NASA Contractor Report 3462

Measurement of Fault Latency in a Digital Avionic Mini Processor

John G. McGough and Fred L. Swern

CONTRACT NAS1-15946
OCTOBER 1981

NASA

NASA Contractor Report 3462

Measurement of Fault Latency in a Digital Avionic Mini Processor

John G. McGough and Fred L. Swern
Bendix Corporation
Teterboro, New Jersey

Prepared for
Langley Research Center
under Contract NAS1-15946

NASA
National Aeronautics
and Space Administration

Scientific and Technical
Information Branch

1981

TABLE OF CONTENTS

1.0 INTRODUCTION 8
1.1 Background 8
1.2 Objectives of the Study 8
1.3 Foreword 9

2.0 SUMMARY 10

3.0 FAULT MODELLING AND SELECTION 12
3.1 Fault Model 12
3.2 Method of Selecting Faults 13

4.0 DESCRIPTION OF EXPERIMENTS 14
4.1 Definition of Failure Detection 14
4.2 Definition of Failure Detection Coverage 14
4.3 Indistinguishable Faults and Effects on Coverage 16
4.4 Objectives of Experiments 18
4.5 Phase I Experiments 19
4.5.1 Fibonacci (FIB) 20
4.5.2 Fetch and Store (FETSTO) 21
4.5.3 Add and Subtract (ADDSUB) 22
4.5.4 Search and Compute (SERCOM) 23
4.5.5 Linear Convergence (LINCON) 24
4.5.6 Quadratic (QUAD) 25
4.6 Phase II Experiments 26
4.6.1 Self-Test 28

5.0 RESULTS OF EXPERIMENTS 30
5.1 Distribution of Faults 30
5.2 Phase I Experiments 30
5.2.1 FETSTO Experiment 30
5.2.2 ADDSUB Experiment 31
5.2.3 FIB Experiment 32
5.2.4 QUAD Experiment 33
5.2.5 SERCOM Experiment 34
5.2.6 LINCON Experiment 35
5.3 Phase II Experiments 36
5.3.1 Indistinguishable Fault Estimates 36
5.3.2 Self-Test Coverage 36
5.3.3 Gate-Level Faults 37
5.3.4 Component-Level Faults 38
5.4 URN Model Parameters 38
5.5 Accuracy and Confidence of Results 39
5.5.1 Phase I Results 39
5.5.2 Phase II Results 41
5.5.3 URN Model Results 41

TABLE OF CONTENTS (CONTINUED)

6.0 SUMMARY OF EXPERIMENTS 131
6.1 Phase I Experiments 131
6.2 Phase II Experiments 132
6.3 URN Model Distributions 132

7.0 ANALYSIS OF UNDETECTED FAULTS 137
7.1 Phase I Experiments 137
7.1.1 Undetected Faults in Phase I 137
7.2 Phase II Experiments 137
7.2.1 Undetected Faults in Phase II 138
7.3 Gate-Level Versus Component-Level Faults 139

8.0 UNIPROCESSOR BIT 141

9.0 URN MODEL 148
9.1 URN Model Description 148
9.2 Generalized URN Model 150
9.2.1 An Alternate Model 151
9.2.2 Examples 155
9.2.3 Comparison of Models 156

10.0 STATISTICAL ANALYSES 159
10.1 Introduction 159
10.2 Estimators for Self-Test Coverage 160
10.3 Estimators for Latency 161
10.3.1 Corrections for Indistinguishable Faults 162
10.4 Estimators for URN Model Parameters 162
10.5 Accuracy and Confidence of Coverage Estimates 164
10.6 Accuracy and Confidence of Latency Estimates 166
10.7 Accuracy and Confidence of URN Model Parameter Estimates . . . 168

11.0 EMULATION DESCRIPTION 175
11.1 BDX-930 Architecture 175
11.2 Description of the Emulator 177
11.3 Preprocessor and Postprocessor 181
11.4 Typical Circuit Representations 182
11.5 Summary of Emulation Characteristics 182

12.0 EXTENSION OF EMULATOR TO MULTIPROCESSOR SYSTEMS 200
12.1 Description of SIFT (see ref (4)) 200
12.2 SIFT Emulator 200
12.3 SIFT Fault Injection Experiments 201

13.0 CONCLUSIONS 204

14.0 RECOMMENDATIONS FOR FUTURE STUDIES 207

15.0 REFERENCES 208

LIST OF ILLUSTRATIONS

FIGURE	TITLE	PAGE
1	Flow Diagram for LINCON	29
2a, 2b	FETSTO - Gate-Level Faults.	50
2c, 2d, 2e	FETSTO - Gate-Level Faults by Partition	52
2f, 2g	FETSTO - Component-Level Faults	55
2h, 2i	FETSTO - Component-Level Faults by Partition.	57
3a, 3b	ADDSUB - Gate-Level Faults.	60
3c, 3d, 3e	ADDSUB - Gate-Level Faults by Partition	62
3f, 3g	ADDSUB - Component-Level Faults	65
3h, 3i	ADDSUB - Component-Level Faults by Partition.	67
4a, 4b	FIB - Gate-Level Faults	70
4c, 4d, 4e	FIB - Gate-Level Faults by Partition.	72
4f, 4g	FIB - Component-Level Faults.	75
4h, 4i	FIB - Component-Level by Partition.	77
5a, 5b	QUAD - Gate-Level Faults.	80
5c, 5d, 5e	QUAD - Gate-Level Faults by Partition	82
5f, 5g	QUAD - Component-Level Faults	85
5h, 5i	QUAD - Component-Level Faults by Partition.	87
6a, 6b, 6c	SERCOM - Gate-Level Faults.	90
6d, 6e, 6f	SERCOM - Gate-Level by Partition.	93
6g, 6h	SERCOM - Component-Level Faults	96

LIST OF ILLUSTRATIONS (CONTINUED)

FIGURE	TITLE	PAGE
6i, 6j	SERCOM - Component-Level Faults by Partition.	98
7a, 7b	LINCON - Gate-Level Faults.	100
7c, 7d, 7e	LINCON - Gate-Level Faults by Partition	102
7f, 7g	LINCON - Component-Level Faults	105
7h, 7i	LINCON - Component-Level Faults by Partition.	107
8a, 8b	URN Model FETSTO - Gate-Level Faults.	116
8c, 8d	URN Model FETSTO - Component-Level Faults	118
9a, 9b	URN Model ADDSUB - Gate-Level Faults.	120
9c, 9d	URN Model ADDSUB - Component-Level Faults	122
10a, 10b	URN Model FIB - Gate-Level Faults	124
10c, 10d	URN Model FIB - Component-Level Faults.	126
11	Detected Combined Faults vs No. of Executed Instructions (Gate-Level Faults).	134
12	Detected Combined Faults vs No. of Executed Instructions (Component-Level Faults)	135
13	Processor Architecture.	186
14	Basic Two Input and Gate Fault Model.	187
15	Typical Gate-Level Computation.	188
16a	Gate-Level D-Latch Model.	190
16b	Typical Functional D-Latch.	191
17a	Tri-State Bus	192
17b	Simulation of Tri-State Bus	192
18a	Gate-Level to Register-Level Conversion Algorithm	193
18b	Register-Level to Gate-Level Conversion Algorithm	194
19	IC 175.	195
20	IC 151.	196
21	IC 153.	197
22	IC 158.	198
23	SIFT System	203
24	SIFT Computer	203

LIST OF TABLES

TABLE	TITLE	PAGE
1	Failure Rates of Partitions of the CPU.	44
2	Number of Faults Injected	45
3a	Phase I Experiments - Number of Gate-Level Faults Injected by Partitions.	46
3b	Phase I Experiments - Number of Component-Level Faults Injected by Partitions	47
4a	Phase II Experiments - Number of Gate-Level Faults Injected by Partitions.	48
4b	Phase II Experiments - Number of Component-Level Faults Injected by Partitions	48
5	FETSTO Latency Data	49
6	ADDSUB Latency Data	59
7	FIB Latency Data.	69
8	QUAD Latency Data	79
9	SERCOM Latency Data	89
10	LINCON Latency Data	89
11	Summary of Phase I Results.	109
12	Self-Test Data.	110
13	Fault Detection by the Individual Tests Comprising Self-Test	111
14	Self-Test Coverage Summaries by Partition	112
15	URN Model Distributions and Parameter Estimates	114
16	Error Covariance Matrix Elements for URN Model Estimates	128
17	Inverse Error Covariance Matrix Elements for URN Model Estimates	129
18	Intermediate URN Model Parameter Estimates.	130
19	Instruction Mix vs Detection in Phase I Experiments	136
20	Failure Rates of Critical Components.	146
21	Inflight Bit Test Procedures.	147
22	Error Ellipse for a Confidence Level of $\gamma = .95$	173
23	Maximum Error Versus Sample Size and Confidence Level	174
24	Parallel Operation of the BDX-930 Processor	183
25	Components of the BDX-930 CPU	184
26	Microcircuits and Equivalent Gate Count	185
27	Value of Nodes.	189
28	Emulator Characteristics.	199
29	Comparison of Latency Estimates	206

1.0 INTRODUCTION

1.1 Background

The advent of redundant and highly reliable airborne digital systems has raised a number of critical issues in connection with the ability of such systems to detect, isolate and recover from hardware faults. In such systems fault detection is a critical factor in achieving system reliability. The present study is essentially an investigation of the nature of faults and the dynamics of fault propagation and detection in digital systems.

Most airborne systems, present and projected, employ comparison-monitoring, self-test or a combination of both techniques to achieve the requisite detection and isolation capability. One of the problems of fault detection by either technique, is that a fault may not manifest itself in a comparison-monitored variable or at an accessible output of a component until the faulted component is exercised by a suitable combination of input or internal state. As a consequence, the fault may not be detected by self-test or, in the case of comparison-monitoring, the fault may remain latent for long periods of time. Prolonged latency in a redundant system can reduce survivability since such faults effectively increase the time-on-risk.

In an effort to determine the dynamics of fault propagation and detection in a digital computer, NASA-Langley Research Center sponsored a pilot program entitled "Modeling of a Latent Fault Detector in a Digital System" (ref. 1), in 1978. The objectives were to study how software reacts to a fault, to account for as many variables as possible affecting detection and to forecast a given software program's detecting ability prior to computation. A series of fault injection experiments were conducted using an emulation of a small, idealized processor with a very limited instruction set. The results of the study were surprising since they contradicted the prevailing belief that most hardware faults cause catastrophic computational errors. In fact, the study showed that a significant proportion of faults remained latent after many repetitions of a program. However interesting these results were, they were greeted with a healthy skepticism. It was not clear, for instance, that similar results could be obtained for a real processor, preferably one used in actual airborne applications. As a consequence, it was decided to extend the study to include a real avionics processor.

1.2 Objectives of the Study

The present study was based on the premise that a gate-level emulation of an avionics, airborne processor was available. Prior to award of contract the Bendix Research Laboratories and the Bendix Flight Systems Division had developed a gate-level emulation of the Bendix BDX-930 digital computer. This computer is used in a number of flight control and avionics programs, notably on the AFTI F-16 FBW system and SIFT. SIFT (Software Implemented Fault Tolerance) is a fault tolerant digital computer system developed by SRI, International with Bendix, Flight Systems Division, as a major subcontractor. A description of the BDX-930 and its emulation will be given in subsequent sections.

Underlying the entire study was the intention to demonstrate that gate-level emulation was a viable and practical tool for coverage measurement failure modes and effects analyses of digital systems. It was for this reason that so much effort was expended in developing a fast and efficient emulator. Admittedly, there are many gate-level emulations available that emulate to a greater level of detail and, perhaps, even with greater fidelity then the one used in the present study. An informal survey of such emulators indicated that, except for hardware emulators, the run time was prohibitive, being on the order of 500,000 to 1,000,000 slower than the emulated processor.

As indicated previously, a primary objective of the present study is to ascertain whether and to what extent the results of the pilot study apply to a real avionics processor. Specifically,

- Given a set of software programs ranging from a simple "fetch and store" to a complicated, multi-instruction algorithm, inject a single fault, selected at random, and observe the time to detection assuming that detection occurs whenever there is a difference between the computed outputs of the faulted and non-faulted processors executing the same program. Determine differences in detection time when faults are injected at the gate-level and component-level.
- Based upon derived empirical latency distributions, develop and validate a model of fault latency that will forecast a software program's detecting ability.

The following additional objectives were added to those of the pilot study:

- Given a typical avionics self-test program inject faults at both the gate-level and component-level and determine the proportion of faults detected.
- Determine why undetected faults were undetected.
- Recommend how the emulation of the BDX-930 can be extended to multi-processor systems such as SIFT.
- Determine the proportion of faults detected by a miniprocessor BIT (built-in-test program) irrespective of self-test.

1.3 Foreword

The authors would like to express their appreciation to:

NASA-Langley Research Center who conceived and initiated the study; NASA Project Engineer Salvatore Bavuso whose advice and encouragement were indispensable and made the task a pleasant one; Bendix Research Laboratories who did most of the development of the emulator; Dr. Allen White of Kentron International for his critique of the statistical analyses; Prof. Mario Barbacci of Carnegie-Mellon University for his advice and assistance in developing the emulator.

Use of trade names of manufacturers in this report does not constitute an official endorsement of such products or manufacturers, either expressed or implied, by the National Aeronautics and Space Administration.

2.0 SUMMARY

- A gate-level emulation of the Bendix BDX-930 digital computer was developed prior to the present study for the purpose of analyzing failure modes and effects in digital systems. The run time of the emulation was 25,000 times slower than the BDX-930 when hosted on a PDP-10.
- Six software programs were emulated and faults were injected at both the gate-level and pin-level (i.e., component-level). The resultant computed outputs were compared with those of a non-faulted computer executing the same program. A fault was considered detected when these outputs differed. The results showed that:
  - Most detected faults are detected in the first repetition. Subsequent repetitions do not appreciably increase the proportion of detected faults.
  - A large proportion of faults remained undetected after as many as 8 repetitions of the program, e.g., 60% at the gate-level.
  - Component-level faults are easier to detect than gate-level faults. For example, after 8 repetitions, the proportion of undetected faults were

GATE-LEVEL	COMPONENT-LEVEL
61.7%	35.5%
58.2%	28%
59.5%	32.3%

  for the program FETSTO, FIB and ADDSUB, respectively.

- The results of the study corroborate the findings of the pilot study of (ref. 1). This was surprising considering that the pilot study used an emulation of a very simple processor. As an illustration, the pilot study indicated that, after 8 repetitions, the proportion of undetected faults were

64.4%
53.7%
44.9%

  for FETSTO, FIB and ADDSUB, respectively.

- The Urn Model, for forecasting fault latency, produced distributions that were in close agreement with the empirical distributions. However, the rationale for the model should be analyzed further.
- A self-test program of 2000 executable instructions was expressly designed for the study. The designer was given the single requirement that fault coverage should be at least 95%. The resultant test consisted of 241 separate subtests for the purpose of exercising the entire instruction set of the BDX-930.

The results indicated that there is a significant difference in coverage of gate-level versus component-level faults. For example,

gate-level coverage = 86.5%

component-level coverage = 97.9%

- Only 48% of all detected faults were detected by a subtest. The remaining detected faults were detected because the first subtest was not computed.
- Most of the subtests were redundant, i.e., only 46 of 241 subtest actually detected a fault.
- 62% of all detected faults were detected by the first 23 subtests.
- A large proportion of "don't care" (i.e., indistinguishable) faults were injected. These proved to be exceedingly difficult to identify.
- The micromemory prom contained the largest proportion of undetected faults.

- The emulation can easily accommodate the SIFT system but with a 7-fold increase in run time.

3.0 FAULT MODELLING AND SELECTION

3.1 Fault Model

At the present time there is little or no data available regarding either the mode or frequency of failures of MSI or LSI devices. Despite this deficiency of data, failure modes and effects analyses are regularly performed for avionics and flight control systems (a typical analysis is described in Section 11). The conventional approach is to assume a set of failure modes for each device. These are usuallly restricted to faults at single pins although, occasionally, multiple faults may be considered. In most cases the failure rate of a device is assumed to be equally distributed over the pins or over the set of postulated failure modes. Except for special devices, faults are assumed to be static, being either S-a-0 or S-a-1.

The point to be made here is that failure modes and their rate of occurrence are necessarily conjectural and the credibility of the present study suffers no less from this deficiency of data then the conventional analysis. The authors emphasize that the emulation approach does not solve this problem.

In the present study the following assumptions are made regarding failure modes:

- Every device can be represented, from the standpoint of performance and failure modes, by the manufacturer-supplied, gate-level equivalent circuit.
- Every fault can be represented as either a S-a-0 or S-a-1 fault at a gate node.
- The failure rate of the device is equally distributed over the gates of the equivalent circuit.
- The failure rate of a gate is equally distributed over the nodes of the gate.
- S-a-0 and S-a-1 faults are equally likely.
- Memory faults are exclusively faults of single bits.
- A memory fault is the complement of its non-faulted state.

Faults are injected into all devices except the main memory. In the case of the microprogram memory, which is emulated at the functional level, faults are injected into the memory cells where they remain active for the duration of the test. Faults are injected at an input or output gate node, and also remain active for the duration of the test. When a fault is injected at an output node it is allowed to propagate to all nodes and devices that are physically connected to the failed node. When a fault is injected at an input node, it does not propagate back to the driving node. This strategy provides a wider variety of failure modes than would otherwise be possible if propagation were allowed. The fault model, although conjectural at the present time, can be updated as fault data becomes available. The proposed model provides a simple, automatic and consistent method of generating faults. The resultant fault set includes a rich assortment of static and dynamic (i.e., data-dependent) faults.

3.2 Method of Selecting Faults

The method of selecting faults is implicit in the fault model. Explicitly,

- Each device is assigned a failure rate.
- The failure rate is equally distributed over the gates of the gate-level representation.
- The failure rate of each gate is equally distributed over the nodes of the gate.
- The failure rate of each node is equally distributed over S-a-0 and S-a-1 faults.
- As a result of this procedure, each S-a-0 and S-a-1 fault is assigned a probability of occurrence equal to the prescribed failure rate. The resultant fault set is then randomly sampled with each fault weighted by its probability of occurrence. It is noted that, according to this procedure, faults in devices with high failure rates will be selected more frequently than faults in devices with lower failure rates.

The above procedure does not distinguish between gate-level and component (i.e., pin)-level faults except by probability of occurrence; the method automatically assigns failure rates to pins. However, a different selection procedure was employed for component-level faults. For these faults it was assumed that:

- The failure rate of each device is equally distributed over the pins.

While this assumption violates the prescribed fault model it is consistent with the conventional method of estimating fault detection coverage by simulating faults in actual hardware. As a consequence, all component-level detection estimates obtained in the study are estimates that would be obtained by proponents of this approach.

