	SA0
85	QII*	↓	↓	U32	p6	SA1
86	QII*	↓	↓	U32	p6	SA0
87	NORM*	↓	↓	U32	p11	SA1
88	NORM*	↓	↓	U32	p11	SA0
89	MEM*	↓	↓	U32	p8	SA1
90	MEM*	↓	↓	U32	p8	SA0
91	MM*	↓	↓	U44	p6	SA1
92	MM*	↓	↓	U44	p6	SA0
93	QIOIN*	↓	↓	U05	p6	SA1
94	QIOIN*	↓	↓	U05	p6	SA0
95	MI*	↓	↓	U35	p6	SA1
96	MI*	↓	↓	U35	p6	SA0

Table III. Concluded

No.	Signal	Processor	Board	Chip	Pin	Fault type
97	PST	Proc 7	TC	U13	p6	SA1
98	PST	↓	↓	U13	p6	SA0
99	BOSC1			U14	p10	SA1
100	BOSC1			U14	p10	SA0
101	BOSC*			U14	p2	SA1
102	BOSC*			U14	p2	SA0
103	ACK*			U14	p8	SA1
104	ACK*			U14	p8	SA0
105	PON			U19	p12	SA1
106	PON			U19	p12	SA0
107	FWOT			U20	p8	SA1
108	FWOT			U20	p8	SA0
109	EOUT*			U04	p6	SA1
110	EOUT*			U04	p6	SA0
111	POS			U20	p6	SA1
112	POS			U20	p6	SA0
113	CDE02*			U46	p9	SA1
114	CDE02*			U46	p9	SA0
115	CDE01*			U46	p10	SA1
116	CDE01*			U46	p10	SA0
117	MMI1*			U46	p4	SA1
118	MMI1*			U46	p4	SA0
119	MMI2*			U46	p5	SA1
120	MMI2*			U46	p5	SA0
121	MEM1*			U58	p9	SA1
122	MEM1*			U58	p9	SA0
123	MEM2*			U58	p10	SA1
124	MEM2*			U58	p10	SA0
125	WE1			U61	p6	SA1
126	WE1			U61	p6	SA0
127	WE2			U61	p8	SA1
128	WE2			U61	p8	SA0
129	QMI*			U38	p8	SA1
130	QMI*			U38	p8	SA0
131	AI*			U08	p9	SA1
132	AI*			U08	p9	SA0
133	B*			U08	p5	SA1
134	B*			U08	p5	SA0
135	BBUF			U14	p4	SA1
136	BBUF			U14	p4	SA0

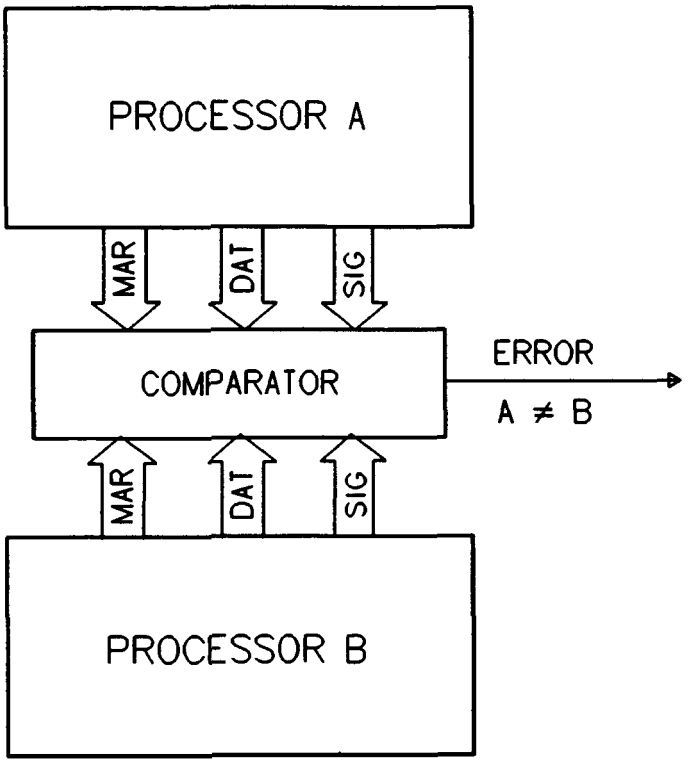

Figure 1. Self-checking dual processor.

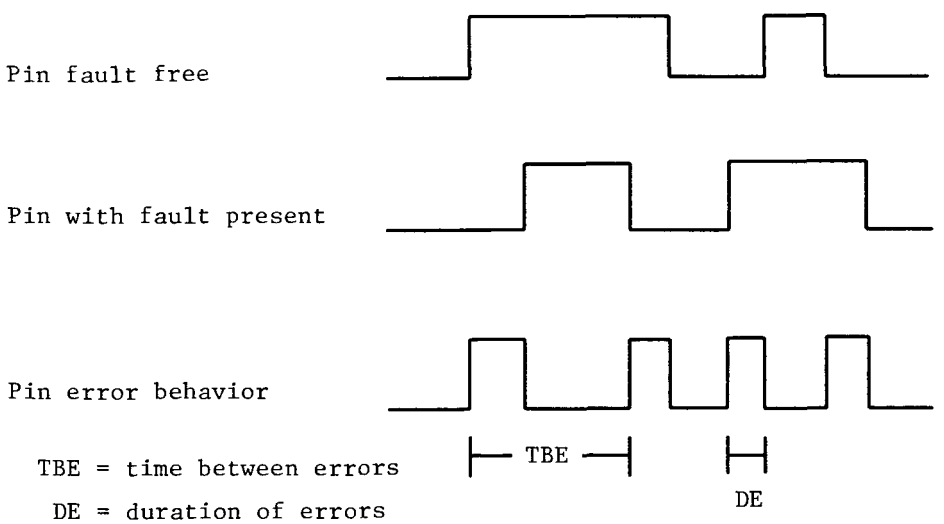

Figure 2. Dynamic error behavior definition.

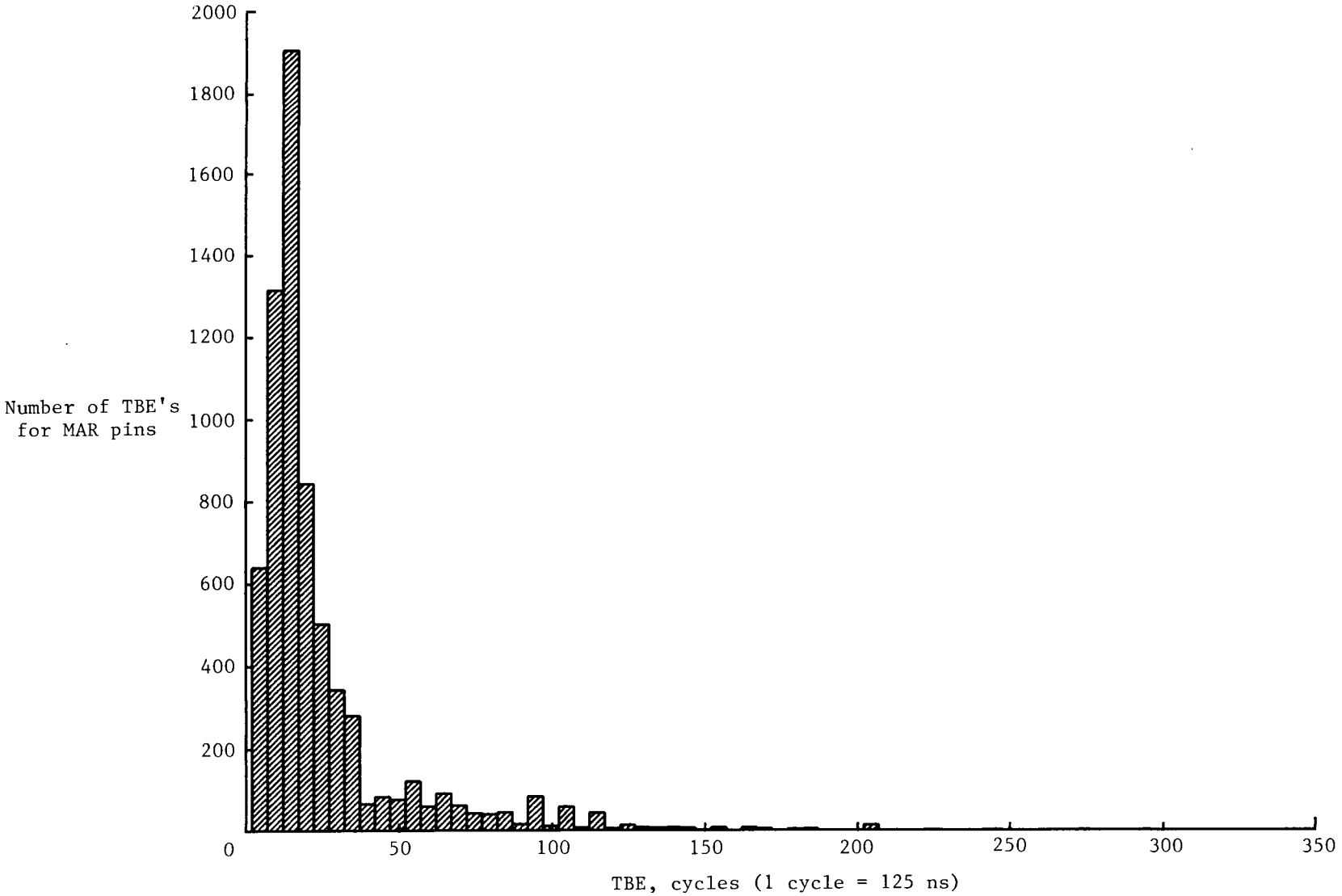

Figure 3. Histogram of TBE for MAR bus.