4.0 DESCRIPTION OF EXPERIMENTS

4.1 Definition of Failure Detection

In the present study, fault coverage and latency estimates are obtained by employing two conventional techniques of failure detection: comparison-monitoring and self-test.

In comparison-monitoring a set of computed variables is compared with a corresponding set computed in another processor. If it is arranged that both processors operate on identical inputs and are closely synchronized, then any difference in a computed variable signifies that one of the processors has failed. In practice each processor executes an algorithm which compares the appropriate variables and signals a discrepancy when such exists. In the present study this algorithm was omitted; a fault is considered to be detected if a difference between corresponding variables exists irrespective of the ability of either processor to recognize the difference or signal the discrepancy. Thus, the fault coverage obtained from the study is somewhat more optimistic than would be obtained in practice.

In self-test, on the other hand, each component of the processor is exercised by a set of computations designed specifically to test that component. The results of each computational set are compared with pre-stored values and any difference signifies that the fault was detected. In practice, and in the study, the processor increments a register after the successful completion of each test and before proceeding to the next test. If the test is not successful the program exits. After an interval of time equal to the maximum time to complete the program, the contents of the counter are decoded. If the value exactly equals that total number of tests, the fault was not detected. Otherwise the fault was detected.

It is emphasized that "failure detection", as it is used in the present study, means almost exactly what it means in an actual airborne avionic system. This is in marked contrast to the commonly employed alternate approach of assuming that a failure is detected whenever the effect to the failure reaches an accessible bus or register, even though the program may not be interrogating these devices at that time.

In the following paragraphs a description is given of the actual computations involved in the experiment with particular emphasis on the explicit definition of "failure detection" in each instance.

4.2 Definition of Failure Detection Coverage

We assume that a test procedure is given for detecting failures of a component, C. Each failure mode of C will require a non-zero time for detection. By considering all failures of C and all combinations of inputs and internal states of C, we obtain in principle, if not in practice, a probability density function for time-to-detect, which is measured from the onset of the failure to the time of detection.

Denoting this density by pdf (τ) where

τ = time-to-detect = latency time

we define

Test Coverage

$$1)\quad 1 - \alpha(\tau) = \int_0^{\tau} \text{pdf}(X)dX$$

= probability of detecting a failure of C in the interval $0 \leqq t \leqq \tau$.

Observe that, according to this definition, test coverage is a function of latency time. The definition can be extended to all devices of the computer as follows:

Subdivide the computer into mutually exclusive components $C_1, C_2,\ldots,C_k$ with failure rates $\lambda_1, \lambda_2,\ldots,\lambda_k$, and test coverages $1 - \alpha_1(\tau), 1 - \alpha_2(\tau),\ldots, 1 - \alpha_k(\tau)$, respectively.

Set $\text{pdf}_i(\tau)$ = probability density for time-to-detect failures of C_i, $i = 1, 2,\ldots,k$.

Then the pdf for all failures of the computer is

$$2)\quad \text{pdf}(\tau) = \sum_{i=1}^{i=k} \frac{\lambda i}{\lambda} \text{pdf}_i(\tau)$$

where $\lambda = \lambda_1 + \lambda_2 + \ldots. + \lambda_k$.

Test coverage of the whole computer is then

$$3)\quad 1 - \alpha(\tau) = \sum_{i=1}^{i=k} \frac{\lambda i}{\lambda} (1 - \alpha_i(\tau)).$$

The method of selecting faults, described in Section 3, is consistent with this definition.

From (3) we obtain

$$4) \quad \alpha(\tau) = \sum_{i=1}^{i=k} \frac{\lambda i}{\lambda} \alpha i(\tau) \text{ , as expected.}$$

One of the objectives of the present study is to obtain estimates of the probability density function, pdf (τ). These estimates are presented in Section 8.

4.3 Indistinguishable Faults and Effects on Coverage

During the development of the emulator it became apparent that a significant proportion of components had no affect whatsoever on the digital process. For the most part, these components are associated with unused pins, e.g., a complementary output of a flip-flop. However, there are other components whose lack of effect are not as obvious as, for example, a component that only affects the process when it is faulted. Certain micromemory bits are in this category. In order to distinguish between these categories of faults we are lead to the following informal definitions:

A fault that has no affect on the computational process is indistinguishable. All other faults are distinguishable.

We note that a distinguishable fault has the property that there exists a software program the output of which differs from that of the same program executed by an identical but non-faulted processor.

Effects on Coverage

The presence of indistinguishable faults can lead to erroneous and misleading estimates of coverage. In theory, indistinguishable faults should be disqualified from the emulation or from the fault selection process. This is consistent with the definition of coverage which implicitly assumes that faults are distinguishable. Unfortunately, in order to disqualify indistinguishable faults from the emulation or from the fault selection process they must be first identified and this is a non-trivial task because of the large number of possible faults. The approach taken in this study was to select faults irrespective of their distinguishability properties and analyze only those faults that were undetected by Self-Test. The proportion of indistinguishable faults from this set was then used as an estimate over all faults.

We now indicate, briefly, how indistinguishable faults affect coverage.

If

γ = proportion of components yielding indistinguishable faults

and

$1 - \alpha$ = coverage of distinguishable faults

then

$1 - \alpha$ = desired coverage

and

5) $(1 - \alpha)(1 - \gamma)$ = coverage when indistinguishable faults are counted as undetected. We note, incidentally that

6) $(1 - \alpha)(1 - \gamma) + \gamma$ = coverage when indistinguishable faults are counted as detected.

The estimate of (5) will be obtained if indistinguishable faults are not disqualified. Then, coverage estimates will be in error by the factor, $1-\gamma$.

In the more general case it may be more convenient to estimate the proportion of indistinguishable faults by partition since the affect on coverage is a function of the relative failure rate of the partition.

Let λ_i = failure rate of Partition #i, i = 1, 2, ..., 6.

γ_i = proportion of indistinguishable faults in Partition #i.

$1 - \alpha_i$ = coverage of distinguishable faults in Partition #i.

$\lambda = \lambda_1 + \lambda_2 + ... + \lambda_6$ = total failure rate.

From the previous section, if all faults are distinguishable then coverage is given by

$$7) \quad 1 - \alpha = \sum_{i=1}^{6} \frac{\lambda i}{\lambda} (1 - \alpha_i)$$

If, however, indistinguishable faults are counted as undetected then the coverage actually obtained is

$$8)\quad 1 - \alpha = \sum_{i=1}^{6} \frac{\lambda i}{\lambda} \quad (1 - \alpha_i)\ (1 - \gamma_i).$$

We note that, if indistinguishable faults are disqualified, the true coverage is

$$9)\quad 1 - \alpha = \frac{\sum_{i=1}^{6} \lambda_i\ (1 - \alpha_i)\ (1 - \gamma_i)}{\sum_{i=1}^{6} (1 - \gamma_i)\ \lambda_i}$$

From (8) it can be seen that the required accuracy of an estimate of γ_i depends upon the relative failure rate, λ_i/λ. If λ_i is sufficiently small then the effect of an inaccurate estimate of γ_i is negligible.

4.4 Objectives of Experiments

Most airborne systems, present and projected, employ conparison-monitoring, self-test or a combination of both to achieve the requisite detection and isolation capability. One of the problems of fault detection, by either method, is that a fault may not manifest itself at either a comparison-monitored variable or at an accessible output of self-test until the faulted component is suitably exercised. As a consequence, faults can remain latent for long periods of time. This is the significance of latency time, τ, in the definition of test coverage of Section 4.2.

One of the objectives of the experiments is to estimate τ for a variety of computations including self-test. The Phase I experiments consist of six software programs ranging from a simple fetch and store to a complicated multi-instruction, linear convergence algorithm. Using comparison-monitoring the probability distribution of τ will be estimated for each of the six programs and the interdependence of these distributions and the number and type of instructions will be ascertained.

The Phase II experiments utilize a typical avionics system self-test program which consists of 241 separate, sequential tests. The program consists of 2000 executed instructions which requires an execution time of 3 milliseconds on the BDX-930.

In practice, the only measure of failure detection coverage when applied to a self-test program is whether or not the fault is detected before the normal completion of the self-test program. In particular, latency time has little or no significance in this context. Nevertheless, an equivalent latency time was estimated for self-test by tabulating the number of the test that actually detected the fault, the tests being executed in the order 1, 2, ..., 241.

A secondary objective of the experiment was to corroborate the results of (ref. 1) which utilized the same Phase I program executed on an idealized, "very simple processor".

4.5 Phase I Experiments

This phase consisted of six programs each of which was coded in the assembly language of the BDX-930. For the purpose of comparison with the experiments performed in (ref. 1) the instructions of the BDX-930 were primarily restricted to the following set:

LOAD

STORE

ADD

SUBTRACT

BRANCH

In the following descriptions only the set of computations labelled "compute" were performed by the target BDX-930 CPU; all other computations, selections, comparisons, etc. were performed by the emulation host computer Executive. Needless to say, there were no failures in these latter computations.

When the non-failed processor completed a computation* and before the start of the next computation the Executive recomputed all initializing variables and stored them in the appropriate locations of the scratchpad memory.

In the parallel mode of operation, when 36 computers are simultaneously being emulated, the initializing variables are stored simultaniously in the 36 copies of the scratchpad memories.

* In the parallel mode of operation one of the emulated processors is non-faulted and, as a consequence, the end of its computation cycle can be determined from its program counter.

4.5.1 Fibonacci (FIB)

a. Procedure

T0) Select integers A, B, at random from the interval

$$-2^9 + 1 \leq x \leq 2^9 - 1.$$

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store A, B in successive locations of memory.

T3) Compute and store in successive locations of memory

$$S_1 = A + B$$

$$S_2 = B + S_1$$

$$S_3 = S_1 + S_2$$

$$\vdots$$

$$S_8 = S_6 + S_7.$$

T4) When the non-failed processor completes its last instruction compare S_1, S_2, ..., S_8, term by term, in both the non-failed and failed processors. If S_k is the first variable to miscompare set L = K (L = latency period). If all S_k compare (undetected failure), set L = 0.

b. Instruction Set

During a typical computation the following instructions were executed:

INSTRUCTION	FREQUENCY
LOAD	2
STORE	1
ADD	3
BRANCH	4
CLEAR	1

4.5.2 Fetch and Store (FETSTO)

a. Procedure

T0) Select 8 integers, A_k, at random from the interval

$$-2^{15} + 1 \leq A_k \leq 2^{15} - 1 .$$

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store the A_k in successive locations of memory.

T3) Compute:

Fetch the A_k and store in successive locations of memory.

T4) If the re-stored value of A_k is denoted by S_k then, when the non-failed processor completes its last instruction compare S_1, S_2, ..., S_8, in both the non-failed and failed processors. If S_k is the first variable to miscompare set L = K. If all S_k compare set L = 0.

b. Instruction Set

During a typical computation the following instructions were executed:

INSTRUCTION	FREQUENCY
LOAD	1
STORE	2
SUBTRACT	1
BRANCH	2

4.5.3 Add and Subtract (ADDSUB)

a. Procedure

T0) Select 8 integers, A_k, at random from the interval

$$-2^{14} + 1 \leq A_k \leq 2^{14} - 1 .$$

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store the A_k in successive locations of memory.

T3) Compute and store in successive locations of memory:

$$S_1 = A_1 - A_2$$

$$S_2 = A_1 + A_2$$

$$S_3 = A_3 - A_4$$

$$S_4 = A_3 + A_4$$

$$S_5 = A_5 - A_6$$

$$S_6 = A_5 + A_6$$

$$S_7 = A_7 - A_8$$

$$S_8 = A_7 + A_8 .$$

T4) When the non-failed processor completes its last instruction compare S_1, S_2, ..., S_8, term by term, in both the non-failed and failed processors. If S_k is the first variable to miscompare set L = K. If all S_k compare set L = 0.

b. Instruction Set

During a typical computation the following instructions were executed.

INSTRUCTION	FREQUENCY
LOAD	2
STORE	2
ADD	2
SUBTRACT	1
BRANCH	2
TRANSFER	2

4.5.4 Search and Compute (SERCOM)

a. Procedure

T0) Select 8 sets of integers, (A_k, B_k, C_k), at random, each component from the interval

$$0 \leq x \leq 20 .$$

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store the (A_k, B_k, C_k) in successive locations of memory.

T3) Compute and store in successive locations of memory

$$\left.\begin{aligned} S_{1k} &= B_k + C_k \\ S_{2k} &= B_k \end{aligned}\right\} \quad \text{if } B_k \leq A_k$$

$$\left.\begin{aligned} S_{1k} &= B_k + C_k \\ S_{2k} &= B_k - C_k \end{aligned}\right\} \quad \text{if } A_k < B_k \text{ and } C_k \leq A_k$$

$$\left.\begin{aligned} S_{1k} &= B_k - C_{k\,(*1)} \\ S_{2k} &= B_k \times C_k \end{aligned}\right\} \quad \text{if } A_k < B_k \text{ and } A_k < C_k .$$

*1 Multiplication is performed by successive addition.

(T4) When the non-failed processor completes its last instruction compare S_{1k}, S_{2k} term by term, in both the non-failed and failed processors.(*2) If S_{1k} or S_{2k} is the first variable to miscompare set L = K. If all S_{1k}, S_{2k} compare set L = 0.

b. Instruction Set

During a typical computation the following instructions were executed:

INSTRUCTION	FREQUENCY
LOAD	6
STORE	6
ADD	17
SUBTRACT	1
BRANCH	24
TRANSFER	5

4.5.5 Linear Convergence (LINCON)

a. Procedure

T0) Select the following integers from the indicated intervals:

$$M_o, \quad -8 \leq M_o \leq 8$$

$$Y_o, \quad -2^{14} + 1 \leq Y_o \leq 2^{14} - 1$$

$$X_1, X_2, \ldots, X_8, \quad 0 \leq X_k \leq 2^{11}.$$

Assume that $X_1 < X_2 < \ldots < X_8$.

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store M_o, Y_o, X_1, X_2, ..., X_8, in successive locations of memory.

*2 Although this program was written to utilize 8 sets of integers, this experiment was performed using only 1 set.

T3) Compute M_k, Y_k, for K = 1, 2, ..., 8 as specified in the following flow flow diagram of figure 1, and store in successive locations of memory. Note again, that all multiplications are performed as successive additions.

T4) When the non-failed processor completes its last instruction compare M_1, Y_1, ..., M_8, Y_8, term by term, in both the non-failed and failed processors. If M_k or Y_k is the first variable to miscompare set L=K*. If all M_k, Y_k compare set L=0.

b. Instruction Set

During a typical computation the following instructions were executed:

INSTRUCTION	FREQUENCY
LOAD	38
STORE	38
ADD	16
SUBTRACT	4
BRANCH	39
TRANSFER	11
CLEAR	1

4.5.6 Quadratic (QUAD)

a. Procedure

T0) Select 8 sets of integers, (A_K, B_K, C_K, X_K), at random from the indicated intervals:

$$A_k, B_k, C_k, \quad 0 \leq X \leq 2^{15} - 1$$

$$X_k, \quad -10 \leq X_k \leq 10$$

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store the (A_k, B_k, C_k, X_k) in successive locations of memory.

* Although this program was written to utilize 8 values of X, this experiment was performed using only X_1.

T3) Compute and store in successive locations of memory (overflows are ignored):

$$S_k = (A_k X_k)X_k - B_k X_k - C_k^{(*1)}$$

$$K = 1, 2, \ldots, 8^{(*2)}$$

T4) When the non-failed processor completes its last instruction, compare S_1, S_2, ..., S_8, term by term, in both the non-failed and failed processors. If S_k is the first variable to miscompare set L = K. If all S_k compare set L = 0.

b. Instruction Set

During a typical computation the following instructions were executed:

INSTRUCTION	FREQUENCY
LOAD	7
STORE	5
ADD	30
SUBTRACT	1
BRANCH	38
TRANSFER	6

4.6 Phase II Experiments

This phase consists of injecting faults and executing a typical avionic flight control system self-test program to determine failure detection coverage. The self-test program was written expressly for this study.

A flight control system may employ one or more self-test programs and in a variety of ways. For example, as a background program, in pre-flight test, in maintenance test or on-line to isolate a failed computer. While the present study does not preclude any of these, the on-line application is the most interresting and critical. For orientation purposes an on-line application of self-test will be briefly described. The control system consists of three, identical

*1 Multiplication is performed by successive addition.
*2 Although this program was written to utilize 8 sets of integers, this experiment was performed using only 4 sets.

digital computers each driving a command port of a triplex, mechanically voted actuator. A first failure is detected via comparison-monitoring and the offending computer is desengaged. The second computer failure is also detected via comparison-monitoring and, upon detection, each of the two computers executes a self- test program designed to detect a specified proportion of failures in a specified period of time. The inability to successfully complete the self-test program or an explicit detection of the failure during this period of time causes the faulted computer to be disengaged from the actuator - leaving the remaining computer in control. The disengagement logic is independent of the CPU. Essentially, disengage is "armed" after the first failure. Either of the remaining computers can, thereafter, call for an interrupt to self-test if it detects a miscomparison. This interrupt occurs in both computers and sets the program counters to the first instruction of the self-test and activates a one-shot of duration slightly in excess of the time it takes for a non-failed computer to complete the self-test program. At any time until the one-shot times - out the self - test program may set a discrete output word whose value indicates whether or not the failure was detected. This discrete word is decoded in hardware.

If the value corresponds to a predetermined value (which does not exist in memory but must be computed) then the computer successfully completed self-test and, of course, did not find the fault. If the value is negative the computer is immediately disengaged. If, however, the word was not changed at all, having been initialized to zero at the start of the self-test, the computer is disengaged after the time-out if and only if the other computer successfully passed its self-test.

Before describing the fault injection procedure a brief overview of the self-test program will be given.

The self-test program provides the option of selecting any one of 14 test sequences, depending upon the coverage desired. The difference between these test sequences is in the number of input and internal state combinations employed in testing an instruction. A test procedure is specified by setting N, in location, ARG, to an integer value between 1 and 14, the complexity and length of the tests increasing with increasing N. In the present study N = 1.

The resultant test procedure consisted of 241 separate tests.

After a successful completion of a test the program increments register A_{14} (A_{14} is initialized to 0) and proceeds to the next test in the sequence. If, however, a failure is detected the program skips the remaining tests and transfers the contents of A_{14} to a designated memory location whose contents, ANSW, became the measure of failure-detection. If the program successfully completed all tests then ANSW = 241.

In the Phase II experiments a fault was defined as detected if, after a complete execution of the self-test program by the non-failed processor, ANSW ≠241 in the faulted processor. Observe that, according to this definition, a fault is detected if the faulted processor jumps out of the program, gets hung-up in an infinite loop or executes a single extra instruction before transferring the contents of A_{14}.

4.6.1 Self-Test

For each fault:

T1) Preset the program counter to the address of the first instruction (i.e., to CPUT) and initialize the stack pointer (i.e., register A_{15}) to the starting location of the scratchpad memory.

T2) Compute (i.e., execute self-test).

T3) When the processor completes the equivalent number of microcycles, corresponding to a complete execution of the self-test program by the non-failed processor, halt. The fault is detected if and only if ANSW ≠241.

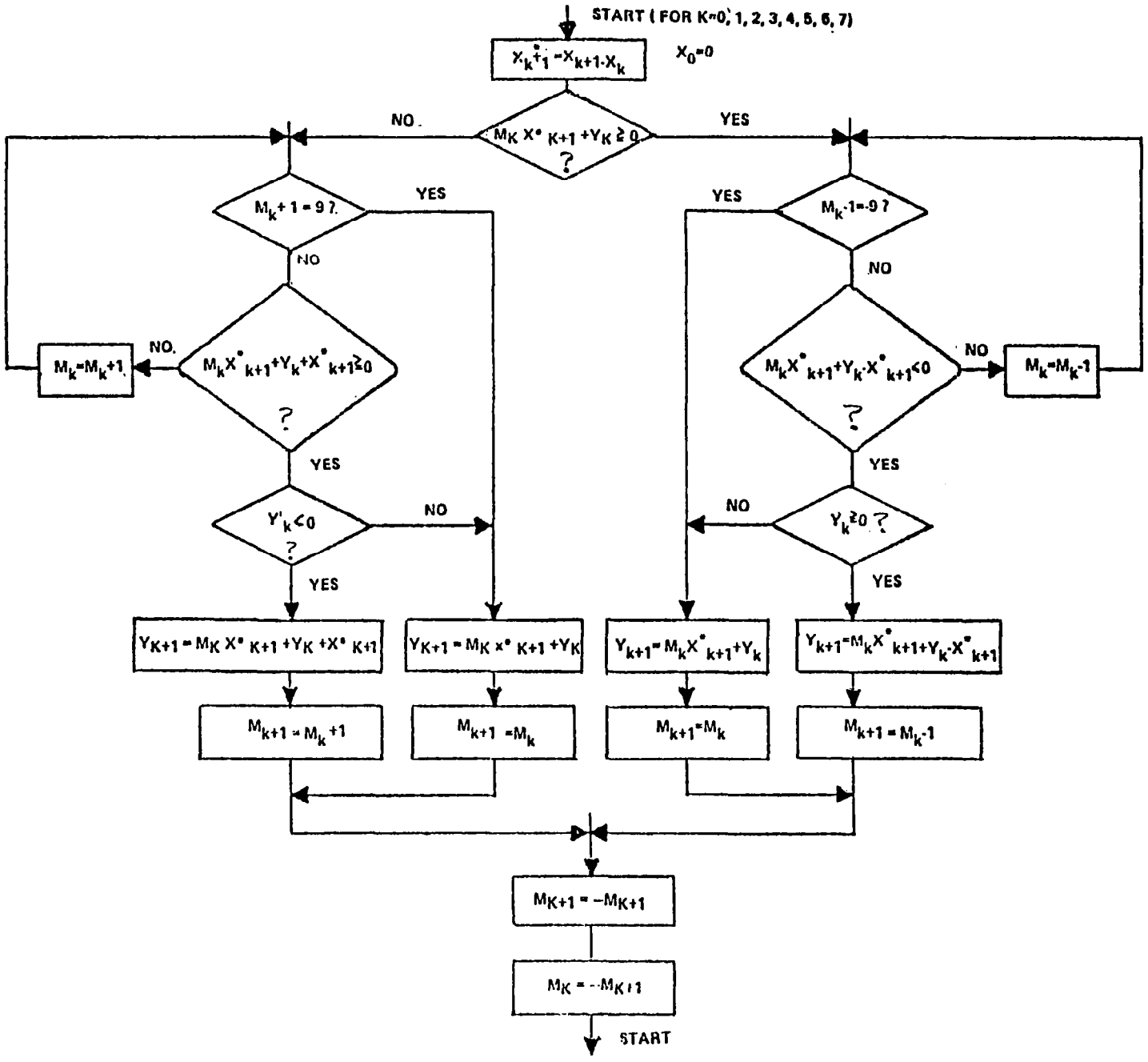


FIGURE 1 FLOW DIAGRAM FOR LINCON

5.0 RESULTS OF EXPERIMENTS

In this section the data from the experiments is presented concisely and with a minimum of commentary. A detailed analysis of the results is given in the next section.

5.1 Distribution of Faults

As indicated previously the selection of faults was random with each device weighted in proportion to its failure rate. S-a-0 and S-a-1 faults were equally weighted. The failure rates associated with each partition are given in Table 1. Initially, 1,000 gate-level and 400 component-level faults were randomly selected. Later, in order to reduce the cost of the runs it was necessary to reduce the number of faults actually injected. The number of faults finally selected for each experiment are given in Table 2. A detailed breakdown of the number of faults injected into each partition are given in Tables 3 & 4 for the Phase I and Phase II experiments, respectively. The numbers in parentheses are the number of faults that should have been selected if the sampling had been stratified over the partitions. Except for the Phase II component-level faults the indicated quantities include indistinguishable faults.

The same set of 400 component-level faults was used in all Phase I experients. From these 400, 200 were randomly selected and used in Phase II (i.e., SELF-TEST). The same 1,000 gate-level faults were used in FETSTO AND SERCOM. From these 1,000, 600 were randomly selected and used in ADDSUB, FIB, QUAD and LINCON. From these 600, 300 were randomly selected and used in Phase II.

5.2 Phase I Experiments

5.2.1 FETSTO Experiment

After each injected fault FETSTO was executed for 8 repetitions. The resultant histograms of detected faults versus repetitions to detection are shown in Figures 2a through 2i. Tabular results of the raw data are given in Table 5.

Figures 2a, 2b, Summarized

(Combined, S-a-1 and S-a-0 gate-level faults)

- o 61.7% undetected after 8 repetitions.
- o 29.9% detected in 1st repetition.
- o 8.4% detected in repetitions 2 through 8.
- o 59.8% of S-a-1 faults undetected.
- o 63.3% of S-a-0 faults undetected.

Figures 2c, 2d, 2e, Summarized

(Combined gate-level faults by partition)

- o 98% of faults in Partition #5 undetected.
- o 96.3% of faults in Partition #6 undetected.

We note that Partition #5 contains the micromemory.

Figures 2f, 2g, Summarized

(Combined, S-a-1 and S-a-0 component-level faults)

- o 35% undetected after 8 repetitions (compared with 61.7% for gate-level faults).
- o 51.3% detected in 1st repetition.
- o 13.2% detected in repetitions 2 through 8.
- o 31% of S-a-1 faults undetected.
- o 40.1% of S-a-0 faults undetected.

Figures 2h, 2i, Summarized

(Combined component-level faults by partition)

- o Partitions #5 and #6 did not allow for component-level faults. Pin faults (i.e,., component-level) were injected at adjacent partitions.

5.2.2 ADDSUB Experiment

After each injected fault ADDSUB was executed for 8 repetitions. The resultant histograms of detected faults versus repetitions to detection are shown Figures 3a through 3i. Tabular results of the raw data are given in Table 6.

Figures 3a, 3b, Summarized

(Combined, S-a-1 and S-a-0 gate-level faults)

- o 59.6% undetected after 8 repetitions.
- o 33.5% detected in 1st repetition.
- o 7.0% detected in repetitions 2 through 8.
- o 54.% of S-a-1 faults undetected.
- o 64.2% of S-a-0 faults undetected.

Figures 3c, 3d, 3e, Summarized

(Combined gate-level faults by partition)

- o 99% of faults in Partition #5 undetected.
- o 100% of faults in Partition #6 undetected.

Figures 3f, 3g, Summarized

(Combined, S-a-1 and S-a-0 component-level faults)

- o 32.3% undetected after 8 repetitions (compared with 59.5% for gate-level faults).
- o 57% detected in 1st repetition.
- o 10.7% detected in repetitions 2 through 8
- o 27.6% of S-a-1 faults undetected.
- o 37.1% of S-a-0 faults undetected.

Figures 3h, 3i, Summarized

(Combined component-level faults by partition)

- o Partitions #5 and #6 did not allow for component-level faults. Pin faults were injected at adjacent partitions.

5.2.3 FIB Experiment

After each injected fault FIB was executed for 8 repetitions. The resultant histograms of detected faults versus repetitions to detection are shown in Figures 4a through 4i. Tabular results of the raw data are given in Table 7.

Figures 4a, 4b, Summarized

(Combined, S-a-1 and S-a-0 gate-level faults)

- o 58.2% undetected after 8 repetitions.
- o 35% detected in 1st repetition.
- o 6.8% detected in repetitions 2 through 8
- o 54.3% of S-a-1 faults undetected.
- o 62.2% of S-a-0 faults undetected.

Figures 4c, 4d, 4e, Summarized

(Combined gate-level faults by partition)

- o 98% of faults in Partition #5 undetected.
- o 100% of faults in Partition #6 undetected.

Figures 4f, 4g, Summarized

(Combined, S-a-1 and S-a-0 component-level faults)

- o 28% undetected after 8 repetitions (compared with 58.2% for gate-level faults).
- o 61.3% detected in 1st repetition.

- o 10.7% detected in repetitions 2 through 8.
- o 22.7% of S-a-1 faults undetected.
- o 33.5% of S-a-0 faults undetected.

Figures 4h, 4i, Summarized

(Combined component-level faults by partition)

- o Partitions #5 and #6 did not allow for component-level faults. Pin faults were injected at adjacent partitions.

5.2.4 QUAD Experiment

After each injected fault QUAD was executed for 4 repetitions. The resultant histograms of detected faults versus repetitions to detection are shown in Figures 5a through 5i. Tabular results of the raw data are given in Table 8.

Figures 5a, 5b, Summarized

(Combined, S-a-1 and S-a-0 gate-level faults)

- o 53.3% undetected after 4 repetitions.
- o 43.2% detected in 1st repetition.
- o 3.5% detected in repetitions 2, 3 and 4.
- o 49.3% of S-a-1 faults undetected.
- o 57.4% of S-a-0 faults undetected.

For comparison purposes it is desirable to extrapolate the results of QUAD to 8 repetitions. To obtain a rough extrapolation we note that the average proportion of detected faults in repetitions 2 through 8 in the FETSTO, ADDSUB and FIB experiment is 7.4%. Using this estimate we obtain:

- o 49.4% undetected after 8 repetitions.

Figures 5c, 5d, 5e, Summarized

(Combined gate-level faults by partition)

- o 97.1% of faults in Partition #5 undetected after 4 repetitions.
- o 100% of faults in Partition #6 undetected after 4 repetitions.

Figures 5f, 5g, Summarized

(Combined, S-a-1 and S-a-0 component-level faults)

- o 23.5% undetected after 4 repetitions (compared with 53.3% for gate-level faults).
- o 71.8% detected in 1st repetition

- o 4.7% detected in repetitions 2, 3 and 4.
- o 20.2% of S-a-1 faults undetected.
- o 26.9% of S-a-0 faults undetected.

Again, to extrapolate the results of QUAD to 8 repetitions we note that the average proportion of detected component-level faults in repetitions 2 through 8 in the FETSTO, ADDSUB and FIB experiment is 11.5%. Using this estimate we obtain:

- o 16.7% undetected after 8 repetitions.

Figures 5h, 5i, Summarized

(Combined component-level faults by partition) After 4 repetitions,

- o 28.6% undetected in Partition #1.
- o 10.2% undetected in Partition #2.
- o 43.6% undetected in Partition #3.
- o 18.5% undetected in Partition #4.

5.2.5 SERCOM Experiment

After each injected fault SERCOM was executed for a single repetition. The resultant histograms of detected faults versus repetitions to detection are shown in Figures 6a through 6i. Tabular results of the raw data are given in Table 9.

Figures 6a, 6b, Summarized

(Combined, S-a-1 and S-a-0 gate-level faults)

- o 60.5% undetected after a single repetition.
- o 39.5% detected in the 1st repetition.
- o 57.3% of S-a-1 faults undetected.
- o 63.6% of S-a-0 faults undetected.

As in QUAD, extrapolating to 8 repetitions, we obtain:

- o 53.1% undetected after 8 repetitions.

Figures 6c, 6d, 6e, Summarized

(Combined gate-level faults by partition)

- o 98% of faults in Partition #5 undetected.
- o 100% of faults in Partition #6 undetected.

Figures 6f, 6g, Summarized

(Combined, S-a-1 and S-a-0 component-level faults)

- o 35.2% undetected after a single repetition (compared with 60.5% for gate-level faults).
- o 64.8% detected in 1st repetition.
- o 27.6% of S-a-1 faults undetected.
- o 43.1% of S-a-0 faults undetected.

Extrapolating the results to 8 repetitions we obtain:

- o 23.8% undetected after 8 repetitions.

Figures 6h, 6i, Summarized

(Combined component-level faults by partition). After a single repetition,

- o 24.4% undetected in Partition #1.
- o 33.6% undetected in Partition #2
- o 46.8% undetected in Partition #3.
- o 35.9% undetected in Partition #4.

5.2.6 LINCON Experiment

After each injected fault LINCON was executed for a single repetition. The resultant histograms of detected faults versus repetitions to detection are shown in Figures 7a through 7i. Tabular results of the raw data are given in Table 10.

Figures 7a, 7b, Summarized

(Combined, S-a-1 and S-a-0 gate-level faults)

- o 48.3% undetected after a single repetition.
- o 51.7% detected in the 1st repetition.
- o 46.7% of S-a-1 faults undetected.
- o 50% of S-a-0 faults undetected.

As is SERCOM, extrapolating to 8 repetitions, we obtain,

- o 40.9% undetected after 8 repetitions.

Figures 7c, 7d, 7e, Summarized

(Combined, S-a-1 and S-a-0 component-level faults)

- o 96.2% of faults in Partition #5 undetected.
- o 100% of faults in Partition #6 undetected.

Figures 7f, 7g, Summarized

(Combined, S-a-1 and S-a-0 component-level faults)

- o 23.5% undetected after a single repetition (compared with 48.3% for gate-level faults).
- o 76.5% detected in 1st repetition.
- o 21.2% of S-a-1 faults undetected.
- o 25.9% of S-a-0 faults undetected.

Extrapolating the results to 8 repetitions we obtain,

- o 12% undetected after 8 repetitions.

Figures 7h, 7i, Summarized

(Combined component-level faults by partition)

- o 20.8% undetected in Partition #1.
- o 10.9% undetected in Partition #2.
- o 41.5% undetected in Partition #3.
- o 26.1% undetected in Partition #4.

The Phase I results are concisely summarized in Table 11.

5.3 Phase II Experiments

5.3.1 Indistinguishable Fault Estimates

As indicated in Section 5.1, 300 gate-level faults and 200 component-level faults were injected in the Phase II experiments. In order to obtain an estimate of the proportion of indistinguishable faults each resultant, undetected fault was analyzed and those faults which were obviously indistinguishable were disqualified. At the gate-level, 71 out of 300 faults were identified as indistinguishable. Thus, the estimated proportion of components yielding indistinguishable are:

$$\gamma^* = \frac{71}{300} = 0.2366 \text{ at the gate-level}$$

and $$\gamma^* = \frac{11}{200} = .055 \text{ at the component-level}$$

Since indistinguishable faults were not disqualified in the Phase I experiments all coverage estimates of Phase I should be divided by the appropriate $1 - \gamma^*$ factor, as prescribed in Section 10.

5.3.2 Self-Test Coverage

Having disqualified 71 indistinguishable faults 229 faults were effectively injected at the gate-level and 189 at the component-level. The resultant raw data is given in Table 12 by partitions.

As indicated previously, after each injected fault the self-test program was executed. Faults were generally detected either because an explicit test detected the fault or the fault caused a jump out of the program. These latter faults are denoted in Table 12 by "wild branches".

5.3.3 Gate-Level Faults

From Table 19 we observe,

- o 198 out of 229 combined faults were detected for a coverage of 86.46%.
- o 100 out of 114 S-a-1 faults were detected for a coverage of 87.72%.
- o 98 out of 115 S-a-0 faults were detected for a coverage of 85.22%.
- o 9 out of 17 faults in Partition #5 were detected for a coverage of 52.94%.
- o 5 out of 8 faults in Partition #6 were detected for a coverage of 62.5%.
- o If faults in Partitions #5 and #6 are disqualified then 184 out of 204 faults were detected for a coverage of 90.2%.

As an indication of fault latency the test that actually caused detection of the fault was recorded. It is recalled that arithmetic register A_{14} is incremented after the successful completion of each of the 241 tests that comprise Self-Test. If a test is unsuccessful or if all tests are successful the contents of A_{14} are transferred to a designated memory location, ANSW. The fault is considered detected if, after a complete execution of Self-Test by the non-failed processor, the contents of ANSW $\neq 241$.

Occasionally a fault can result in an incorrect incrementation of A_{14} or prevent the transfer to ANSW. In the former case the test cannot be identified correctly. In the latter case the contents of ANSW remains at its initial value of zero and, as a consequence, does not indicate the correct test number either.

The procedure used to identify the test was to set:

(Test #)- 1 = (ANSW) when (ANSW) $\neq$ 0.

(Test #)- 1 = (A_{14}) when (ANSW) = 0.

This results in the correct identification of the test, in most cases.

Table 13 gives the number of gate-level and component-level faults detected. When (Test #)- 1 = 0 the effect is referred to, in Table 12, as a "wild branch" since, in most cases, the fault caused a jump out of the program.

From Table 13 we observe:

- o 103 out of the 198 faults detected resulted in wild branches, i.e., 52%.
- o 95 faults were detected by an explicit test (even though it was not always possible to identify the test).

- Out of the 241 possible tests, at most 46 actually resulted in a detection, i.e., most of the tests were, effectively, redundant.

5.3.4 Component-Level Faults

From Tables 12 and 13 we observe:

- 185 out of 189 combined faults were detected for a coverage of 97.9%.
- 97 out of 100 S-a-1 faults were detected for a coverage of 97%.
- 88 out of 89 S-a-0 faults were detected for a coverage of 98.9%.
- 106 out of 189 faults detected resulted in wild branches, i.e., 56%.
- 79 faults were detected by an explicit test (even though it was not always possible to identify the test).
- Out of 241 possible tests, at most 44 actually resulted in a detection.

Table 14 shows the self-test coverage at the gate and component-levels by partitions.

5.4 URN Model Parameters

From the Phase I experiments the parameters of the Urn Model were estimated for the three programs

FETSTO

ADDSUB

FIB

for combined, S-a-1 and S-a-0 gate-level and component-level faults.