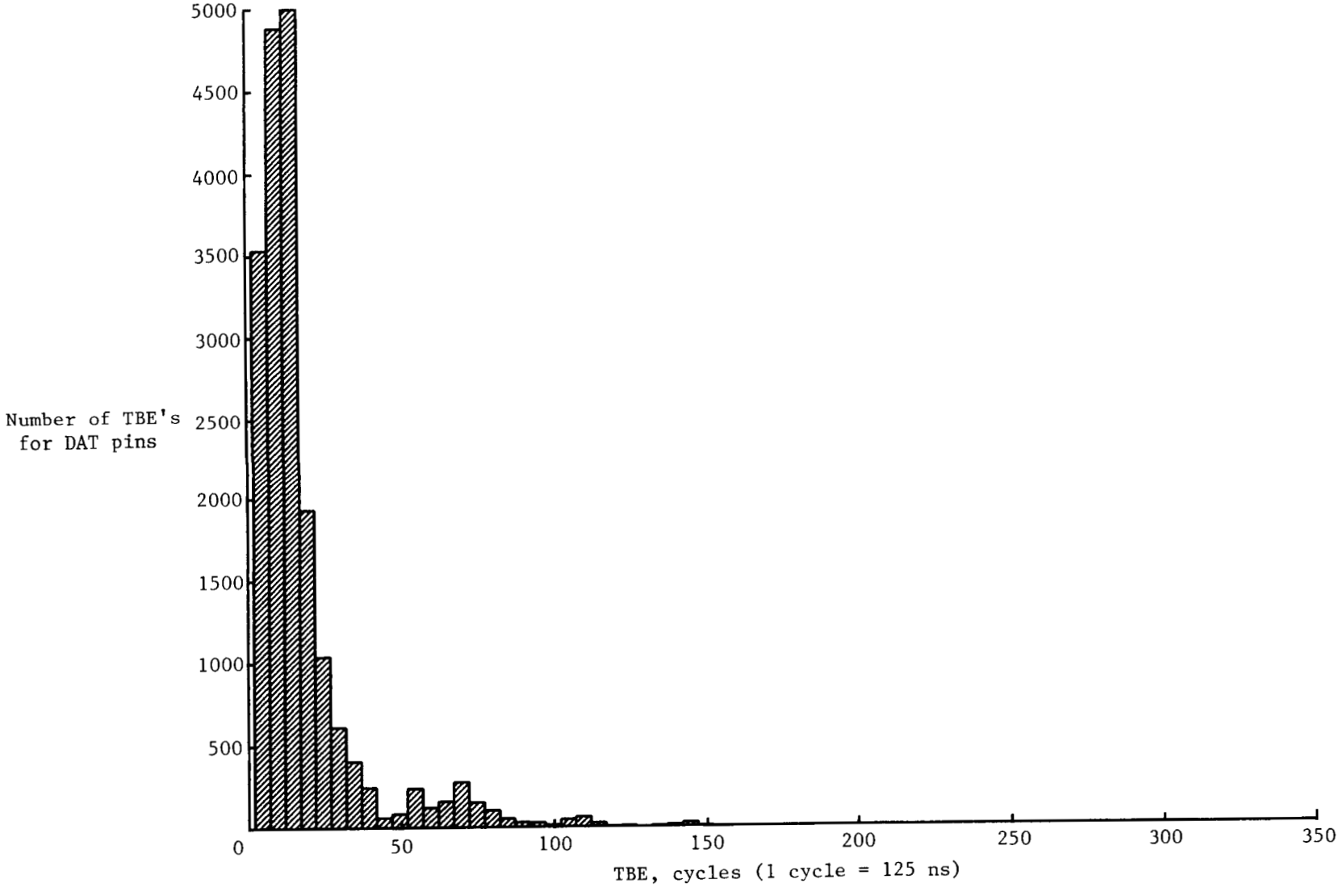

Figure 4. Histogram of TBE for DAT bus.

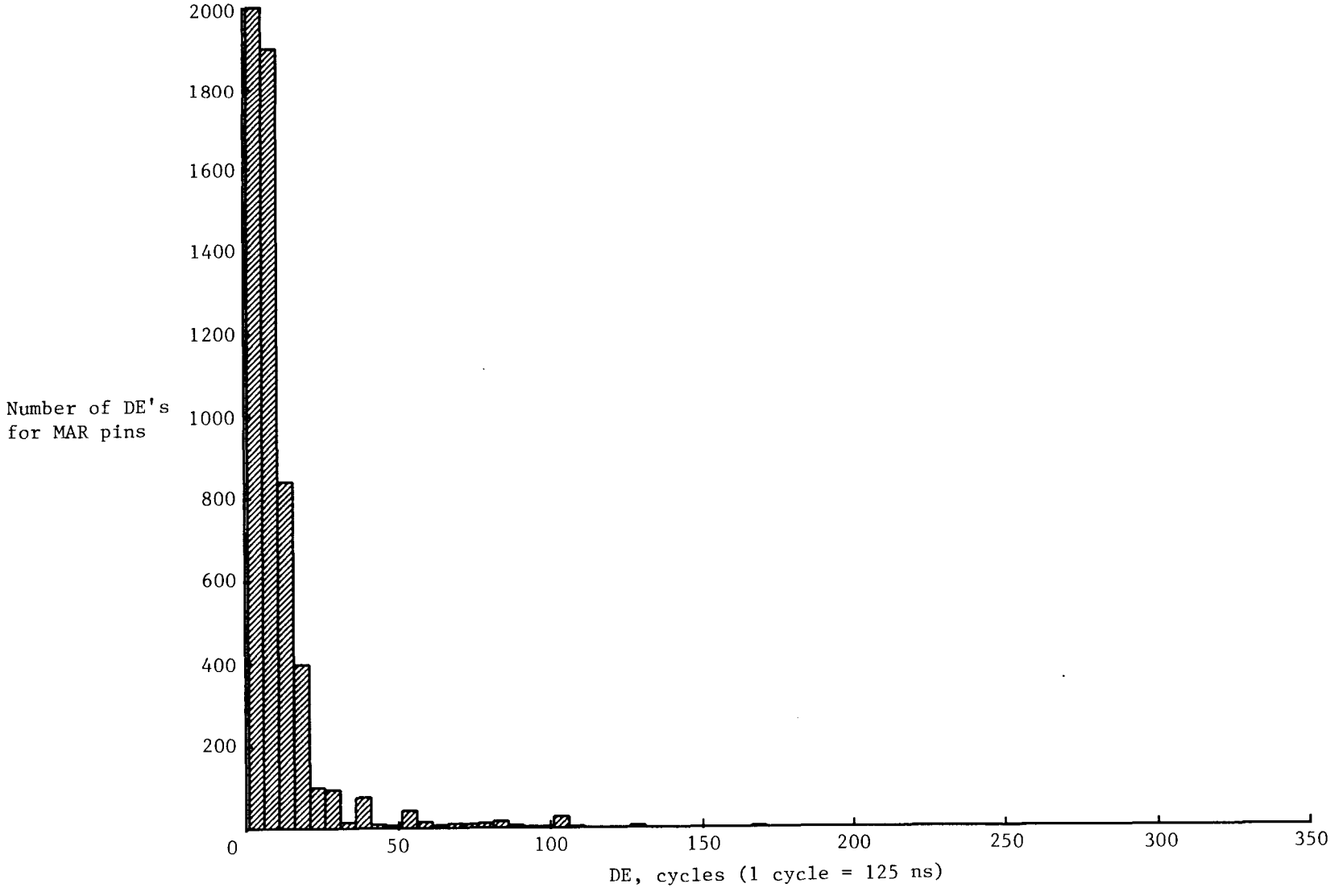

Figure 5. Histogram of DE for MAR bus.

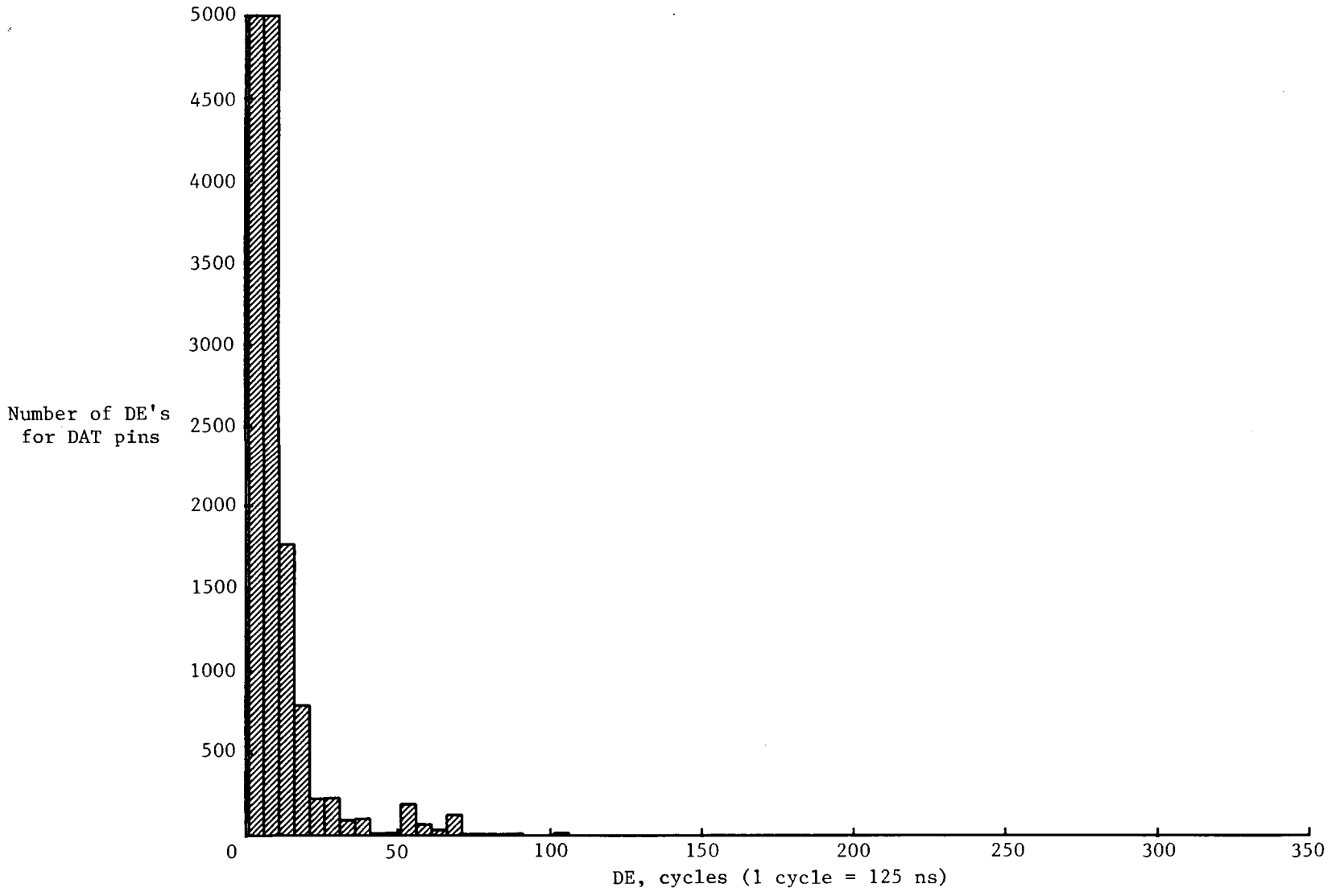

Figure 6. Histogram of DE for DAT bus.