Table 15

This table gives the exact and approximate maximum likelihood estimates of a, P and P_0, as defined in Section 10. Also shown are the resultant, computed, Urn Model distribution of terms of the occupancy probabilities of cells 1, 2, ..., 8. These correspond to the probabilities X_i, Y_i or Z_i for S-a-0, S-a-1 and combined faults, respectively. In keeping with our subsequent notation, the occupancy probability of cell 9 is actually the probability that the fault is undetected in the previous 8 repetitions. As a comparison, the corresponding empirical distributions are also given. These were obtained directly from the latency distributions of Section 5.2.

Referring to the table, we note that the approximate estimates are accurate to two decimal places, in most cases.

Figures 8, 9, 10

The resultant, computed Urn Model distributions are shown graphically in these figures using, in all cases, the exact estimators. The distributions are superimposed on the corresponding empirical distributions of Section 5.2.

Table 16

This table gives the elements of the error covariance matrix for the Urn Model estimates. The matrix was obtained using the exact maximum likelihood estimates.

Table 17

This table gives the elements of the inverse error covariance matrix of Table 16.

Table 18

For completeness, the intermediate parameters that were used to obtain the estimates of a, P and P_o are given in this table. The intermediate parameters are m_1, m_2, ..., m_9, A, B, C, D as defined in Section 10. We note that the symbols m_1 referred to S-a-0 faults previously but, in the context of Table 18, they refer to S-a-0, S-a-1 or combined faults, as the case may be.

5.5 Accuracy and Confidence of Results

5.5.1 Phase I Results

The accuracy of the Phase I results will be illustrated by the combined, gate-level FETSTO experiments. Using the marginal distributions for latency cells #1 and #9 and using (24) of Section10.6 gives, for the errors at the 95% confidence level,

$$\varepsilon_1 = 1.96 \sqrt{\frac{.299\ (.701)}{1000}} = .028\ (9.36\%)$$

$$\varepsilon_9 = 1.96 \sqrt{\frac{.617\ (.383)}{1000}} = .030\ (4.86\%)$$

where $x_1 \approx .299$

$x_9 \approx .617$

$m = 1000.$

When the multivariate distribution is used, i.e., (23) of Section 10.6, the corresponding errors are

$$\varepsilon_1 = 2.45 \sqrt{\frac{.299}{1000}} = .042 \ (14\%)$$

$$\varepsilon_9 = 2.45 \sqrt{\frac{.617}{1000}} = .061 \ (9.9\%)$$

5.5.2 Phase II Results

The accuracy of the Phase II results can be estimated using (13) of Section10.5. We illustrate using the combined gate and component-level faults.

At the gate-level the estimated coverage is

$$z \approx .8646$$

with a sample size of 229, the indistinguishable faults having been disqualified. At the 95% confidence level the resultant error is

$$\varepsilon = 1.96 \sqrt{\frac{.8646\ (.1354)}{229}} = .044\ (5.1\%),\ \text{approximately.}$$

At the component-level the estimated coverage is

$$z \approx .979$$

with a sample size of 189. At the 95% confidence level the resultant error is

$$\varepsilon = 1.96 \sqrt{\frac{.979\ (.021)}{189}} = .020\ (2.0\%),\ \text{approximately.}$$

5.5.3 Urn Model Results

The accuracy of the Urn Model results will be illustrated by the combined, gate-level FETSTO experiment. In this experiment the parameters were estimated to be

$$p \approx .781$$

$$p_0 \approx .383$$

$$a \approx .464$$

using the approximate estimators (28) of Section 10.7.

From (29) the errors at the 95% level are, approximately,

$$\varepsilon_p = 1.96 \sqrt{\frac{.781\ (.219)}{1000 \times .383}} = .041\ (5.2\%)$$

$$\varepsilon_{p_0} = 1.96 \sqrt{\frac{.383\ (.617)}{1000}} = .030\ (7.8\%)$$

$$\varepsilon_a = 1.96 \sqrt{\frac{(.464)^2\ (.536)}{1000 \times .383\ (.219)}} = .073\ (15.7\%).$$

TABLE 1

FAILURE RATES OF PARTITIONS OF THE CPU

PARTITION	FAILURE RATE (FAILURES/HR x 10^{-6})
#1	7.282
#2	11.916
#3	7.922
#4	9.652
#5	7.063
#6	1.188
	45.023

TABLE 2

NUMBER OF FAULTS INJECTED

EXPERIMENT	GATE-LEVEL	COMPONENT-LEVEL
FETSTO	1000	400
ADDSUB	600	400
FIB	600	400
QUAD	600	400
SERCOM	1000	400
LINCON	600	400
SELF-TEST	300	200

TABLE 3a

PHASE I EXPERIMENTS

NUMBER OF GATE-LEVEL FAULTS INJECTED BY PARTITIONS

PROGRAMS: FETSTO, SERCOM

PARTITION	S-a-0	S-a-1	COMBINED
1	82 (81)	76 (80)	158 (162)
2	147 (133)	127 (132)	274 (265)
3	87 (89)	98 (87)	185 (176)
4	98 (108)	108 (107)	206 (214)
5	75 (74)	75 (78)	150 (157)
6	14 (13)	13 (13)	27 (26)
TOTAL	503 (503)	497 (497)	1000 (1000)

PROGRAMS: FIB, ADDSUB, QUAD, LINCON

PARTITION	S-a-0	S-a-1	COMBINED
1	49 (48)	45 (49)	94 (97)
2	90 (78)	78 (80)	168 (159)
3	46 (52)	66 (53)	112 (106)
4	53 (63)	57 (65)	110 (129)
5	52 (46)	52 (48)	104 (94)
6	6 (8)	6 (8)	12 (16)
TOTAL	296 (296)	304 (303)	600 (601)

( ) = Theoretical

TABLE 3b

PHASE I EXPERIMENTS

NUMBER OF COMPONENT-LEVEL FAULTS INJECTED BY PARTITIONS

PROGRAMS: FETSTO, FIB, ADDSUB, SERCOM, QUAD, LINCON

PARTITION	S-a-0	S-a-1	COMBINED
1	38 (39)	39 (40)	77 (79)
2	58 (64)	79 (66)	137 (130)
3	52 (42)	42 (44)	94 (86)
4	49 (52)	43 (53)	92 (105)
TOTAL	197 (197)	203 (203)	400 (400)

( ) = Theoretical

TABLE 4a

PHASE II EXPERIMENTS

NUMBER OF GATE-LEVEL FAULTS INJECTED BY PARTITIONS

PARTITION	S-a-0	S-a-1	COMBINED
1	17 (24)	17 (24)	34 (48)
2	35 (40)	39 (40)	74 (80)
3	28 (27)	27 (26)	55 (53)
4	40 (32)	34 (32)	74 (64)
5	25 (24)	25 (23)	50 (47)
6	7 (4)	6 (4)	13 (8)
TOTAL	152 (151)	148 (149)	300 (300)

TABLE 4b

PHASE II EXPERIMENTS

NUMBER OF COMPONENT-LEVEL FAULTS INJECTED BY PARTITIONS

PARTITION	S-a-0	S-a-1	COMBINED
1	15 (18)	20 (20)	35 (37)
2	38 (29)	35 (32)	73 (61)
3	21 (19)	22 (22)	43 (41)
4	15 (23)	23 (26)	38 (50)
TOTAL	89 (89)	100 (100)	189 (189)

( ) = Theoretical

FETSTO LATENCY DATA

GATE-LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	m_1	n_1	m_2	n_2	m_3	n_3	m_4	n_4	m_5	n_5	m_6	n_6	m_7	n_7	m_8	n_8	m	n
P1	50	37	4	4	2	1	0	0	0	0	0	0	0	0	0	0	82	76
P2	46	72	9	10	11	0	0	0	0	0	4	2	0	1	0	3	147	127
P3	24	34	8	3	0	0	0	0	0	0	0	0	0	0	2	2	87	98
P4	16	19	1	6	6	1	0	0	0	0	0	0	0	1	0	0	98	108
P5	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	75	75
P6	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	14	13
TOTAL	136	163	22	26	19	2	0	0	0	0	4	2	0	2	2	5	503	497

m_1 = detected S-a-0 faults, ith cell

n_1 = detected S-a-1 faults, ith cell

COMPONENT-LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	m_1	n_1	m_2	n_2	m_3	n_3	m_4	n_4	m_5	n_5	m_6	n_6	m_7	n_7	m_8	n_8	m	n
P1	22	27	2	0	3	1	0	0	0	0	0	0	0	0	0	0	38	39
P2	36	49	4	13	2	2	1	0	0	0	0	0	5	0	0	2	58	79
P3	18	19	4	4	0	0	0	0	0	0	0	0	0	0	0	0	52	42
P4	16	18	2	5	2	0	0	0	0	0	0	0	0	0	1	0	49	43
P5																		
P6																		
TOTAL	92	113	12	22	7	3	1	0	0	0	0	0	5	0	1	2	197	203

TABLE 5

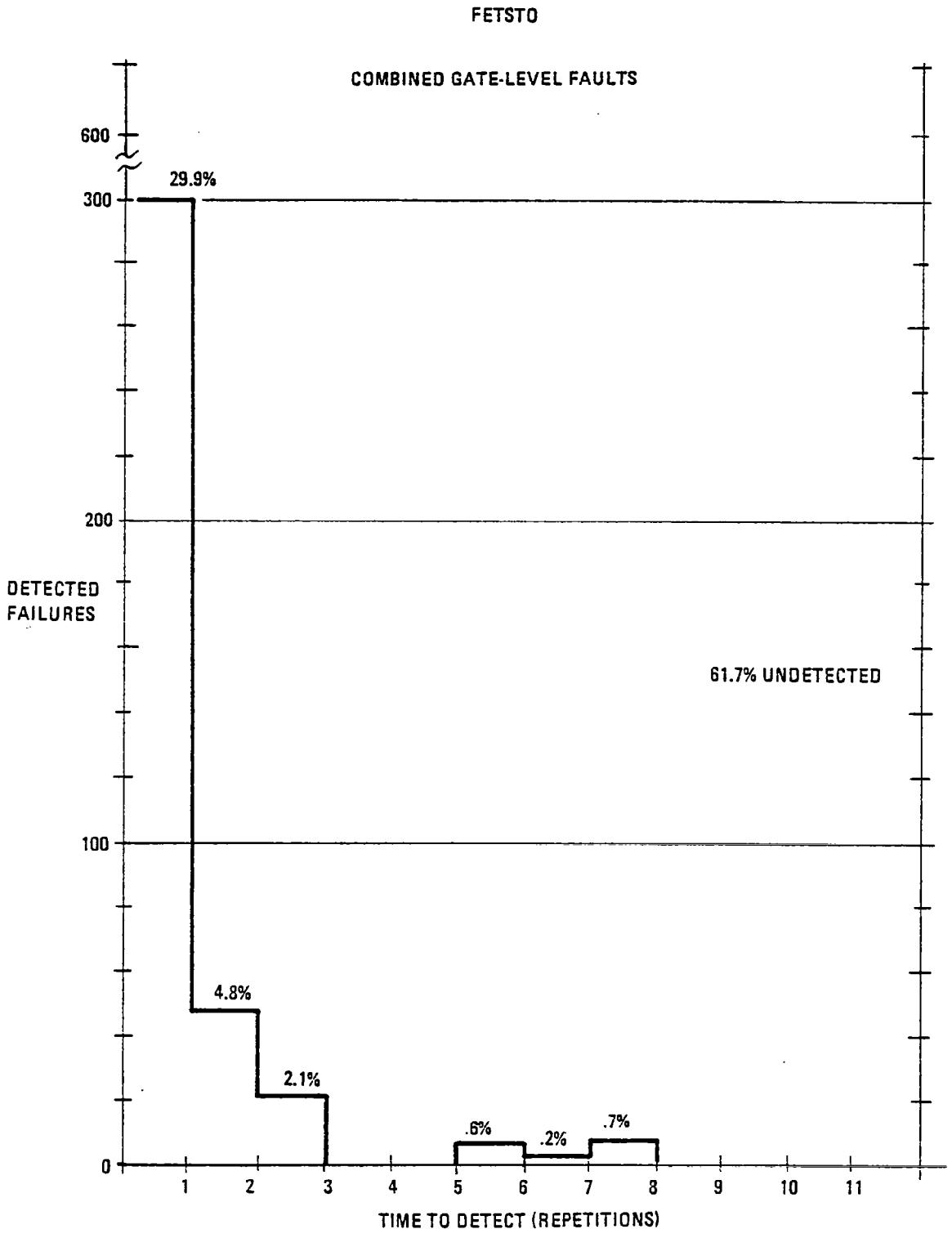
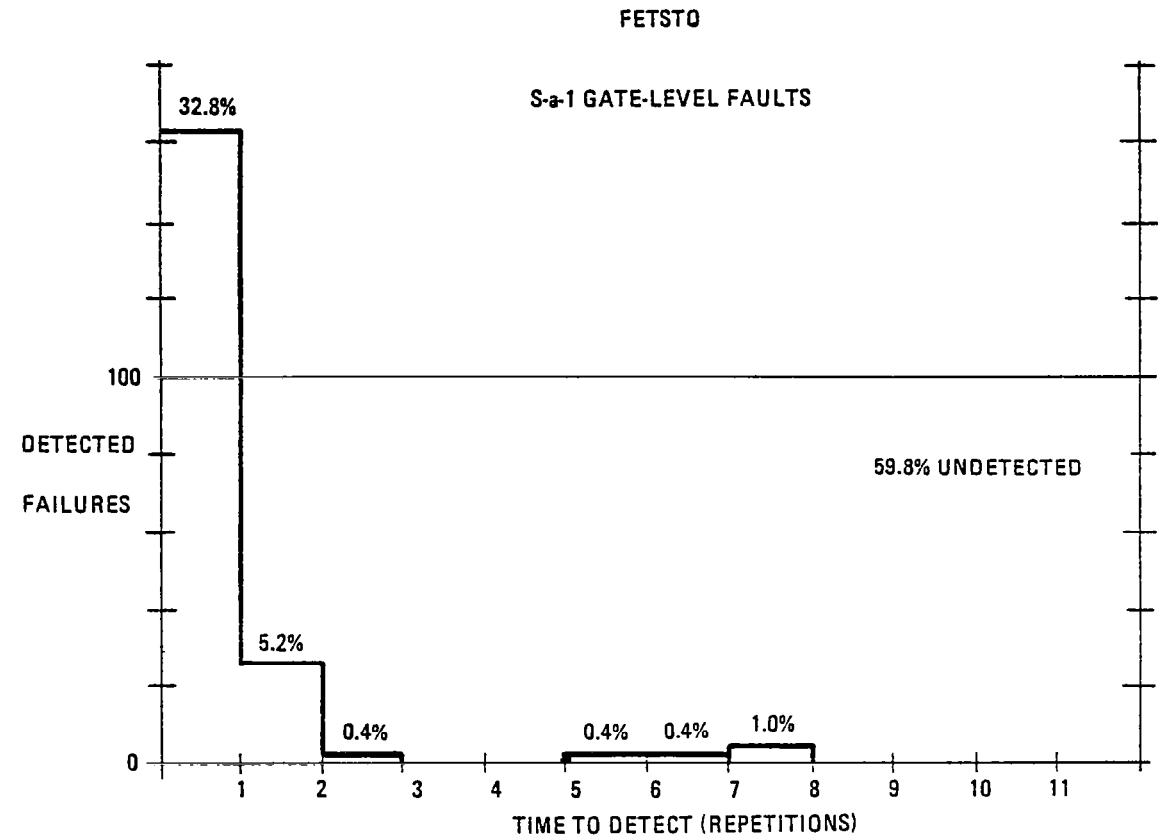

FETSTO
COMBINED GATE-LEVEL FAULTS
600
300
200
100
0
29.9%
4.8%
2.1%
.6%
.2%
.7%
61.7% UNDETECTED
DETECTED FAILURES
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 2a


FETSTO
S-a-1 GATE-LEVEL FAULTS
32.8%
100
DETECTED
FAILURES
59.8% UNDETECTED
5.2%
0.4%
0.4%
0.4%
1.0%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

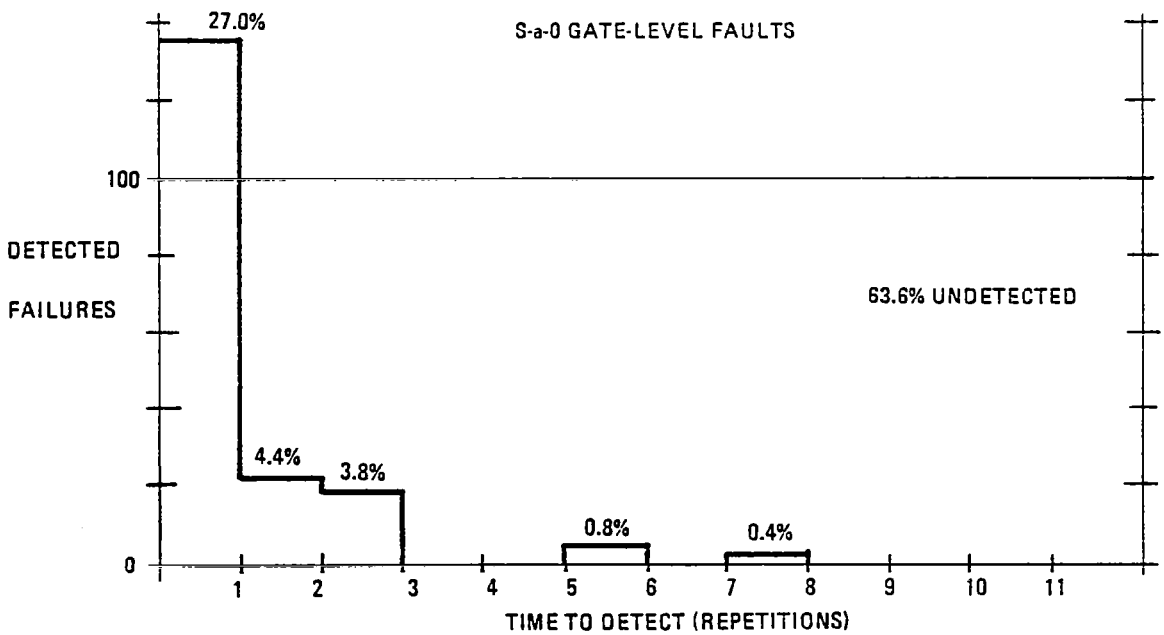
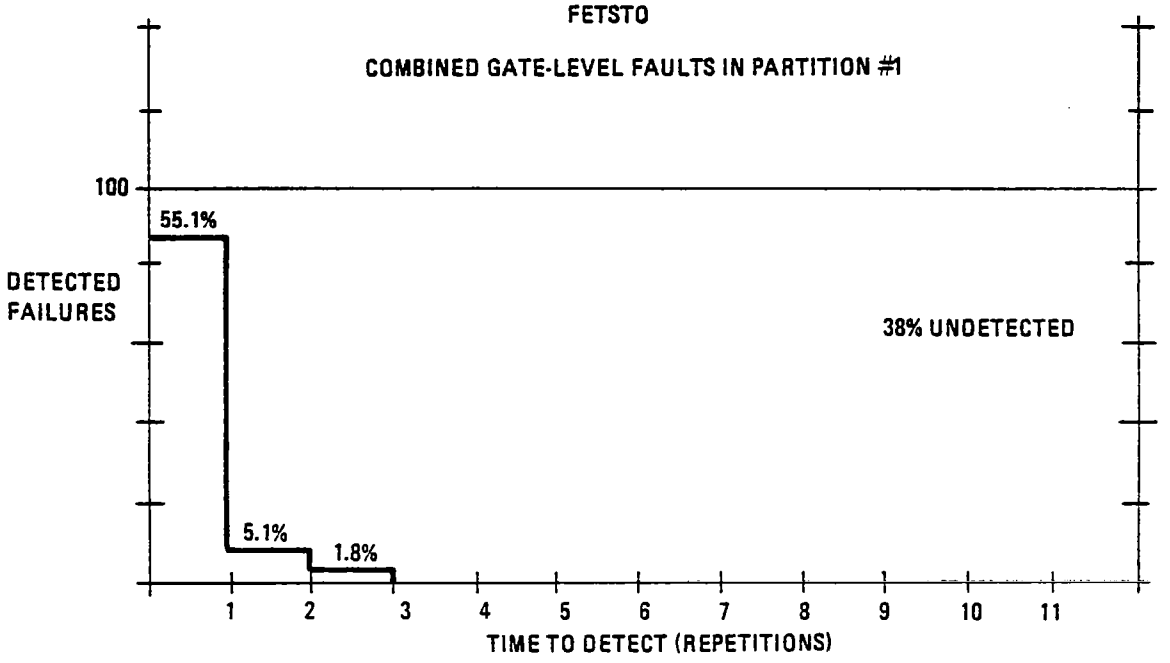

27.0%
S-a-0 GATE-LEVEL FAULTS
100
DETECTED
FAILURES
63.6% UNDETECTED
4.4%
3.8%
0.8%
0.4%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 2b


FETSTO
COMBINED GATE-LEVEL FAULTS IN PARTITION #1
100
55.1%
DETECTED
FAILURES
38% UNDETECTED
5.1%
1.8%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

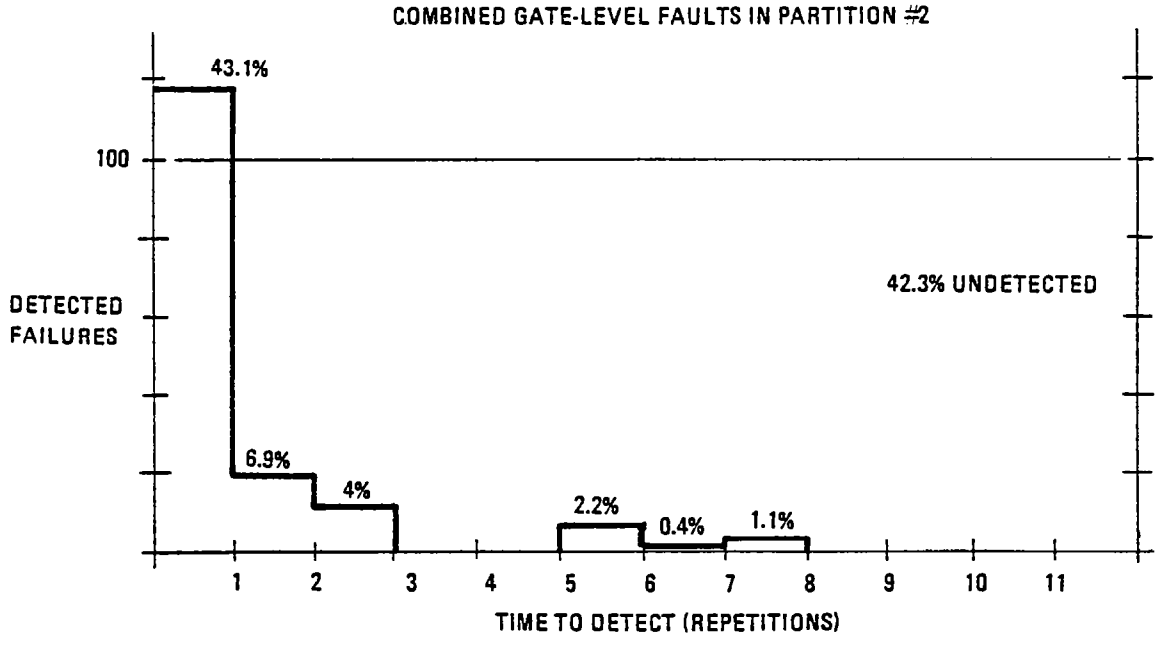

COMBINED GATE-LEVEL FAULTS IN PARTITION #2
43.1%
100
DETECTED
FAILURES
42.3% UNDETECTED
6.9%
4%
2.2%
0.4%
1.1%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 2c

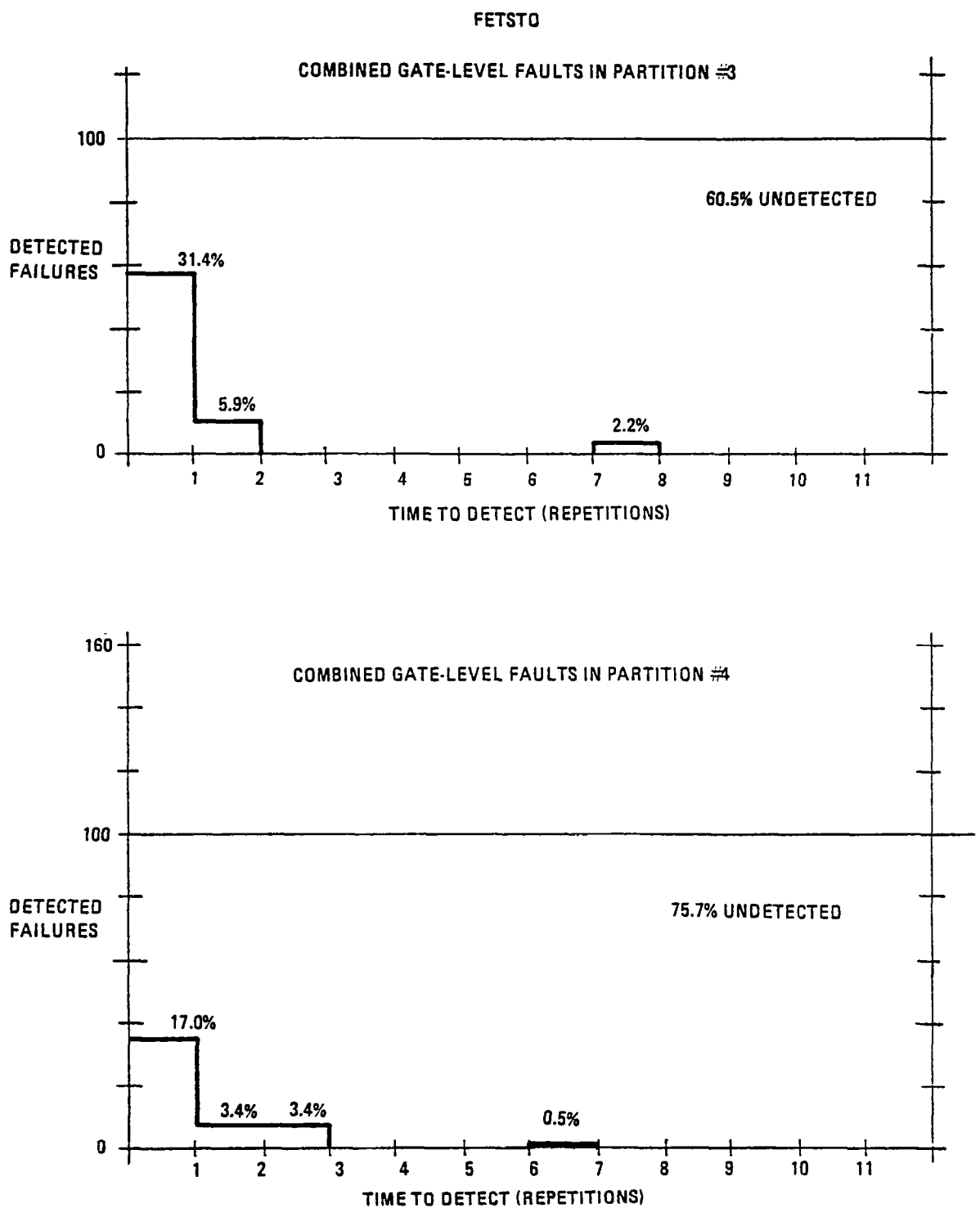
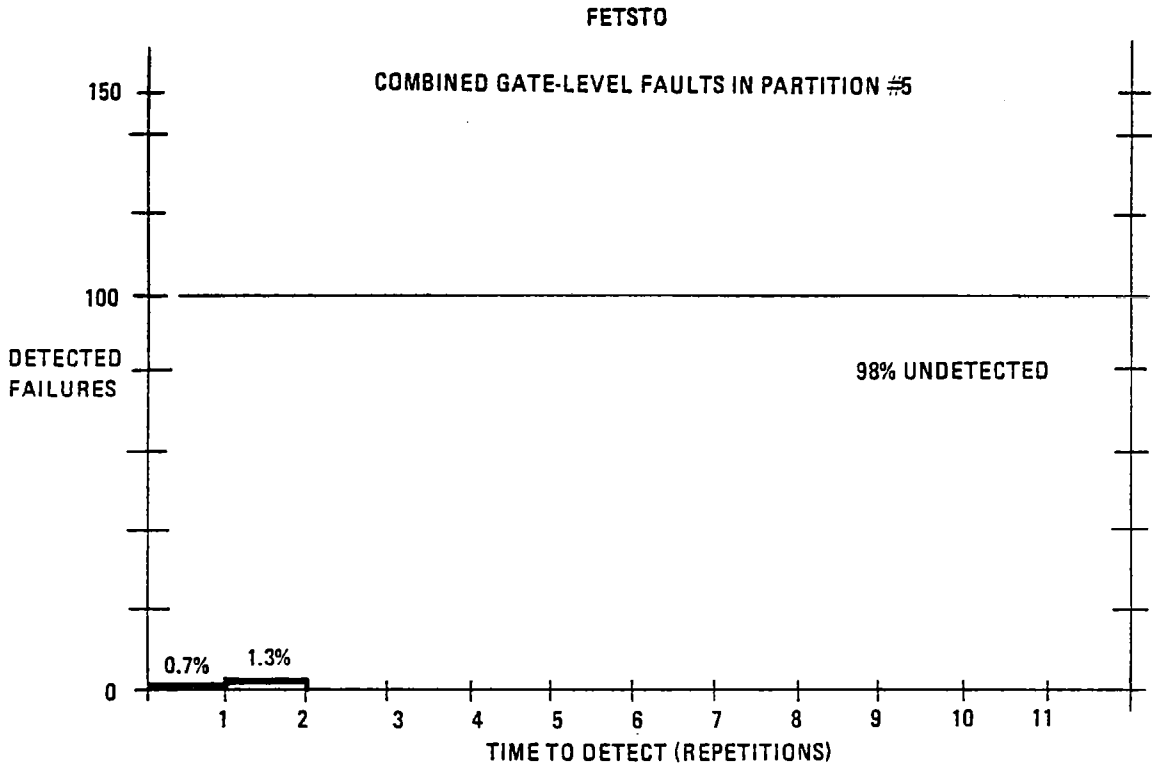

FETSTO
COMBINED GATE-LEVEL FAULTS IN PARTITION #3
100
60.5% UNDETECTED
DETECTED FAILURES
31.4%
5.9%
2.2%
0
1 2 3 4 5 6 7 8 9 10 11
TIME TO DETECT (REPETITIONS)
160
COMBINED GATE-LEVEL FAULTS IN PARTITION #4
100
DETECTED FAILURES
75.7% UNDETECTED
17.0%
3.4%
3.4%
0.5%
0
1 2 3 4 5 6 7 8 9 10 11
TIME TO DETECT (REPETITIONS)

FIGURE 2d


FETSTO
COMBINED GATE-LEVEL FAULTS IN PARTITION #5
150
100
DETECTED FAILURES
98% UNDETECTED
0.7%
1.3%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

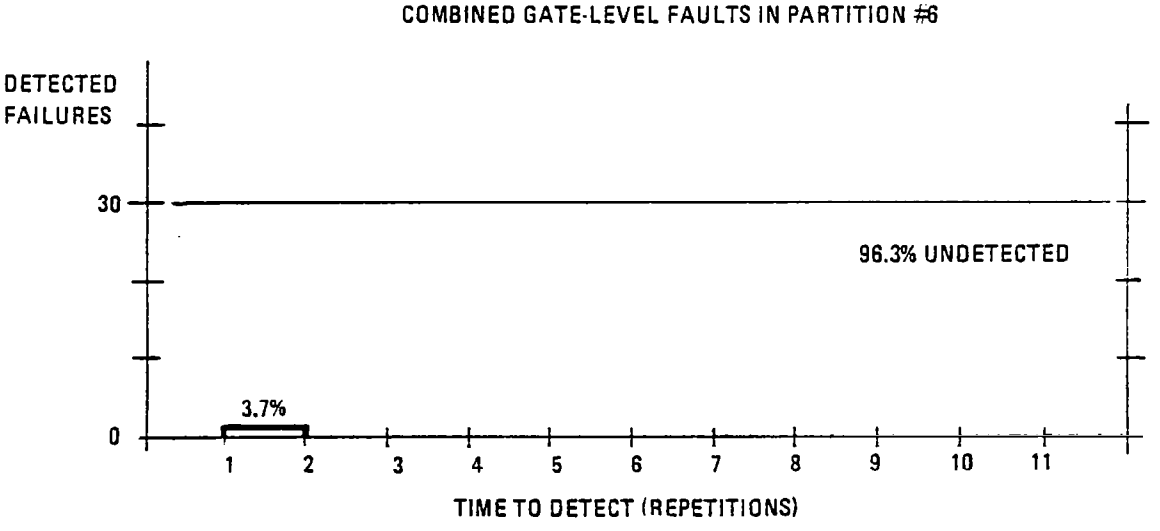

COMBINED GATE-LEVEL FAULTS IN PARTITION #6
DETECTED FAILURES
30
96.3% UNDETECTED
3.7%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 2e

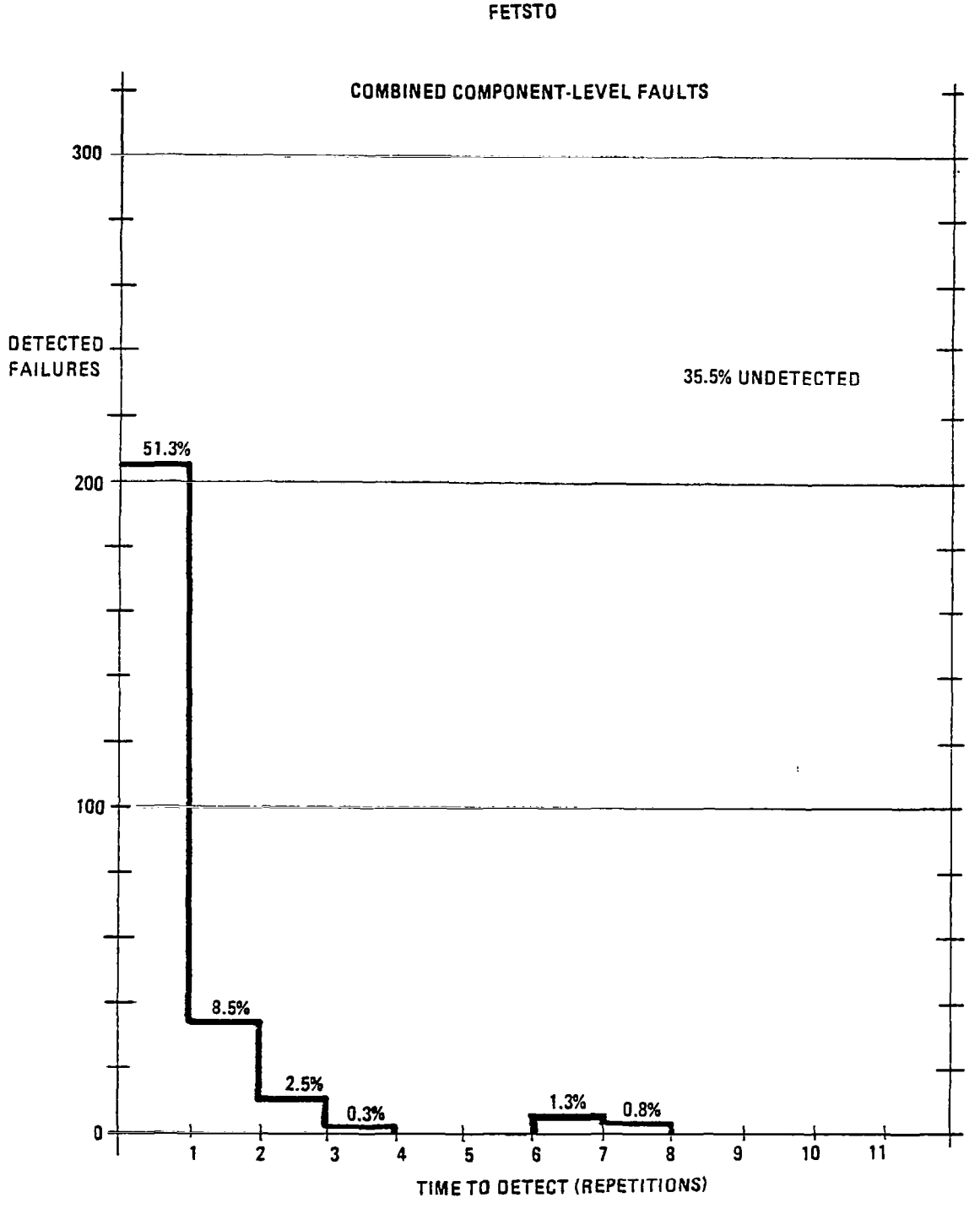

FETSTO
COMBINED COMPONENT-LEVEL FAULTS
300
DETECTED FAILURES
35.5% UNDETECTED
51.3%
200
100
8.5%
2.5%
0.3%
1.3%
0.8%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 2f