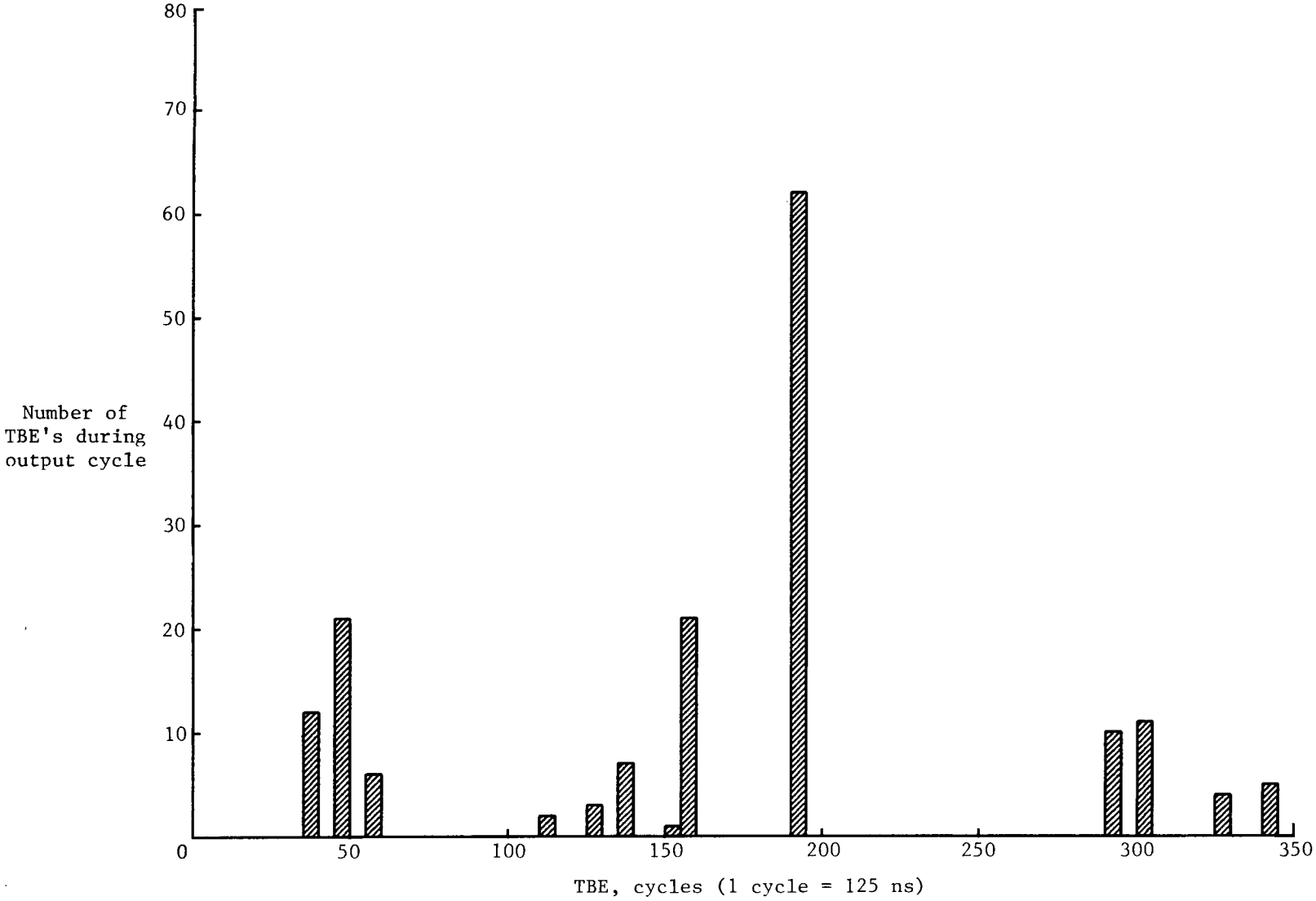

Figure 7. Histogram of TBE during output cycle.

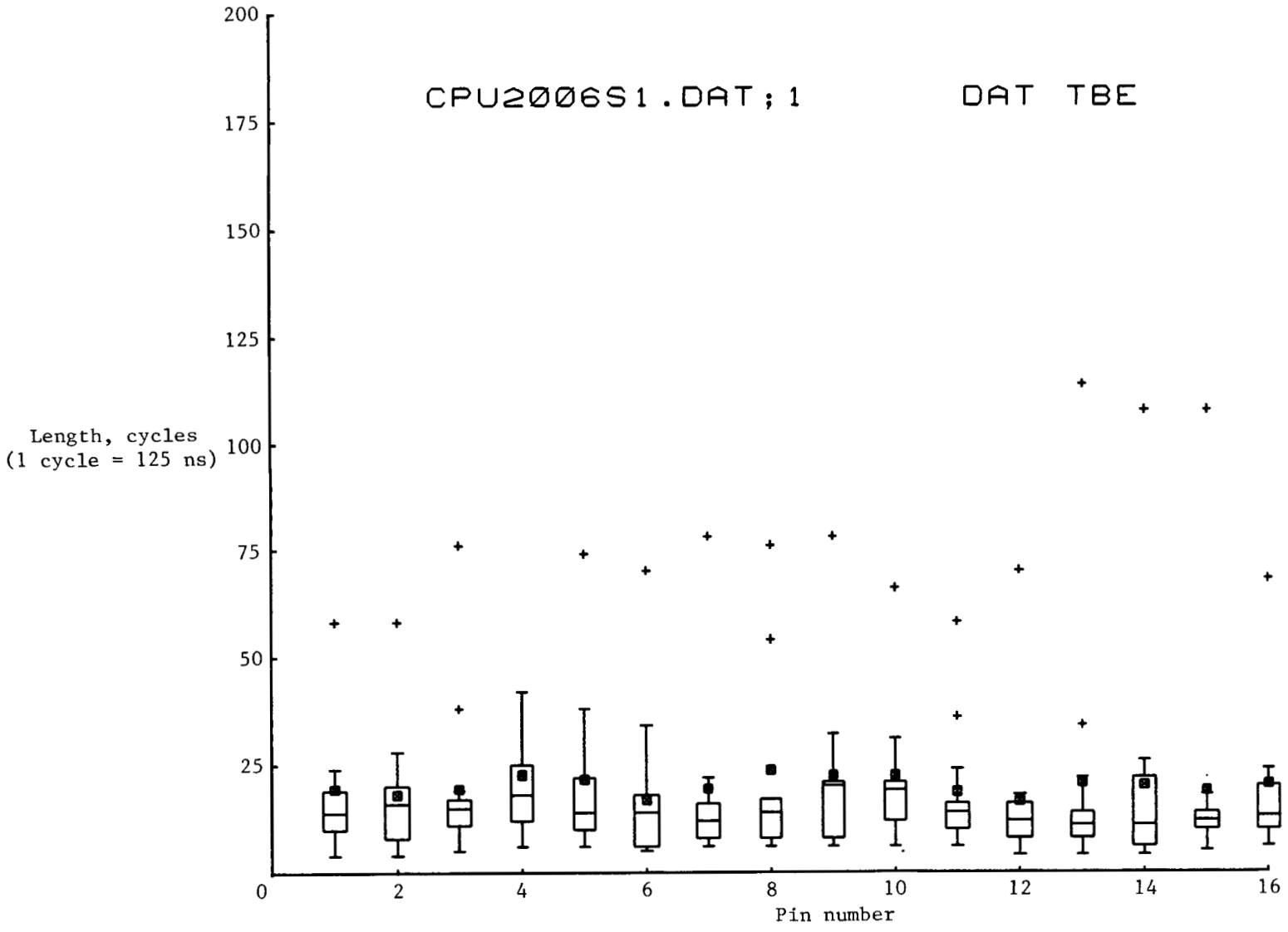

Figure 8. Box plot of internal fault for TBE from DAT bus.

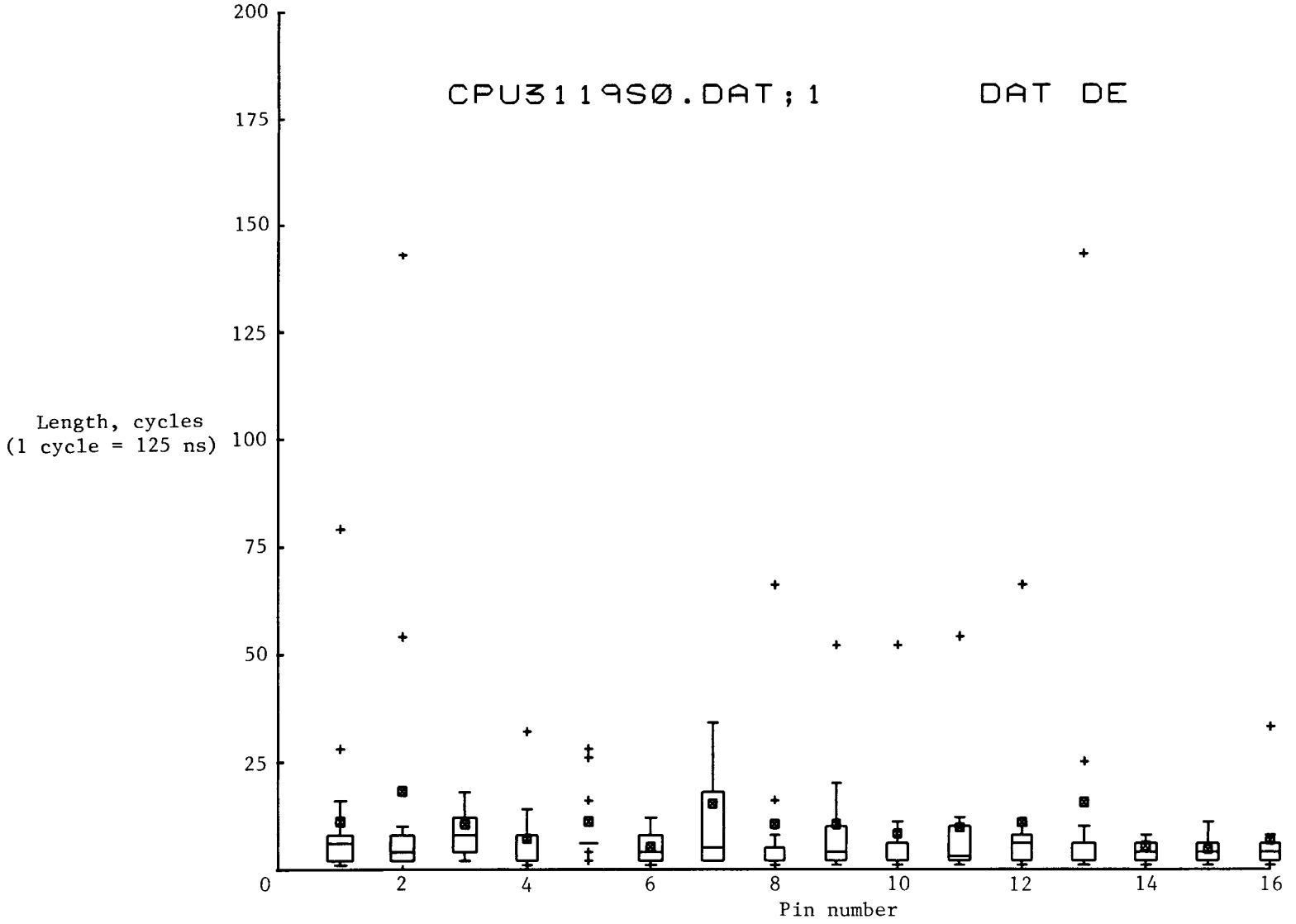

Figure 9. Box plot of pin-level fault (DAT00) for TBE from DAT bus.

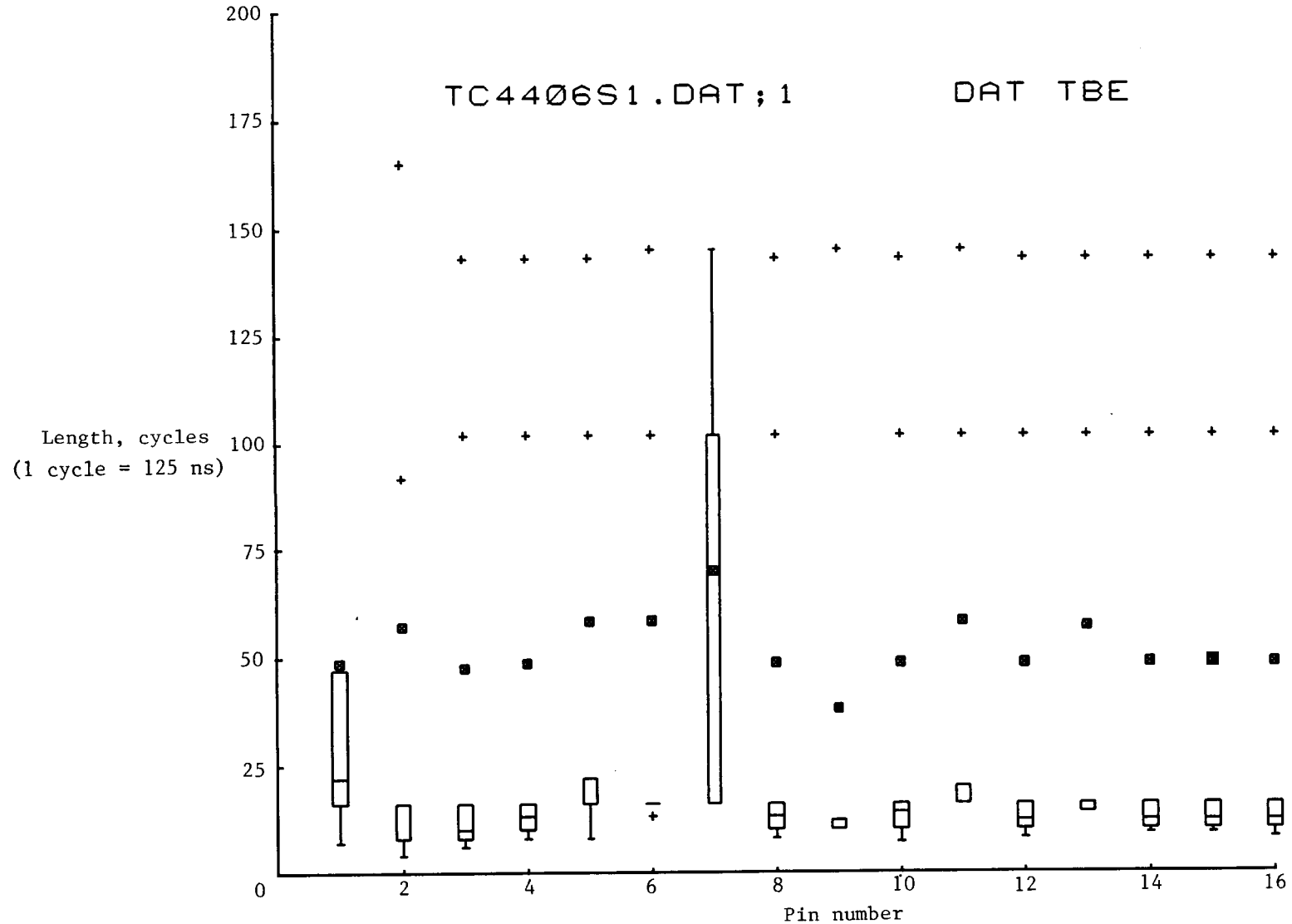

Figure 10. Box plot of different internal fault behavior.

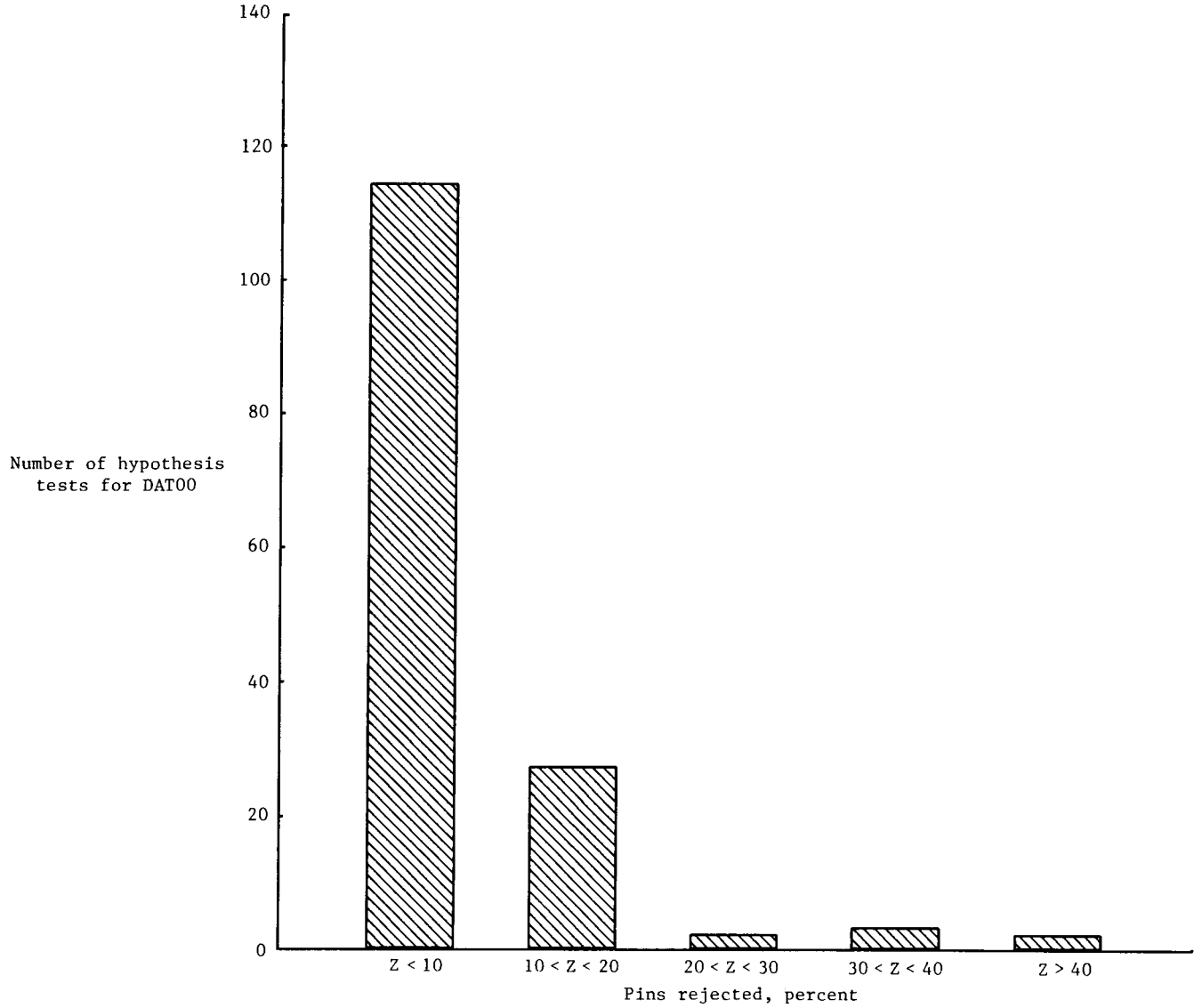

Figure 11. Test results, DAT00 vs all internal faults, for TBE.