FETSTO

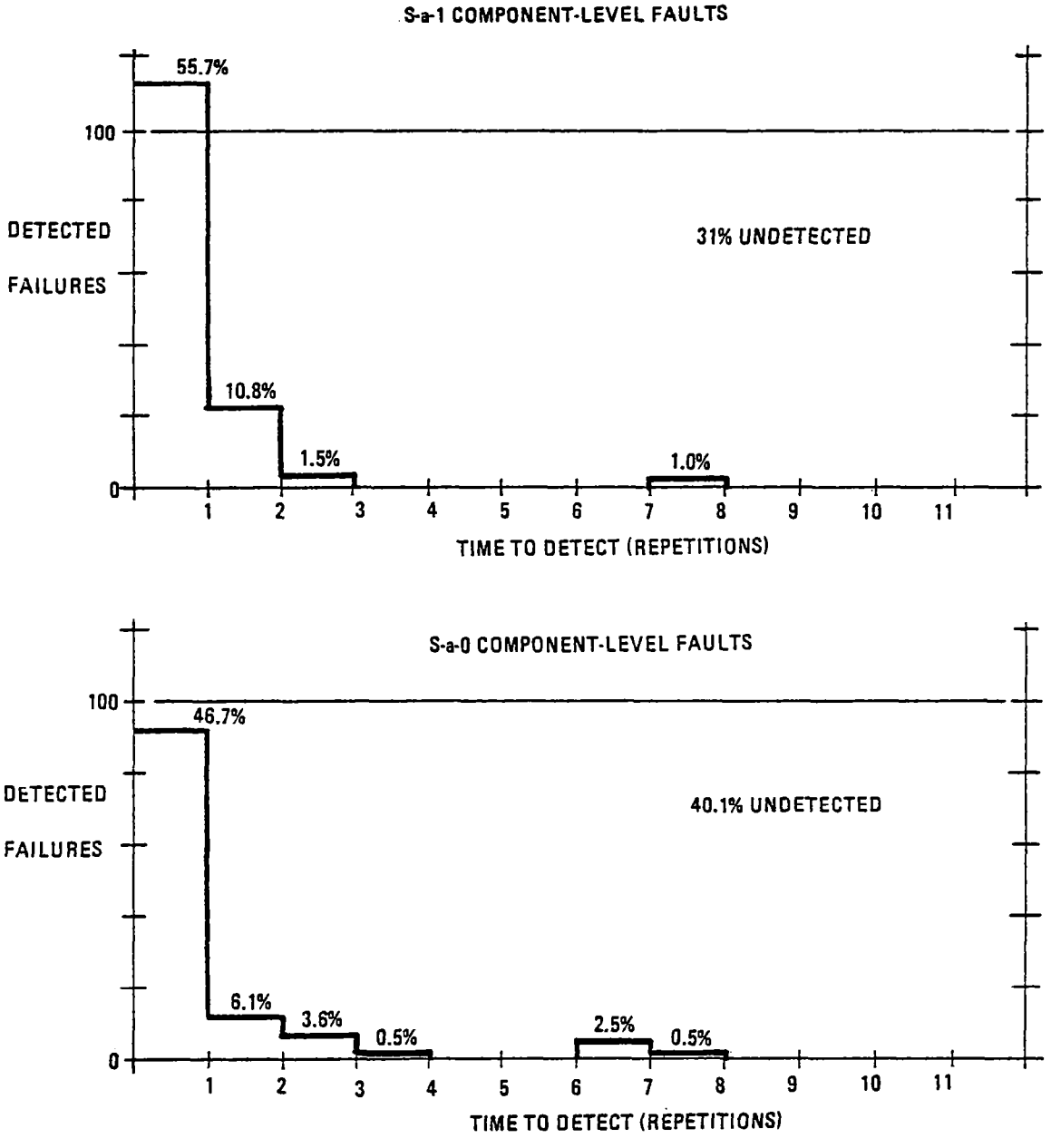
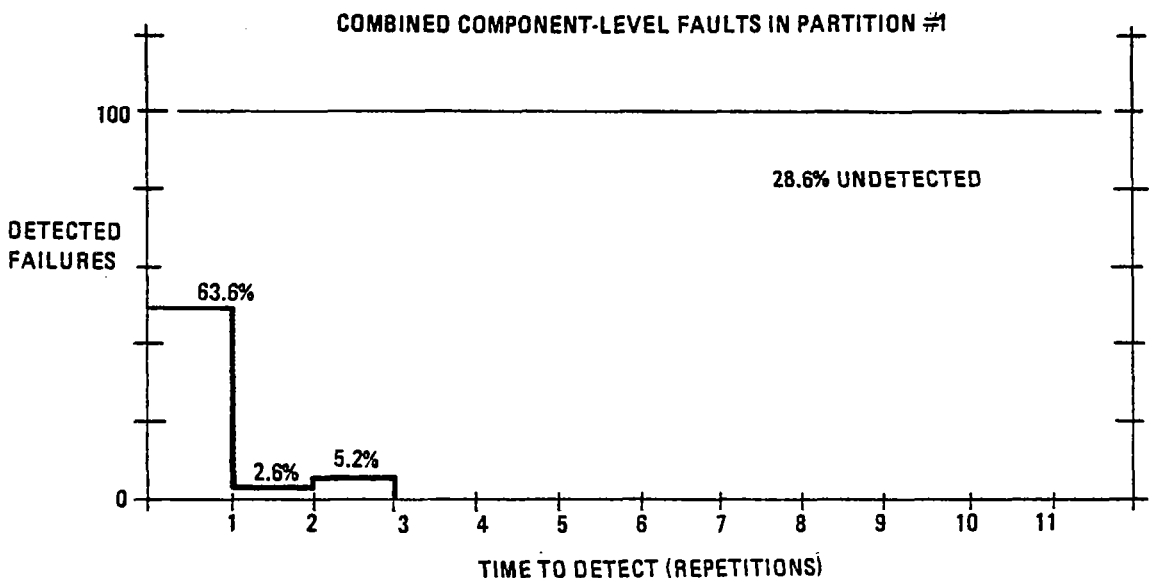


FIGURE 2g

FETSTO



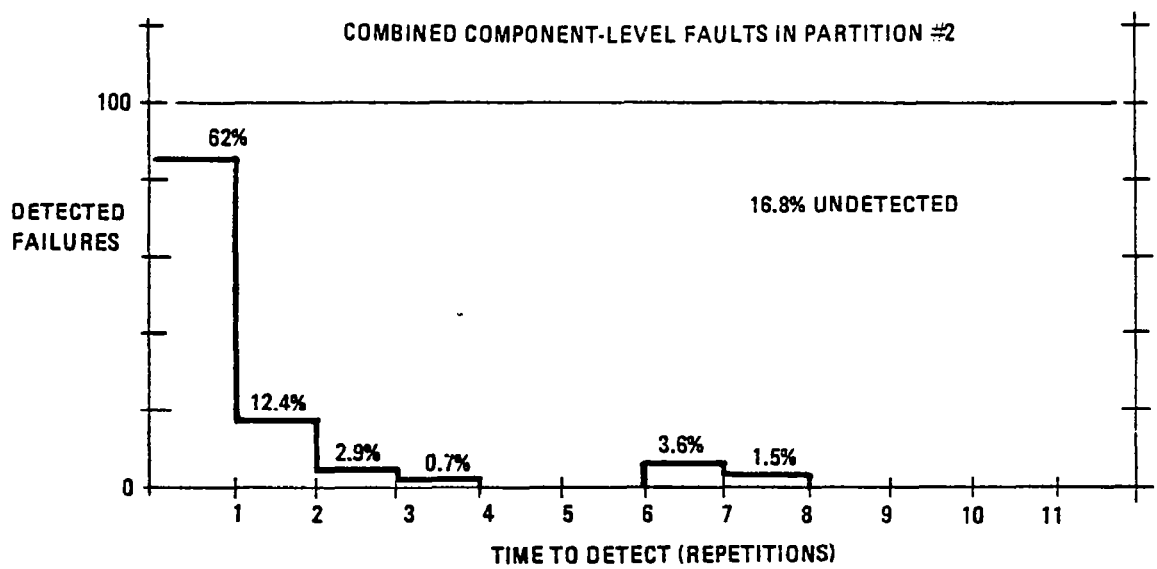
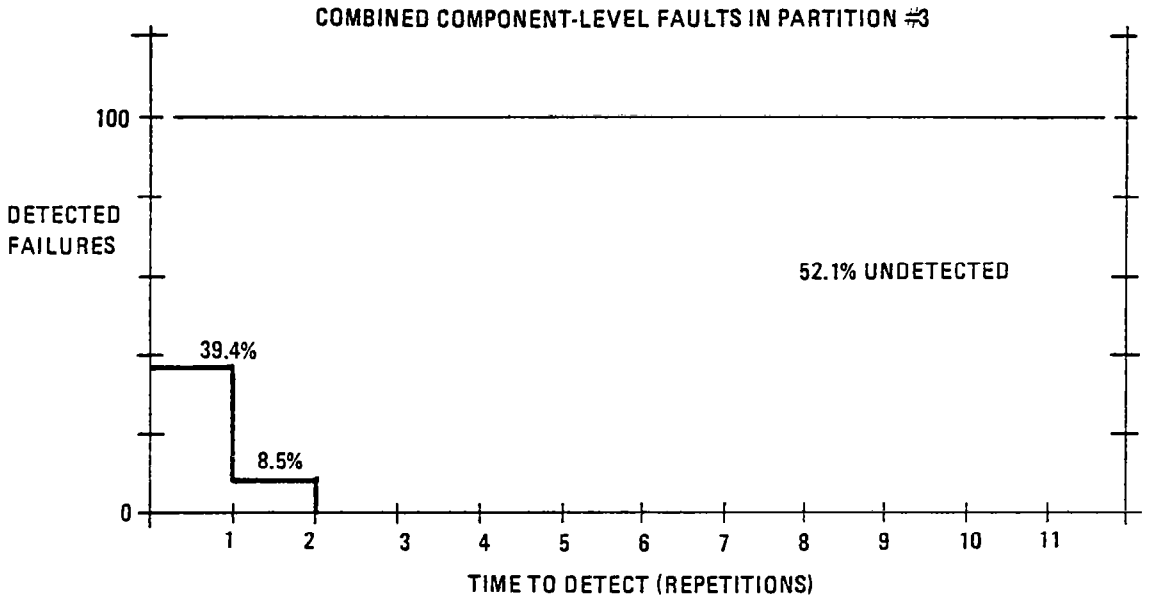


FIGURE 2h


COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #3
100
DETECTED FAILURES
52.1% UNDETECTED
39.4%
8.5%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

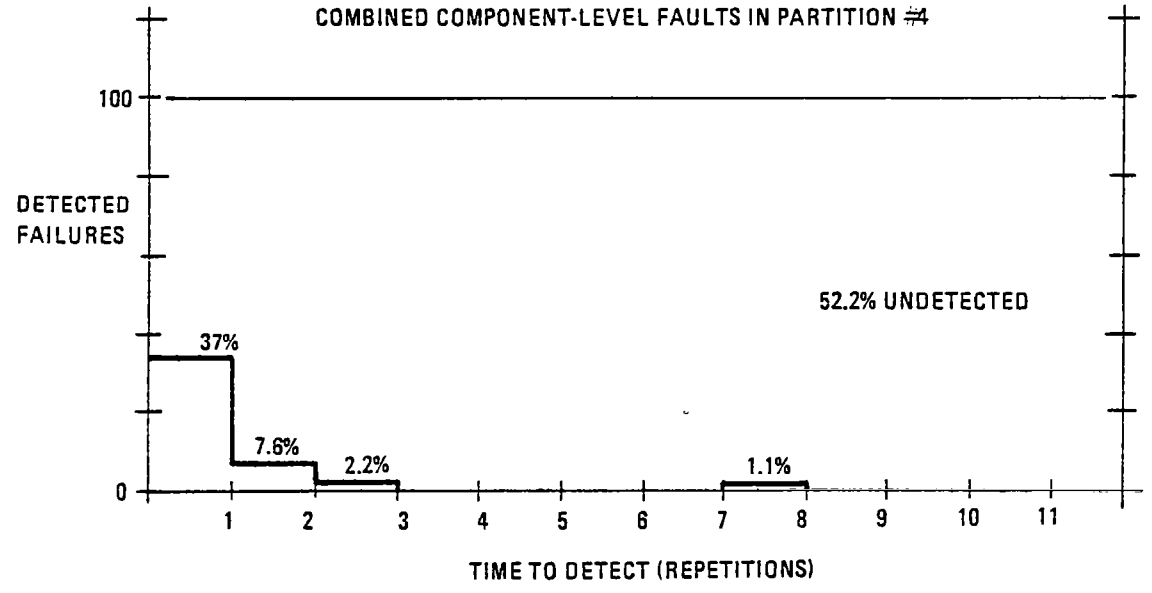

COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #4
100
DETECTED FAILURES
52.2% UNDETECTED
37%
7.6%
2.2%
1.1%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 21

ADDSUB LATENCY DATA

GATE-LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	m_1	n_1	m_2	n_2	m_3	n_3	m_4	n_4	m_5	n_5	m_6	n_6	m_7	n_7	m_8	n_8	m	n
P1	26	27	0	0	4	0	0	0	0	0	0	0	0	0	0	0	49	45
P2	32	46	7	4	0	3	0	0	1	0	2	0	1	0	0	0	90	78
P3	17	27	0	1	2	2	0	0	0	0	0	0	0	0	0	0	46	66
P4	9	17	0	4	4	2	0	2	1	1	0	0	0	0	0	0	53	57
P5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	52	52
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	6
TOTAL	84	117	7	9	10	8	0	2	2	1	2	0	1	0	0	0	296	304

m_1 = detected S-a-0 faults, ith cell n_1 = detected S-a-1 faults, ith cell

COMPONENT-LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	m_1	n_1	m_2	n_2	m_3	n_3	m_4	n_4	m_5	n_5	m_6	n_6	m_7	n_7	m_8	n_8	m	n
P1	22	23	0	0	2	1	0	0	0	0	0	0	0	0	0	0	38	39
P2	35	51	11	10	1	3	0	0	0	0	1	1	0	0	0	0	58	79
P3	19	23	1	0	3	2	0	0	0	0	0	0	0	0	0	0	52	42
P4	24	31	4	1	1	1	0	0	0	0	0	0	0	0	0	0	49	43
P5																		
P6																		
TOTAL	100	128	16	11	7	7	0	0	0	0	1	1	0	0	0	0	197	203

TABLE 6

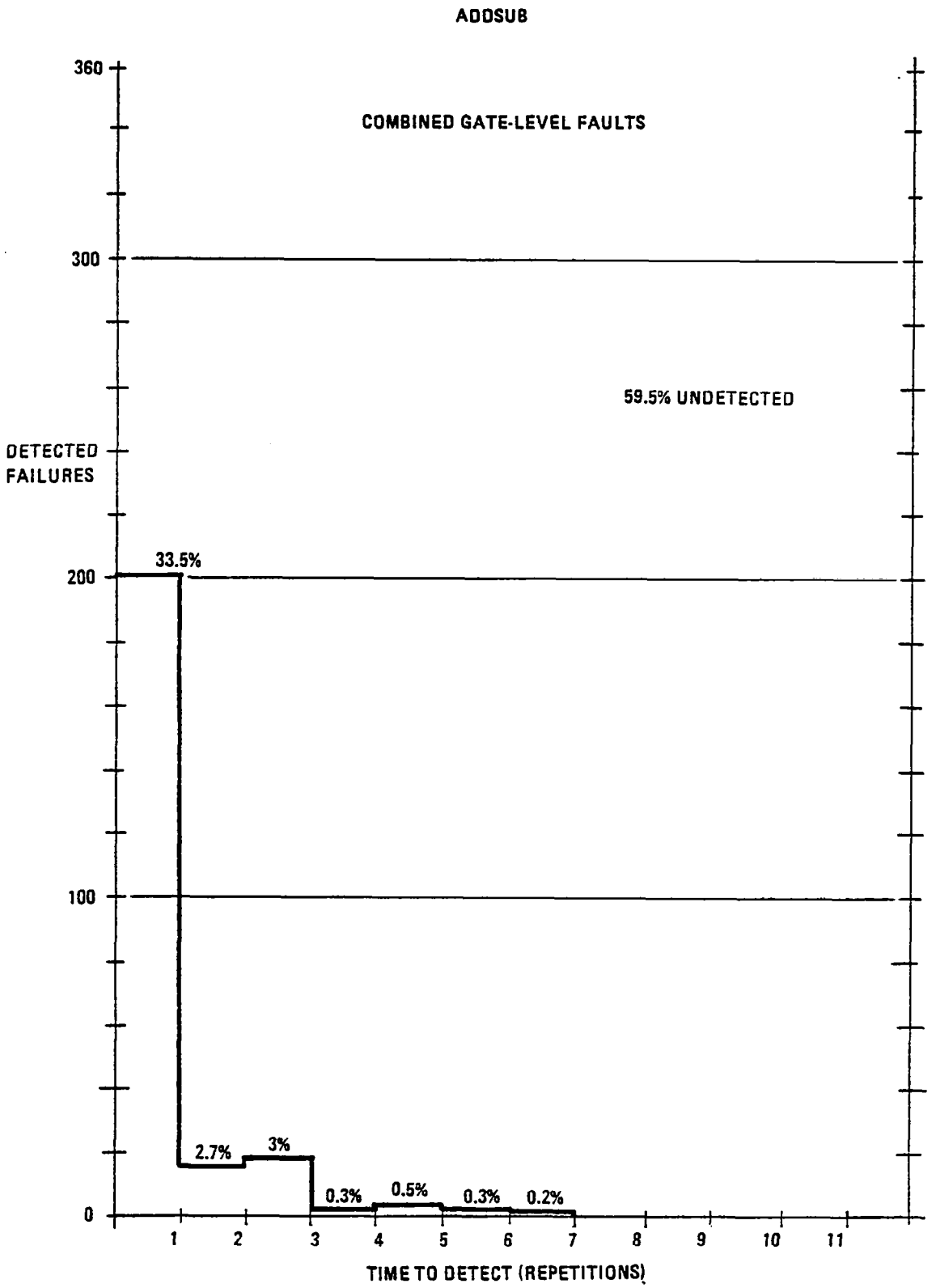
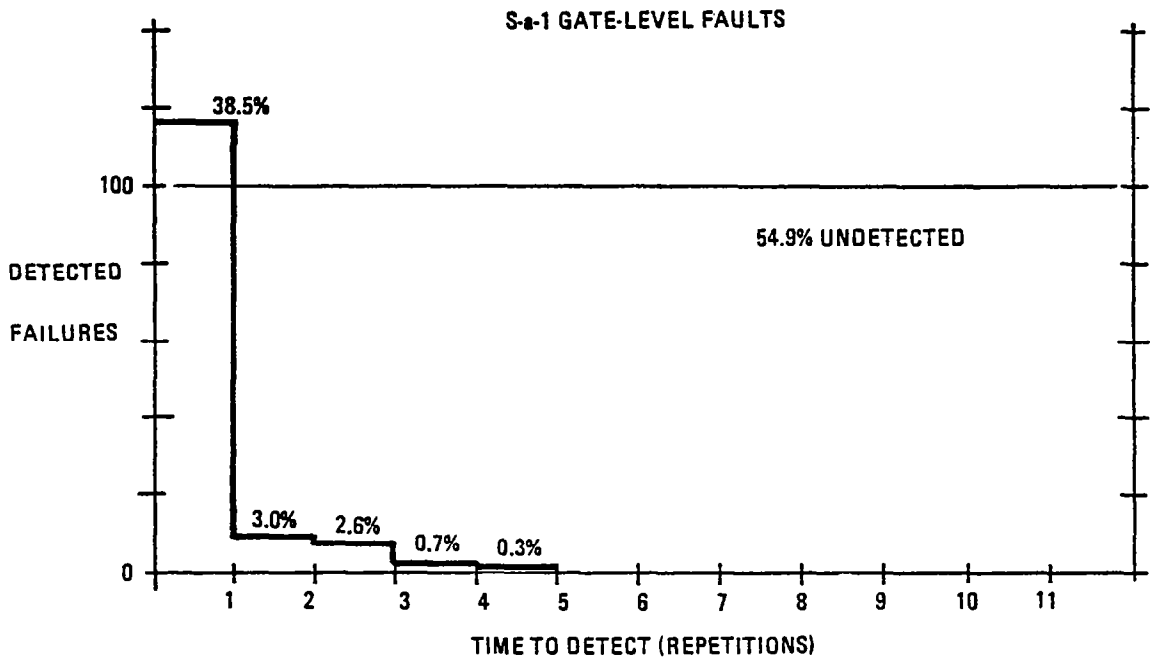