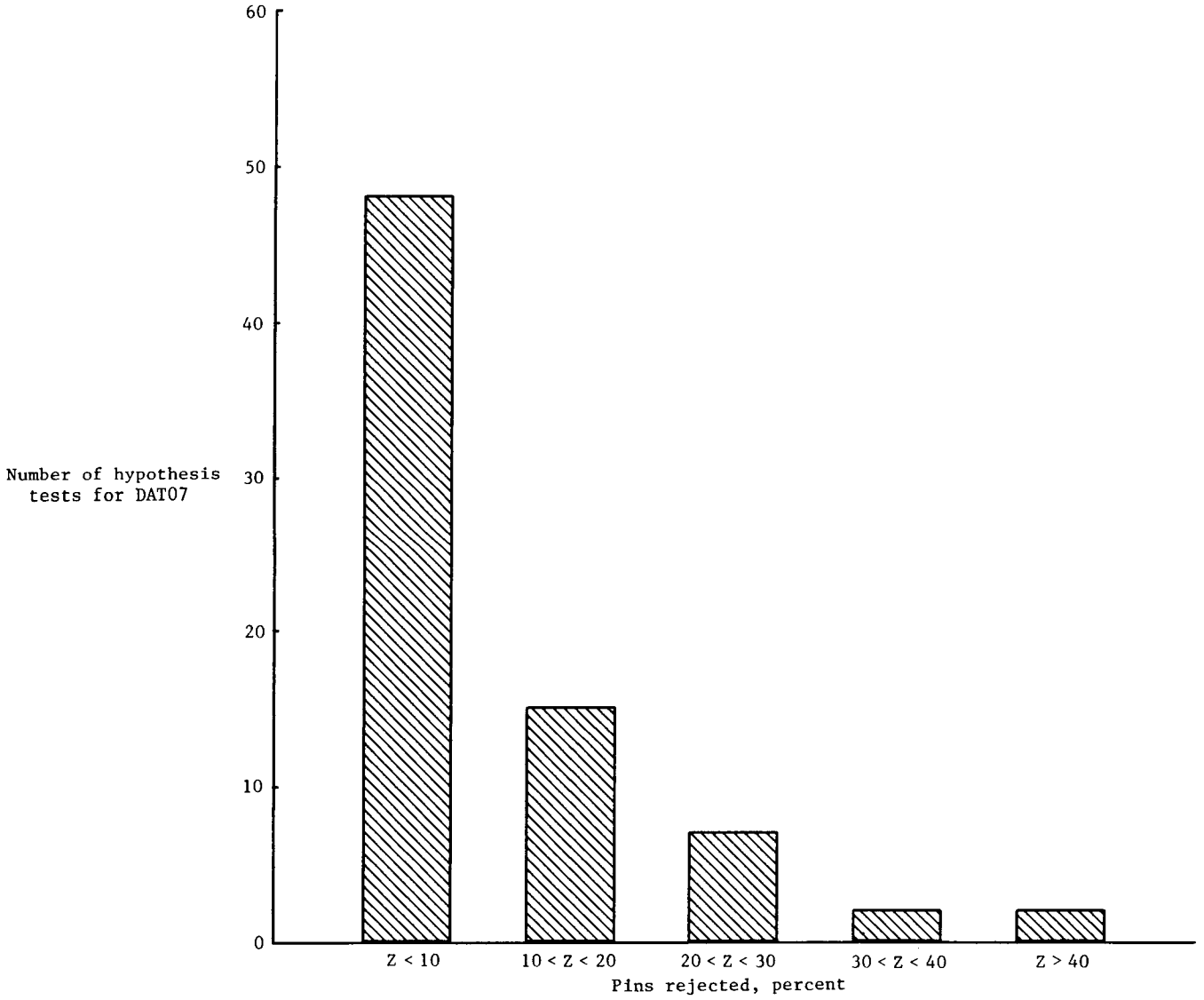

Figure 12. Test results, DAT07 vs all internal faults, for TBE.

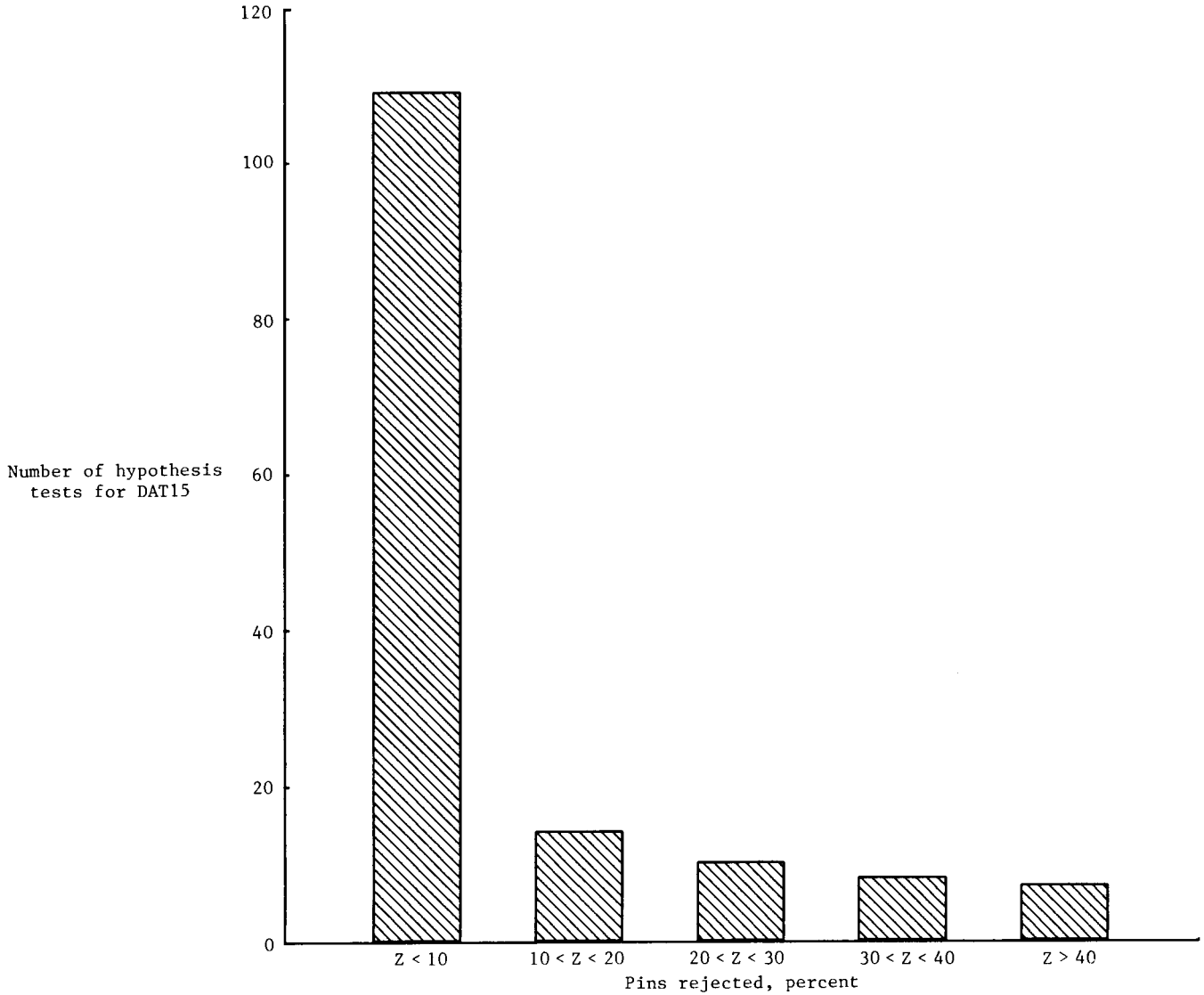

Figure 13. Test results, DAT15 vs all internal faults, for TBE.

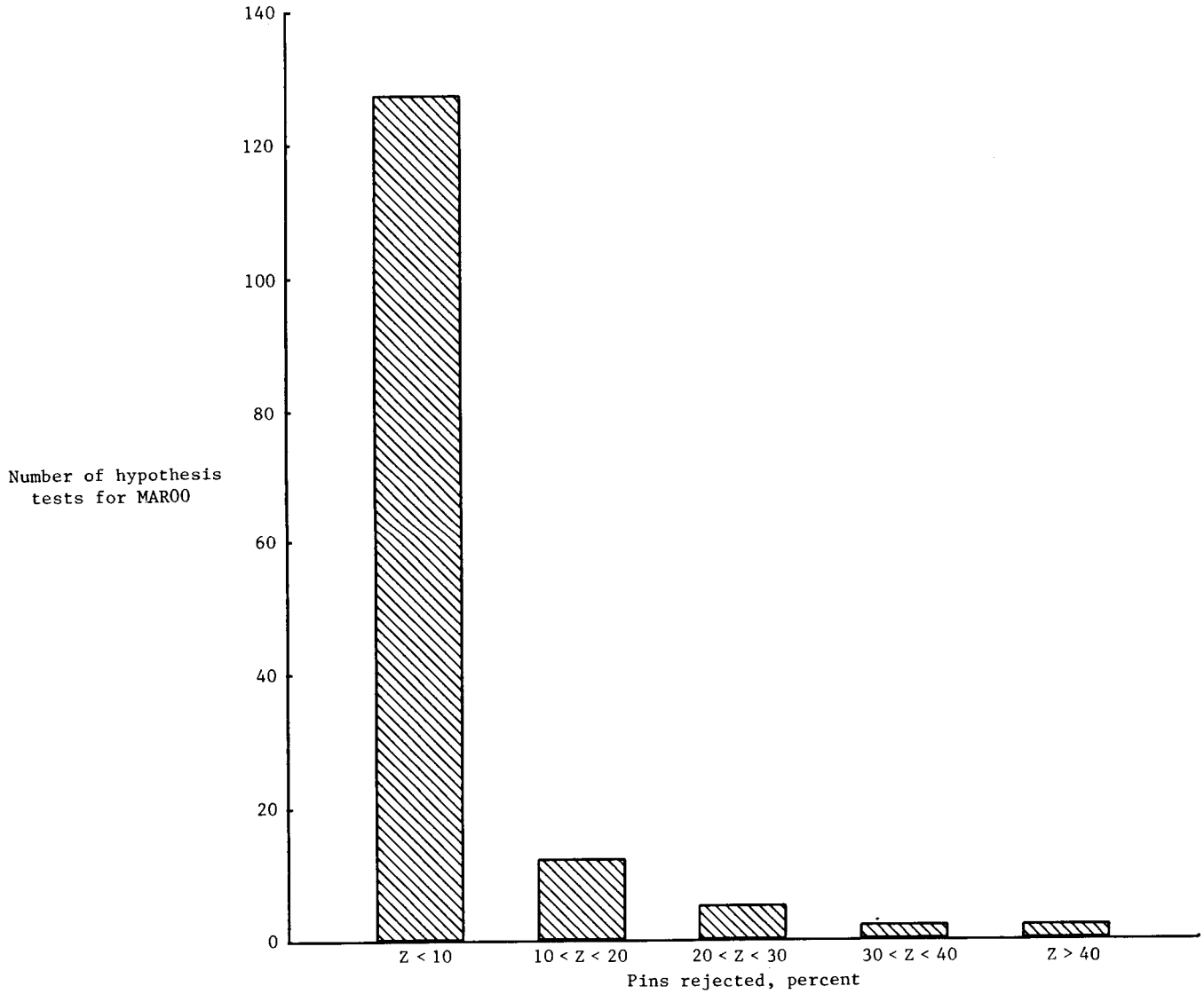

Figure 14. Test results, MAR00 vs all internal faults, for TBE.