ADDSUB
COMBINED GATE-LEVEL FAULTS
360
300
200
100
0
DETECTED FAILURES
59.5% UNDETECTED
33.5%
2.7%
3%
0.3%
0.5%
0.3%
0.2%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 3a

ADDSUB



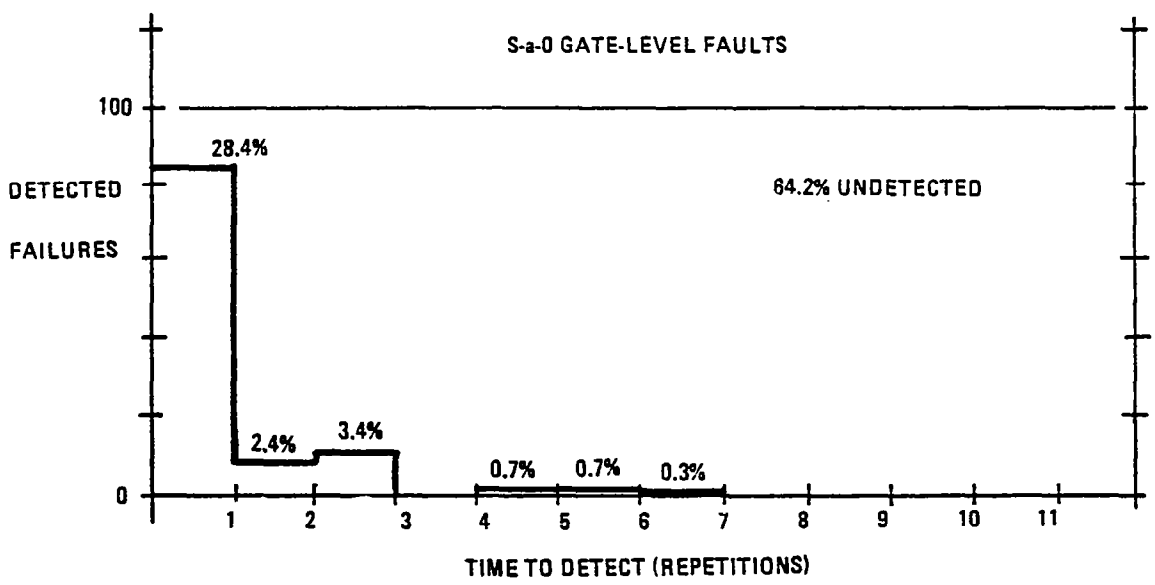
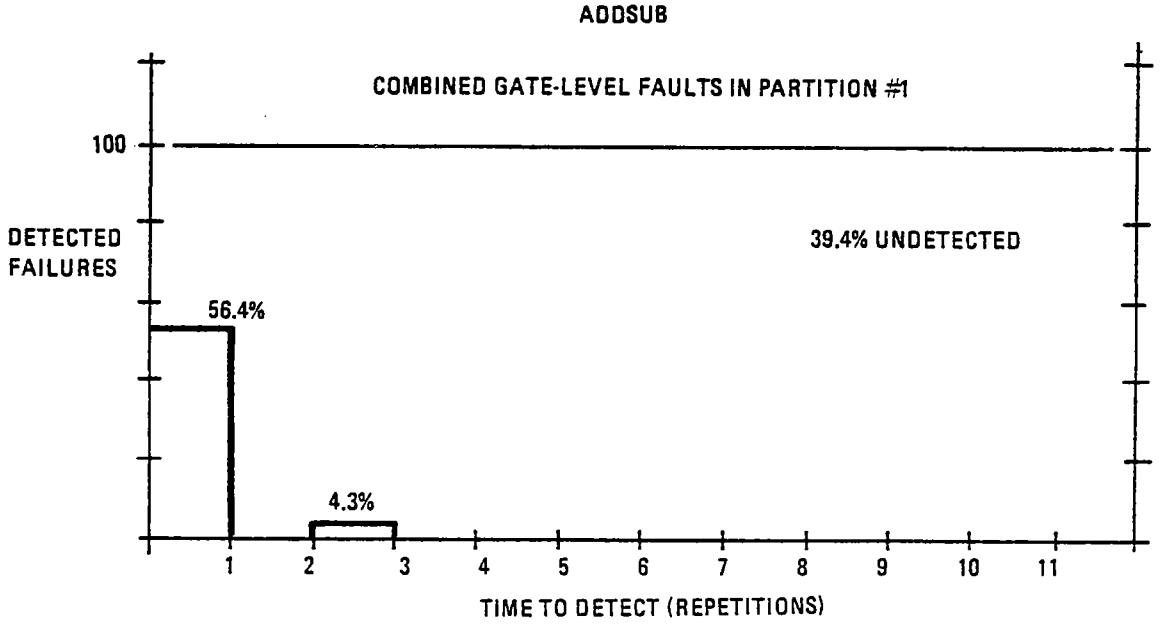


FIGURE 3b


ADDSUB
COMBINED GATE-LEVEL FAULTS IN PARTITION #1
100
DETECTED FAILURES
39.4% UNDETECTED
56.4%
4.3%
1 2 3 4 5 6 7 8 9 10 11
TIME TO DETECT (REPETITIONS)

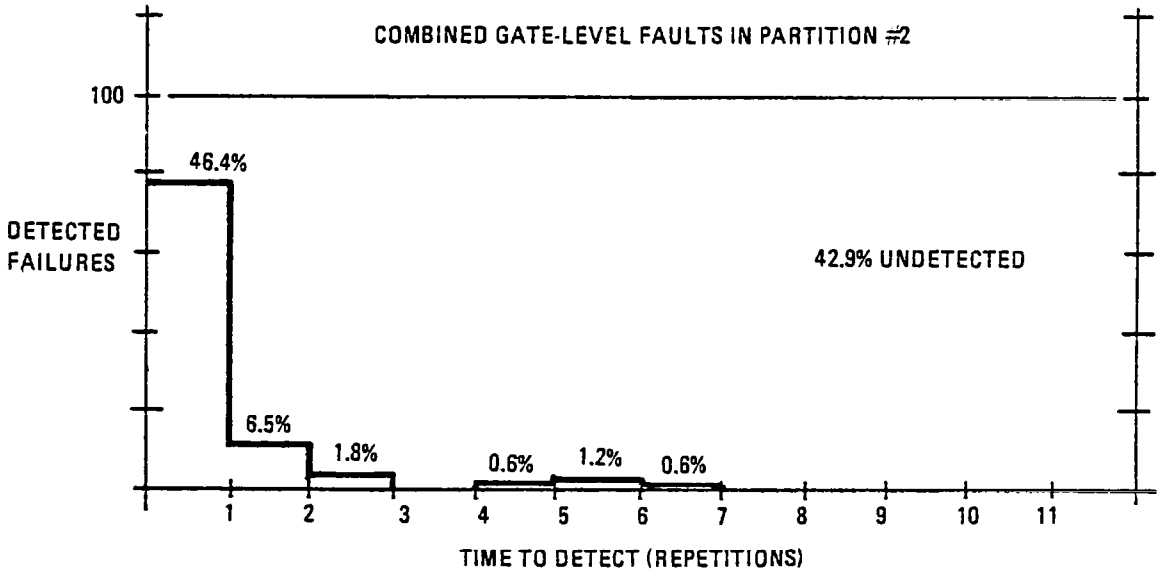
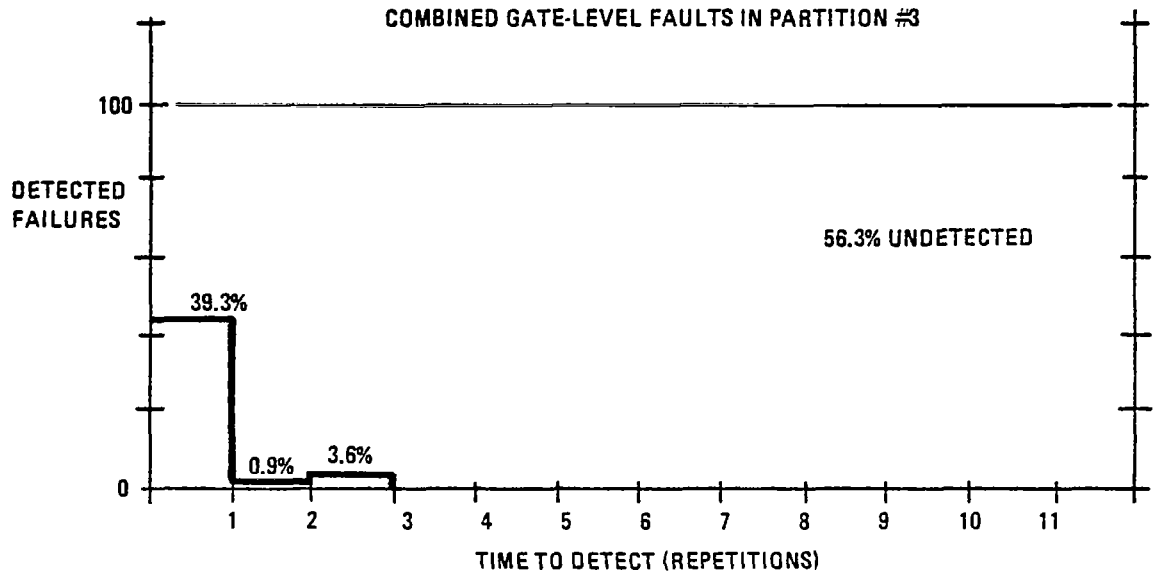

COMBINED GATE-LEVEL FAULTS IN PARTITION #2
100
DETECTED FAILURES
42.9% UNDETECTED
46.4%
6.5%
1.8%
0.6%
1.2%
0.6%
1 2 3 4 5 6 7 8 9 10 11
TIME TO DETECT (REPETITIONS)

FIGURE 3c

ADDSUB



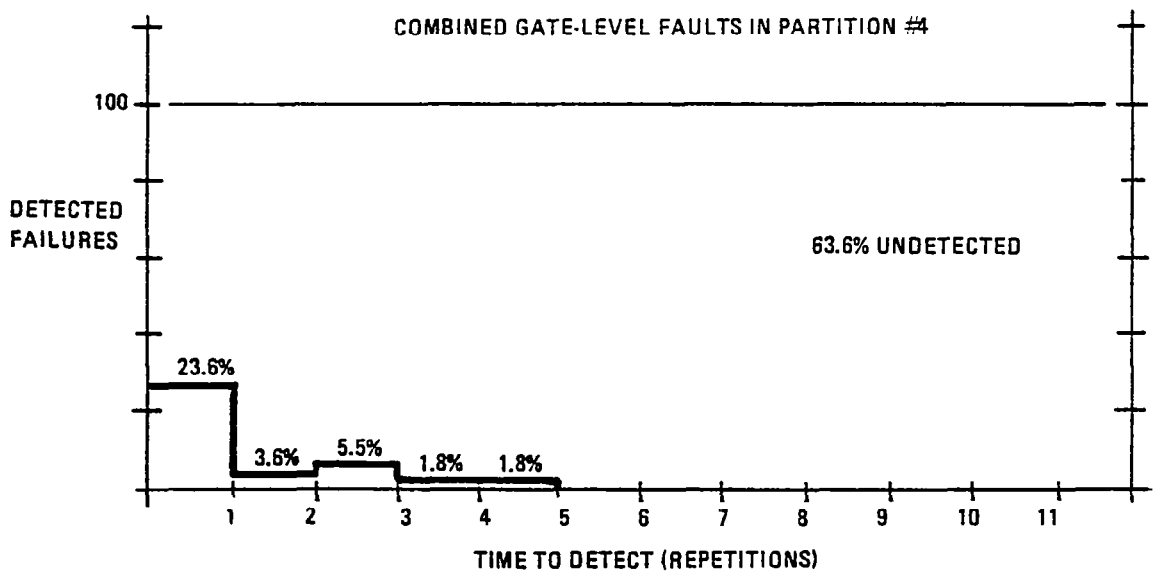
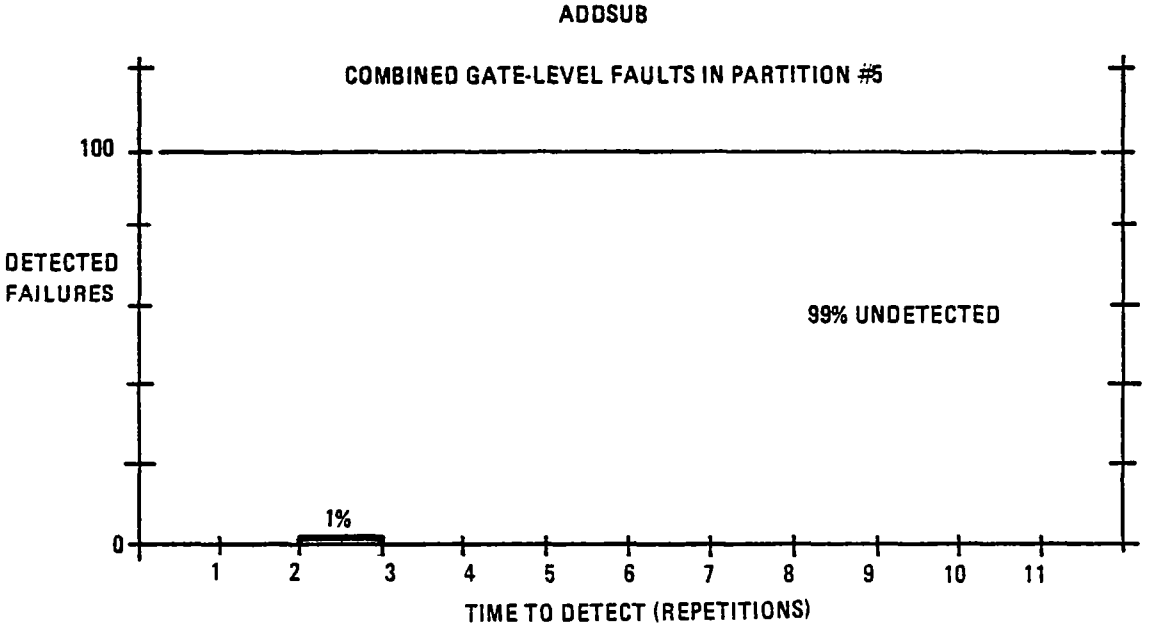


FIGURE 3d


ADDSUB
COMBINED GATE-LEVEL FAULTS IN PARTITION #5
100
DETECTED
FAILURES
99% UNDETECTED
1%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

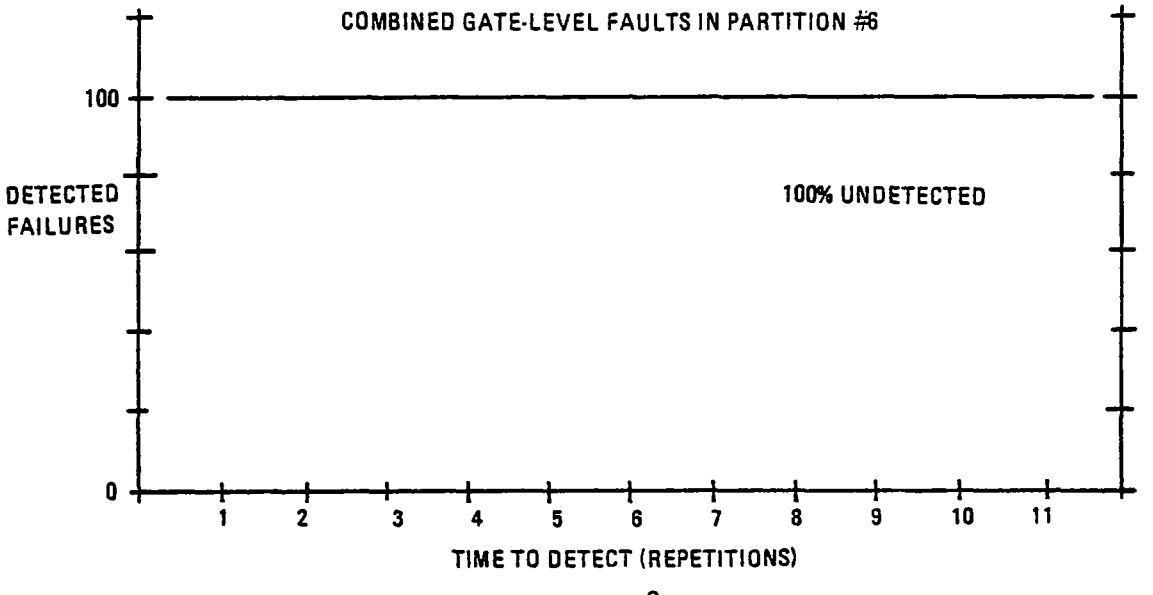

COMBINED GATE-LEVEL FAULTS IN PARTITION #6
100
DETECTED
FAILURES
100% UNDETECTED
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 3e