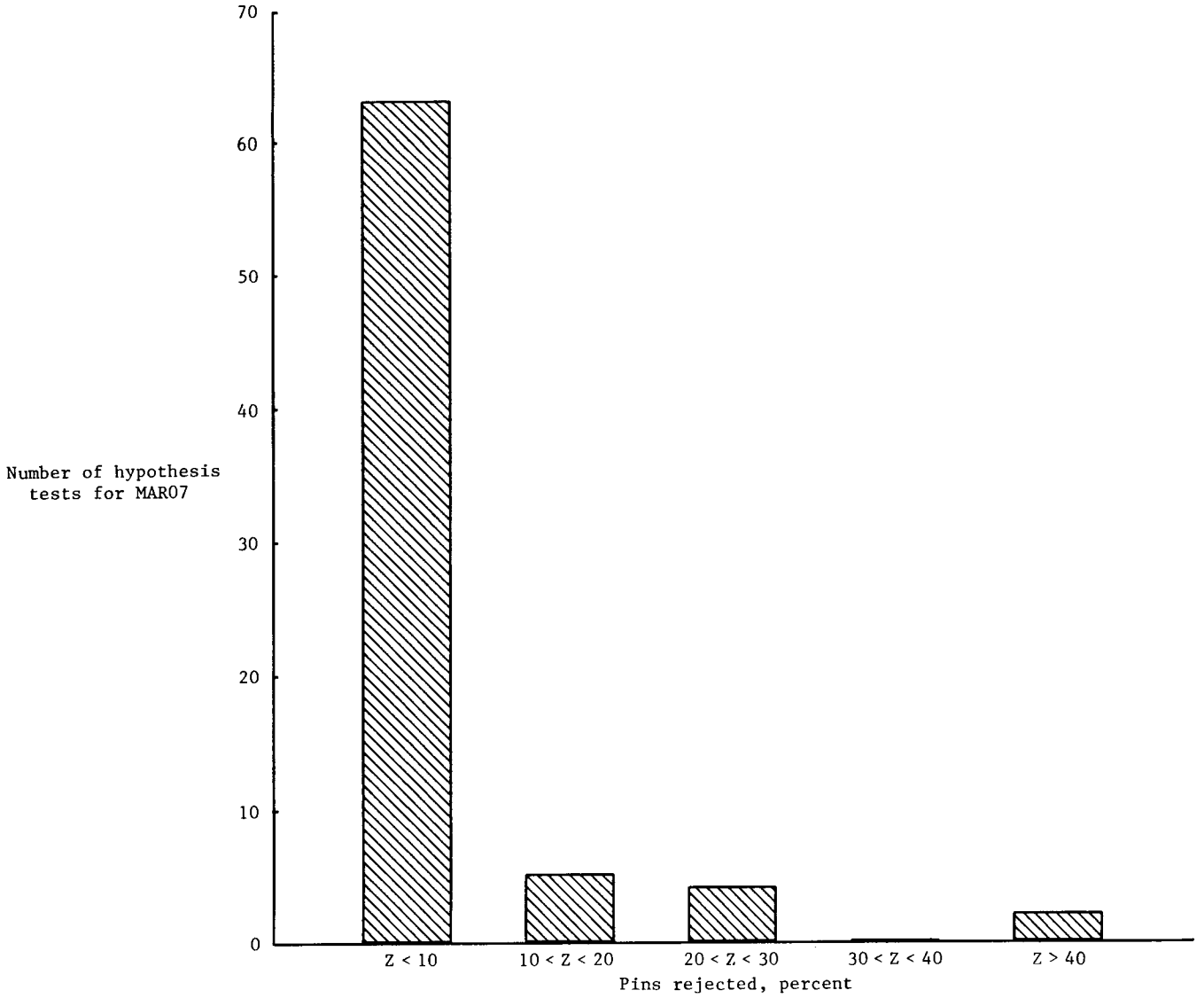

Figure 15. Test results, MAR07 vs all internal faults, for TBE.

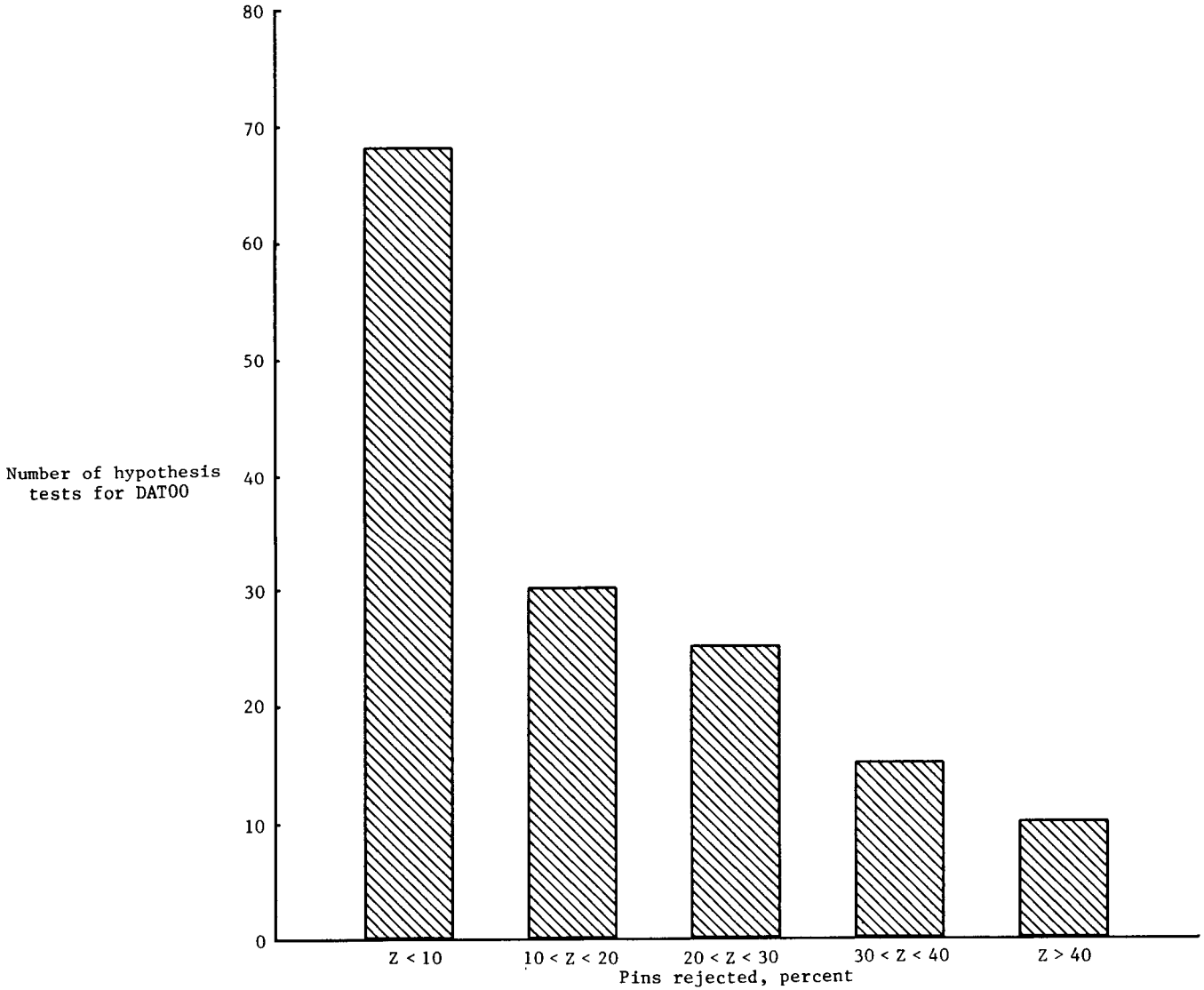

Figure 16. Test results, DAT00 vs all internal faults, for DE.

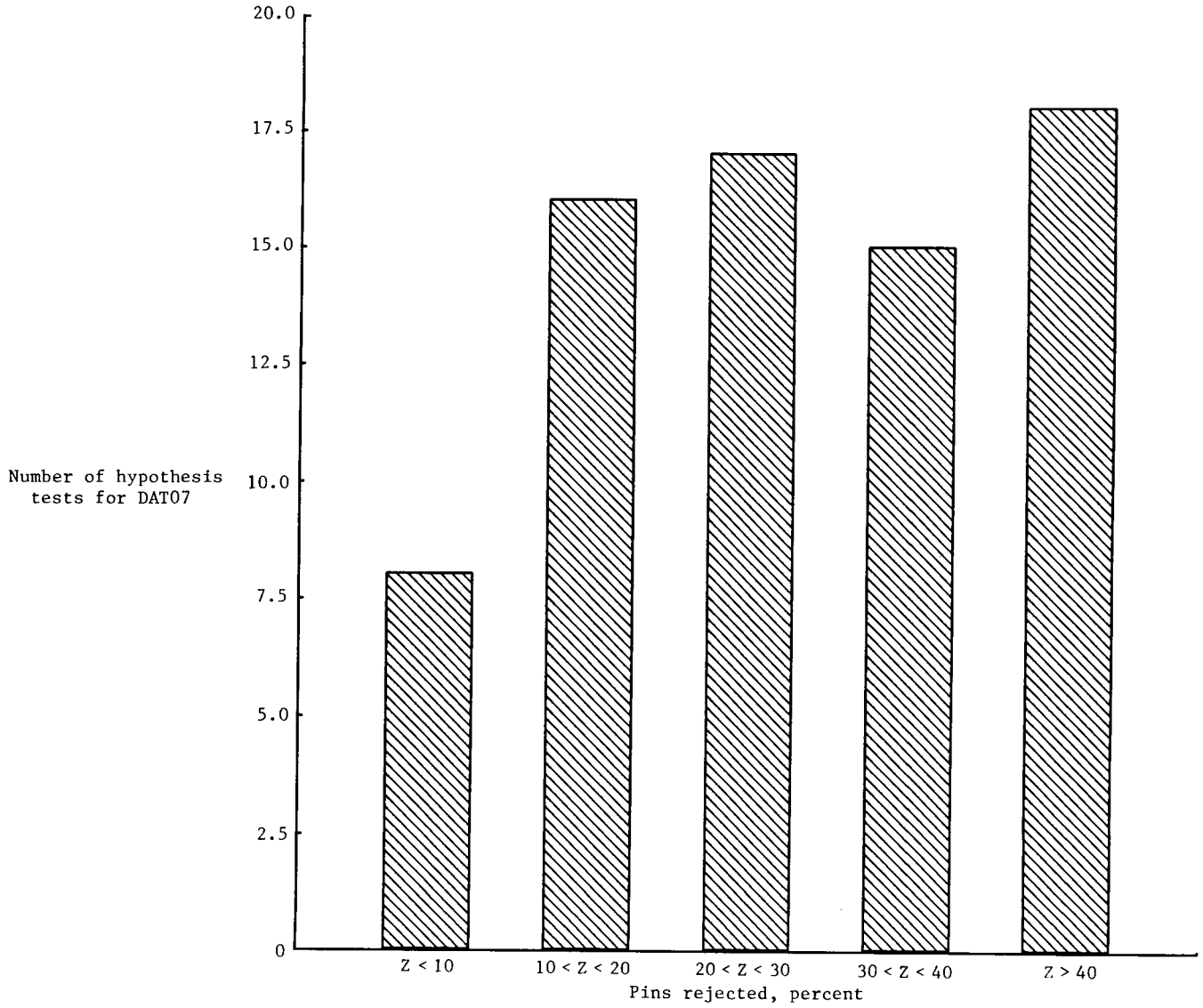

Figure 17. Test results, DAT07 vs all internal faults, for DE.

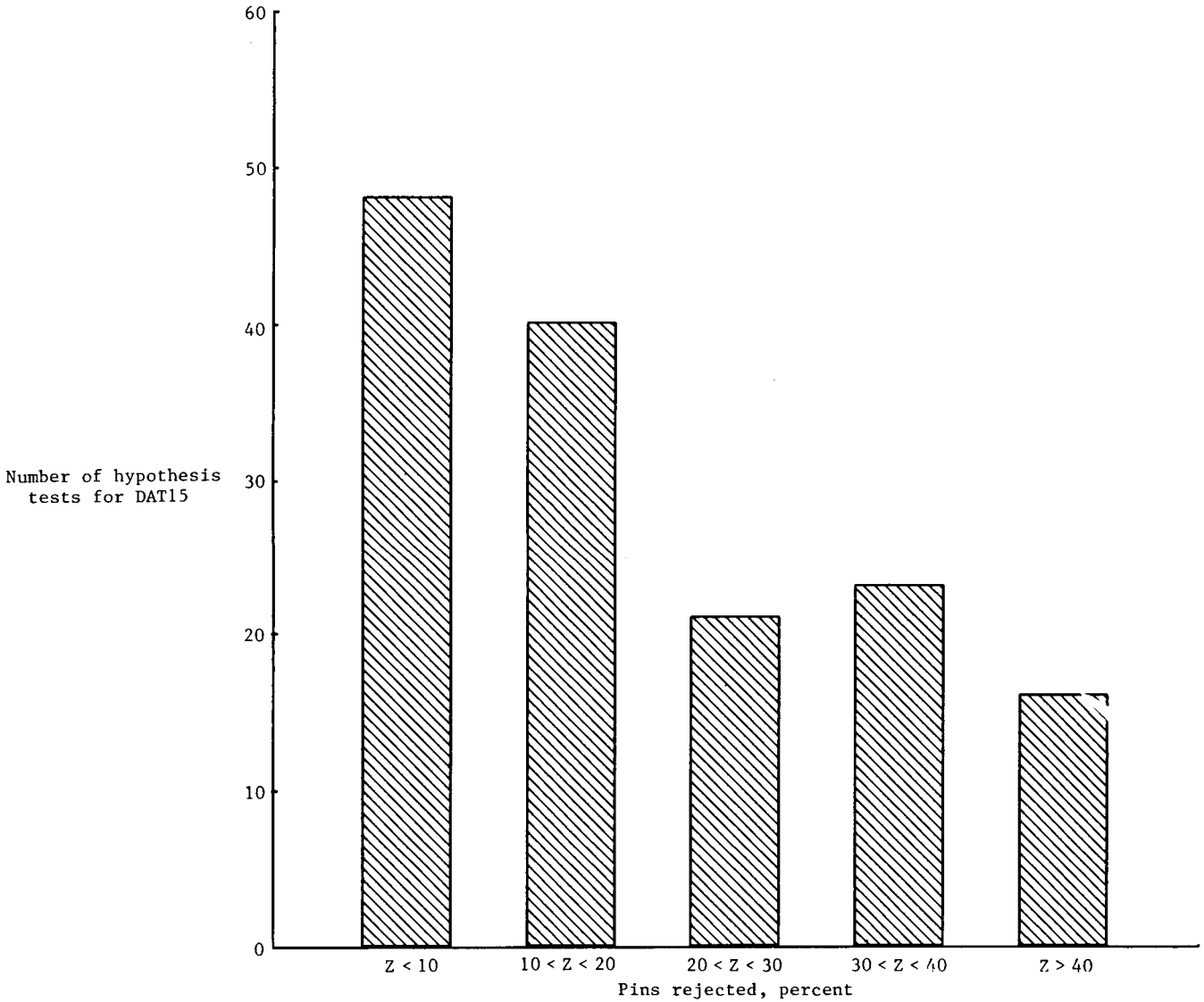

Figure 18. Test results, DAT15 vs all internal faults, for DE.

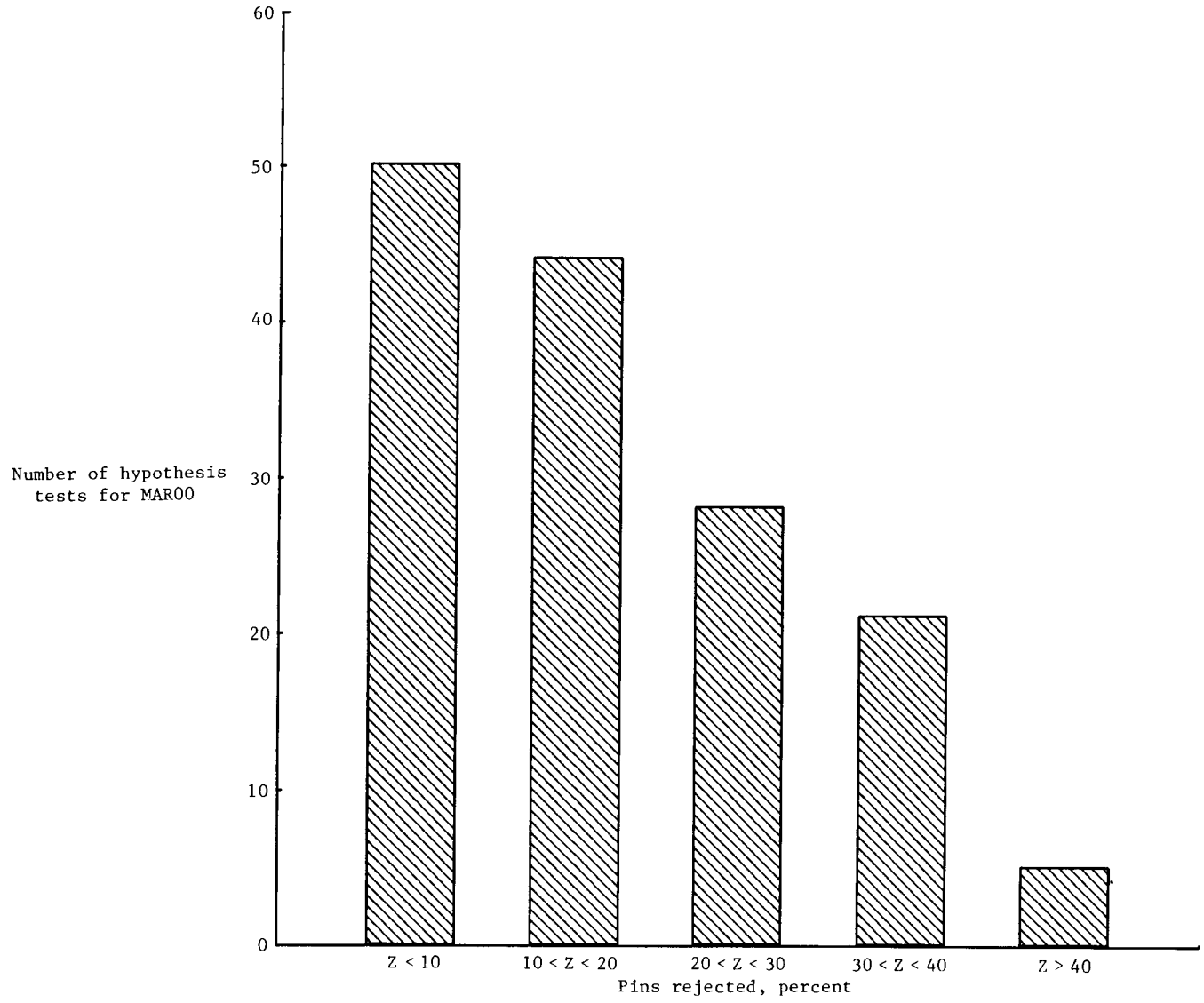

Figure 19. Test results, MAR00 vs all internal faults, for DE.

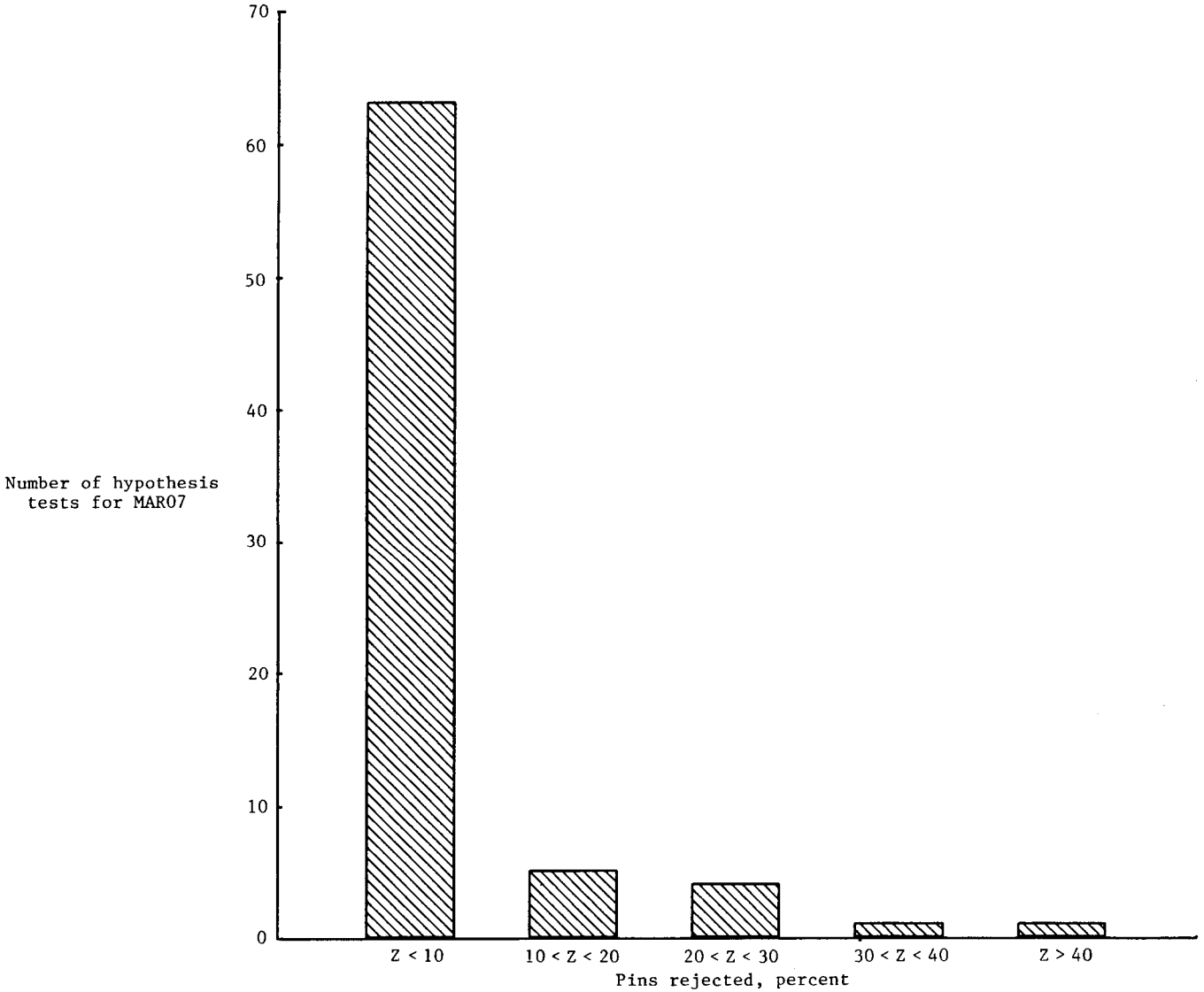

Figure 20. Test results, MAR07 vs all internal faults, for DE.

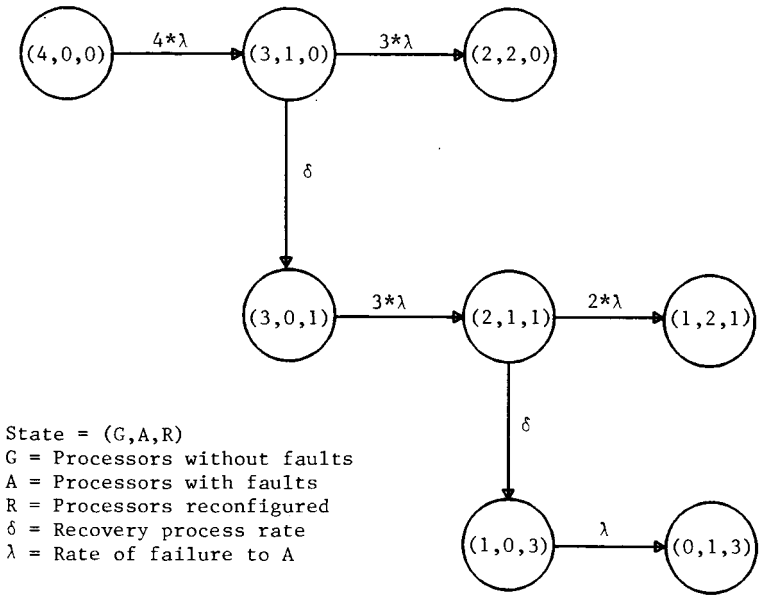

Figure 21. Model of failure arrival-recovery process.

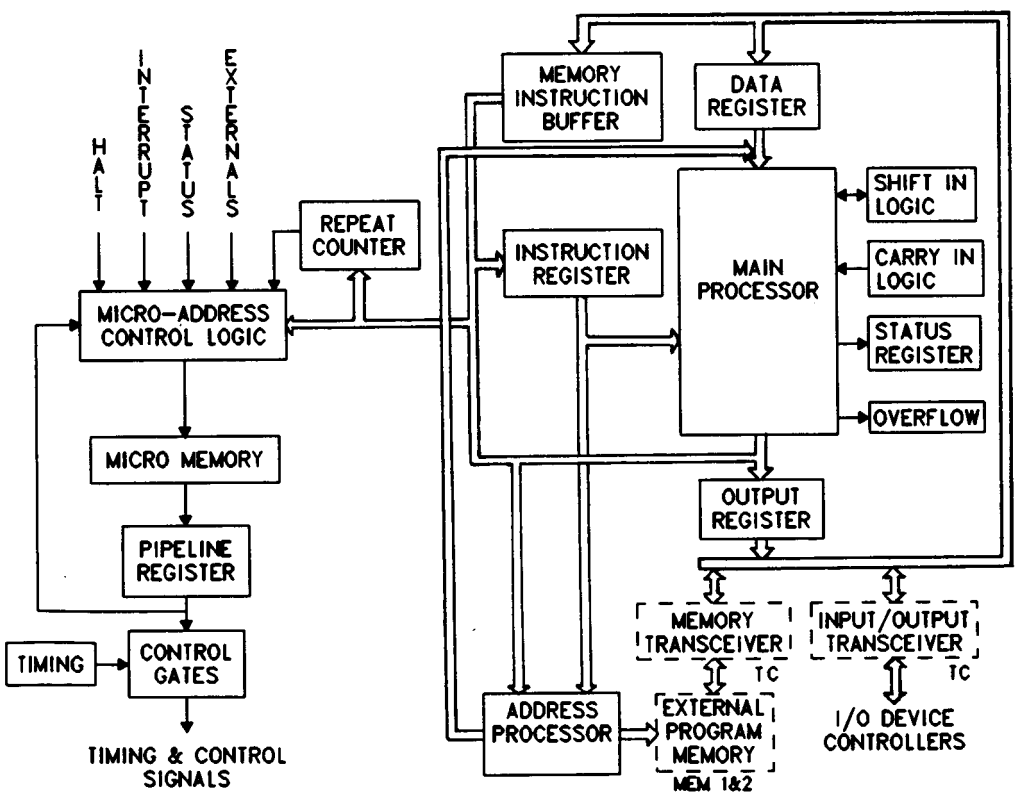

Figure 22. Block diagram of the BDX-930.

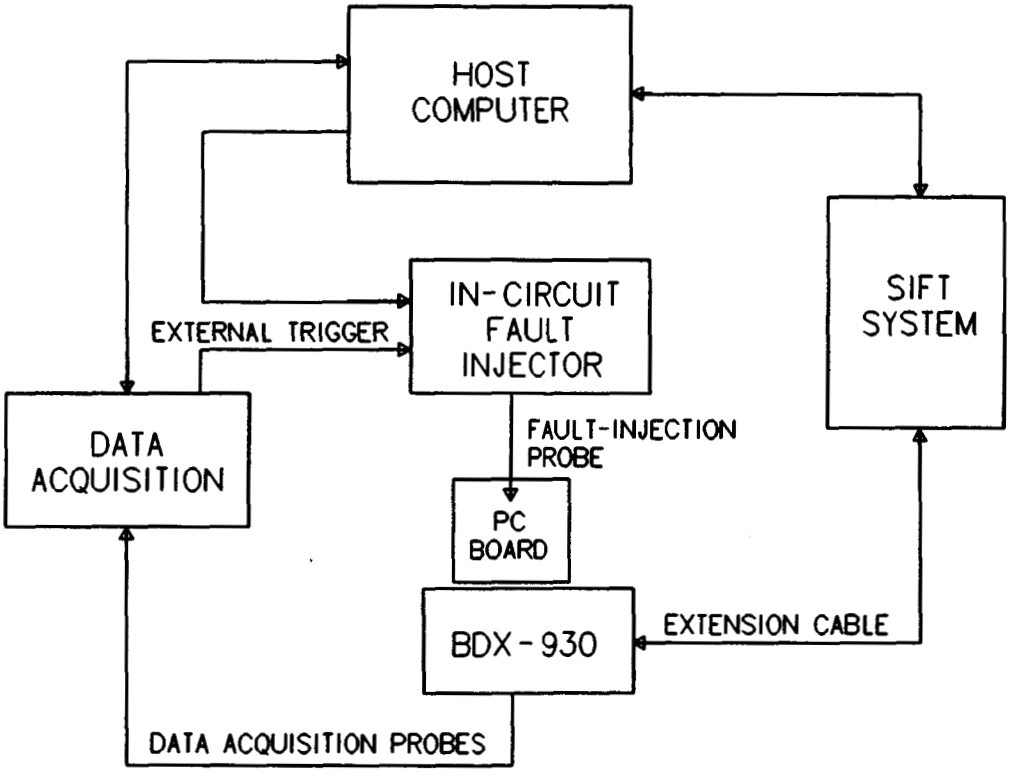

Figure 23. Experiment configuration.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TP-2738		

4. Title and Subtitle	5. Report Date
A Technique for Evaluating the Application of the Pin-Level Stuck-At Fault Model to VLSI Circuits	September 1987
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Daniel L. Palumbo and George B. Finelli	L-16269
9. Performing Organization Name and Address	10. Work Unit No.
NASA Langley Research Center Hampton, VA 23665-5225	505-66-21-01
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Washington, DC 20546-0001	Technical Paper
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Accurate fault models are required to conduct the experiments defined in validation methodologies for highly reliable fault-tolerant computers (e.g., computers with a probability of failure of 10^{-9} for a 10-hour mission). This paper describes a technique by which a researcher can evaluate the capability of the pin-level stuck-at fault model to simulate true error behavior symptoms in very large scale integrated (VLSI) digital circuits. The technique is based on a statistical comparison of the error behavior resulting from faults applied at the pin-level of and internal to a VLSI circuit. As an example of an application of the technique, the error behavior of a microprocessor simulation subjected to internal stuck-at faults is compared with the error behavior which results from pin-level stuck-at faults. The error behavior is characterized by the time between errors and the duration of errors. Based on this example data, the pin-level stuck-at fault model is found to deliver less than ideal performance. However, with respect to the class of faults which cause a system "crash," the pin-level stuck-at fault model is found to provide a good modeling capability.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
Fault tolerance Fault injection Recovery mechanisms Fault models	Unclassified—Unlimited Subject Category 38

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	44	A03

NASA FORM 1626 OCT 86 NASA-Langley, 1987

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171