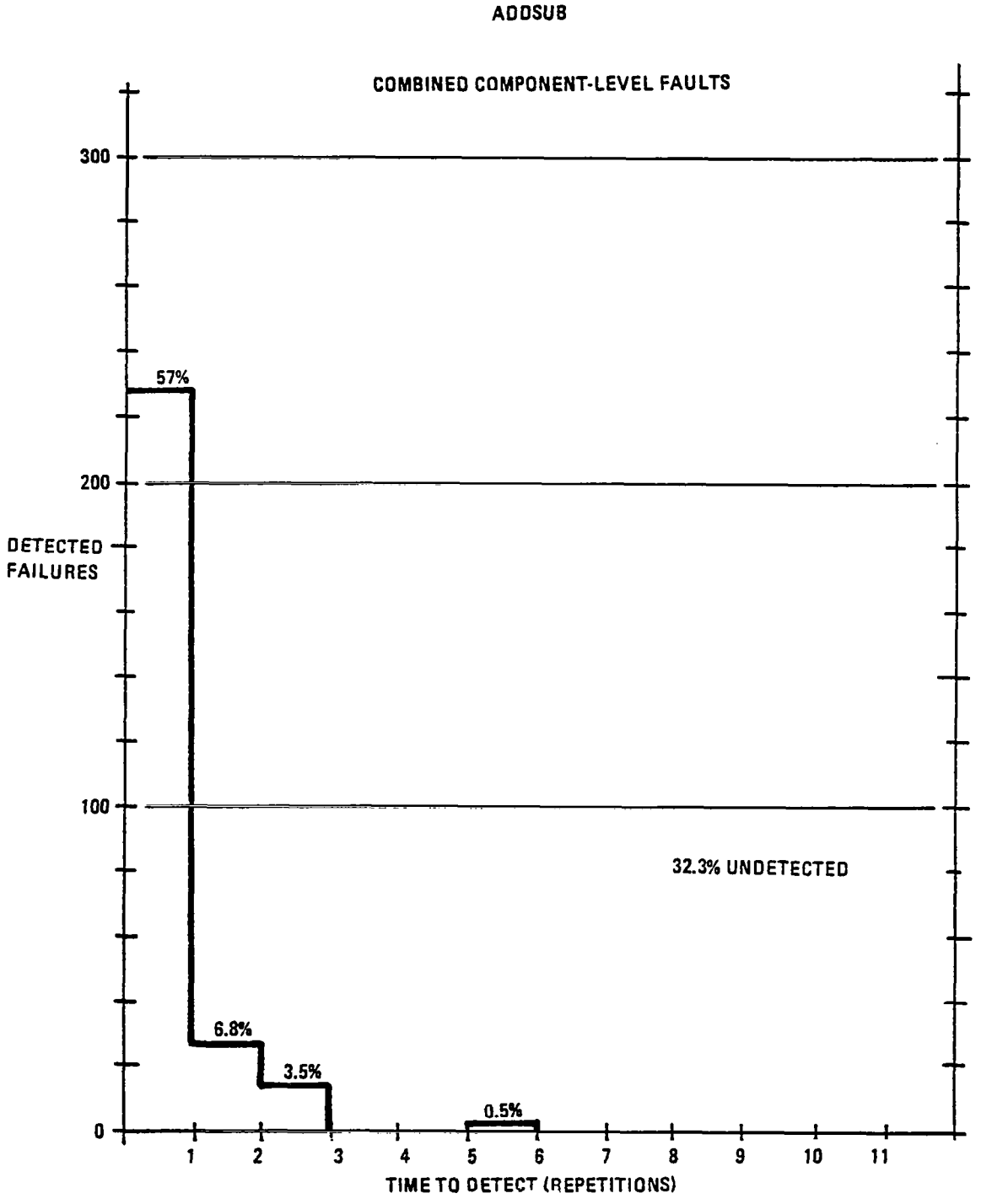
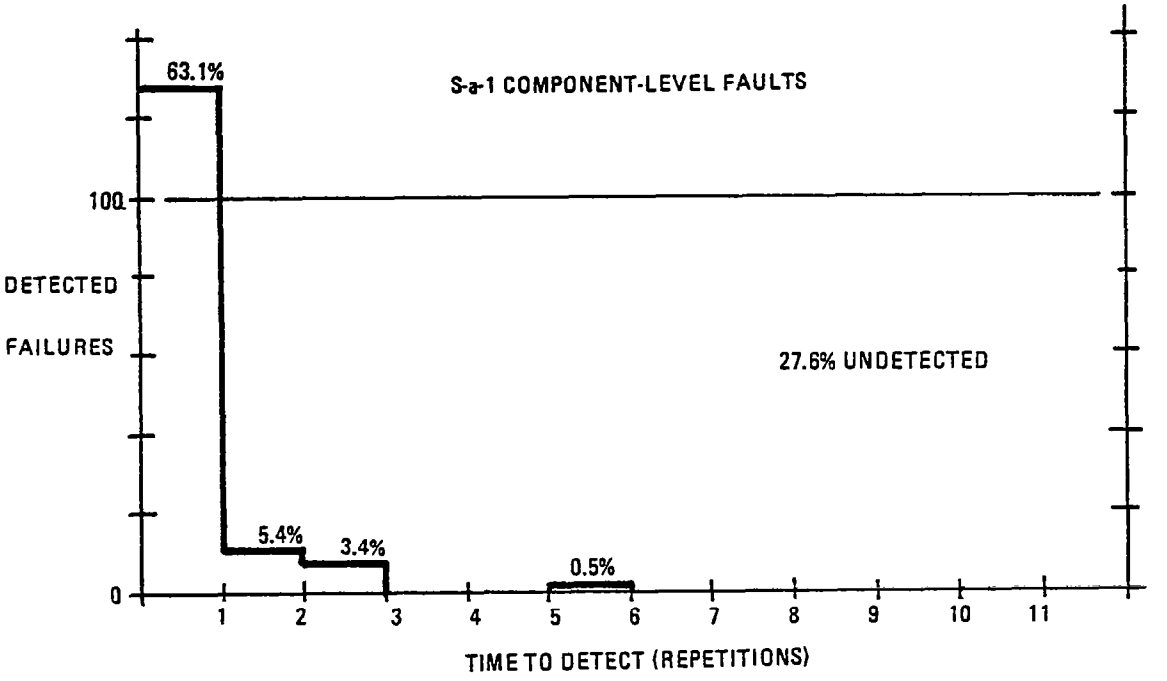

ADDSUB
COMBINED COMPONENT-LEVEL FAULTS
300
200
100
0
DETECTED FAILURES
57%
6.8%
3.5%
0.5%
32.3% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 3f

ADDSUB



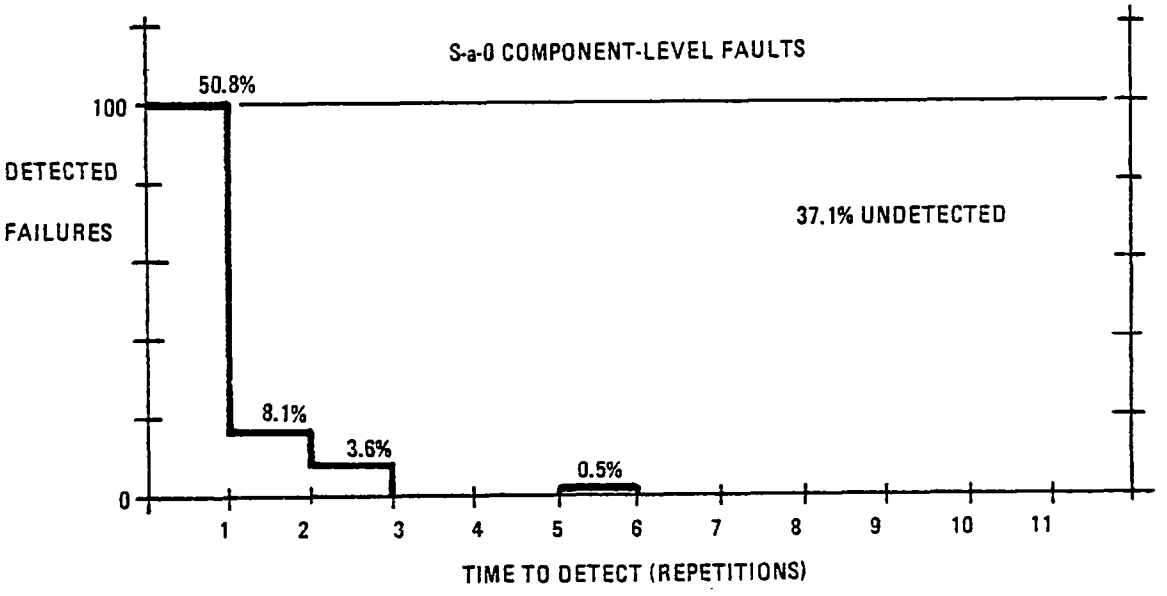
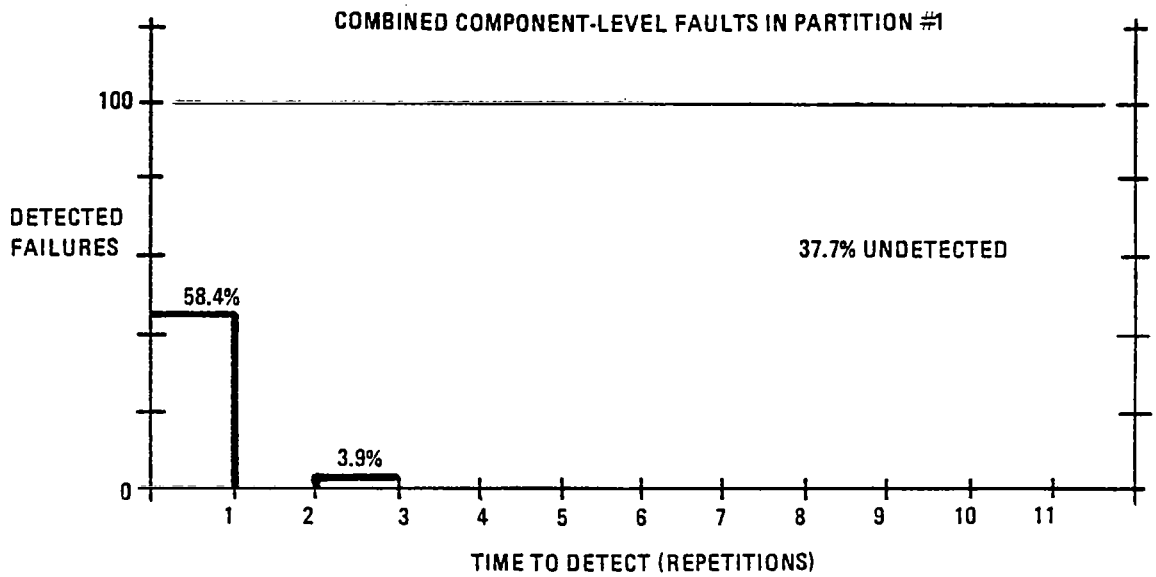


FIGURE 3g

ADDSUB



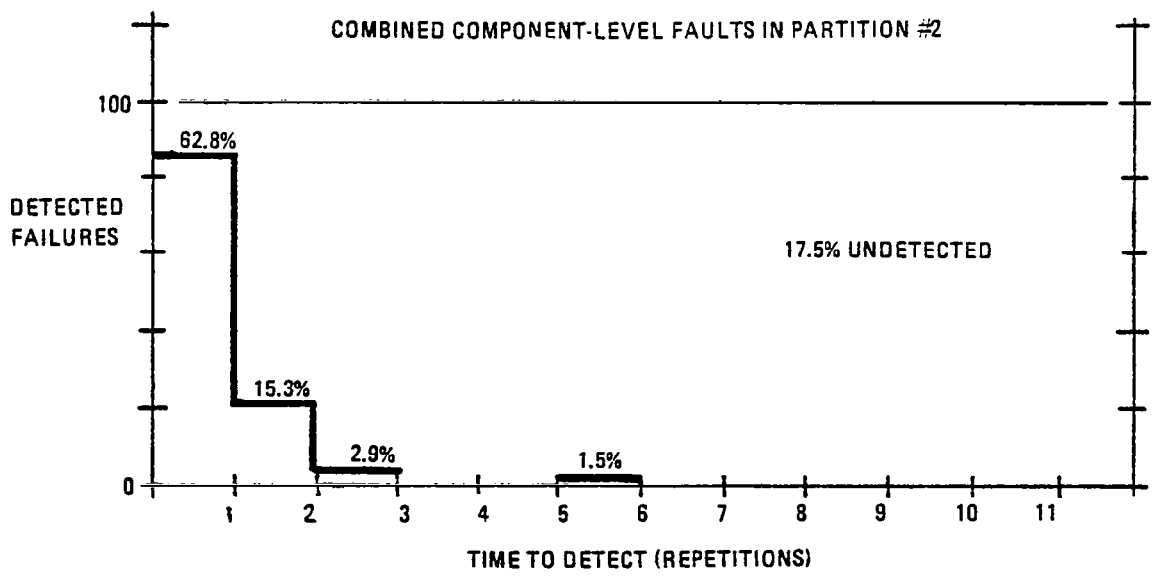
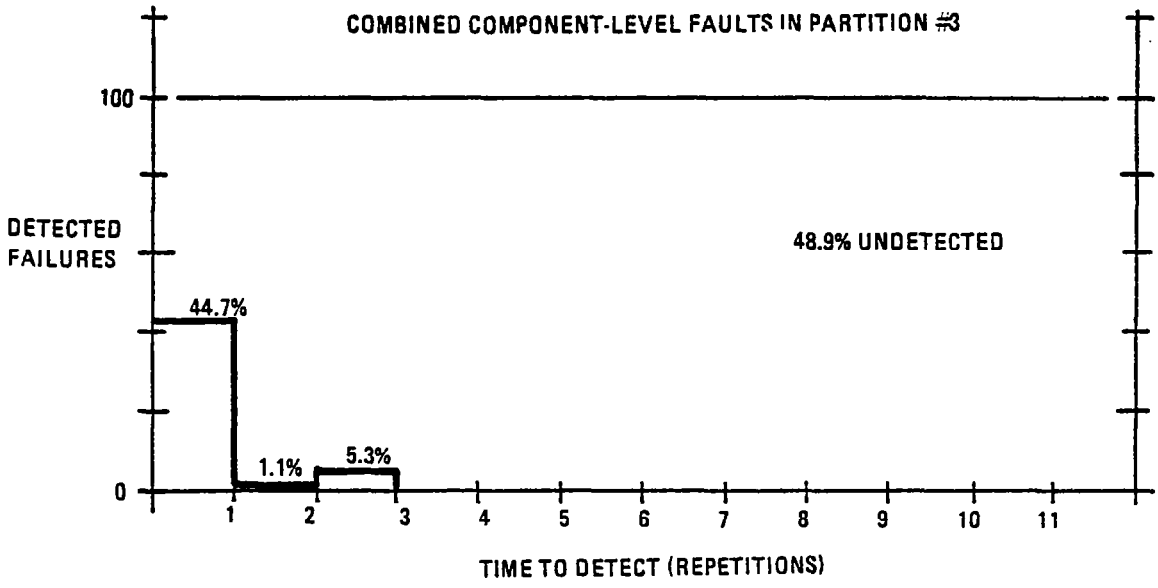


FIGURE 3h

ADDSUB



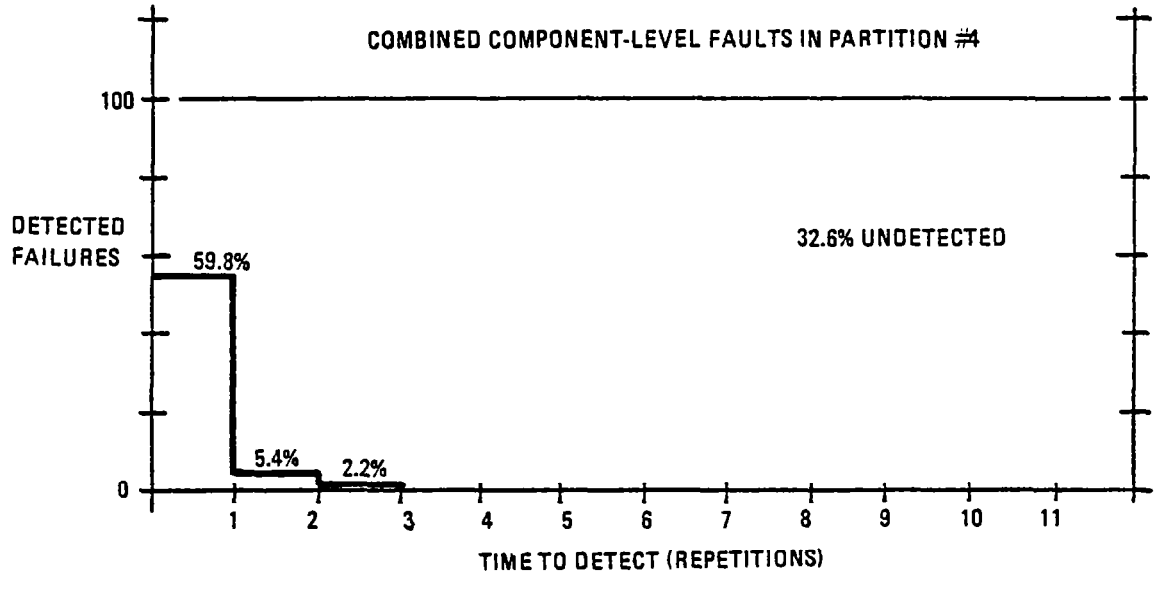


FIGURE 3i

FIB LATENCY DATA

GATE-LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	m_1	n_1	m_2	n_2	m_3	n_3	m_4	n_4	m_5	n_5	m_6	n_6	m_7	n_7	m_8	n_8	m	n
P1	30	29	4	1	0	0	0	0	0	0	0	0	0	0	0	0	49	45
P2	33	48	10	2	1	2	0	0	0	1	0	0	0	0	0	0	90	78
P3	17	27	2	4	0	0	0	0	0	0	0	0	0	0	0	0	46	66
P4	10	15	3	6	0	1	1	0	0	0	0	1	0	0	1	0	53	57
P5	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	52	52
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	6
TOTAL	90	120	19	14	1	3	1	0	0	1	0	1	0	0	1	0	296	304

m_1 = detected S-a-0 faults, ith cell n_1 = detected S-a-1 faults, ith cell

COMPONENT-LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	m_1	n_1	m_2	n_2	m_3	n_3	m_4	n_4	m_5	n_5	m_6	n_6	m_7	n_7	m_8	n_8	m	n
P1	21	29	5	2	0	0	0	0	0	0	0	0	0	0	0	0	38	39
P2	43	60	6	2	0	2	0	1	0	1	0	1	0	0	0	0	58	79
P3	22	22	3	3	0	0	0	0	0	0	0	0	0	0	0	0	52	42
P4	25	23	6	9	0	1	0	0	0	0	0	0	0	0	0	1	49	43
P5																		
P6																		
TOTAL	111	134	20	16	0	3	0	1	0	1	0	1	0	0	0	1	197	203

TABLE 7

FIB

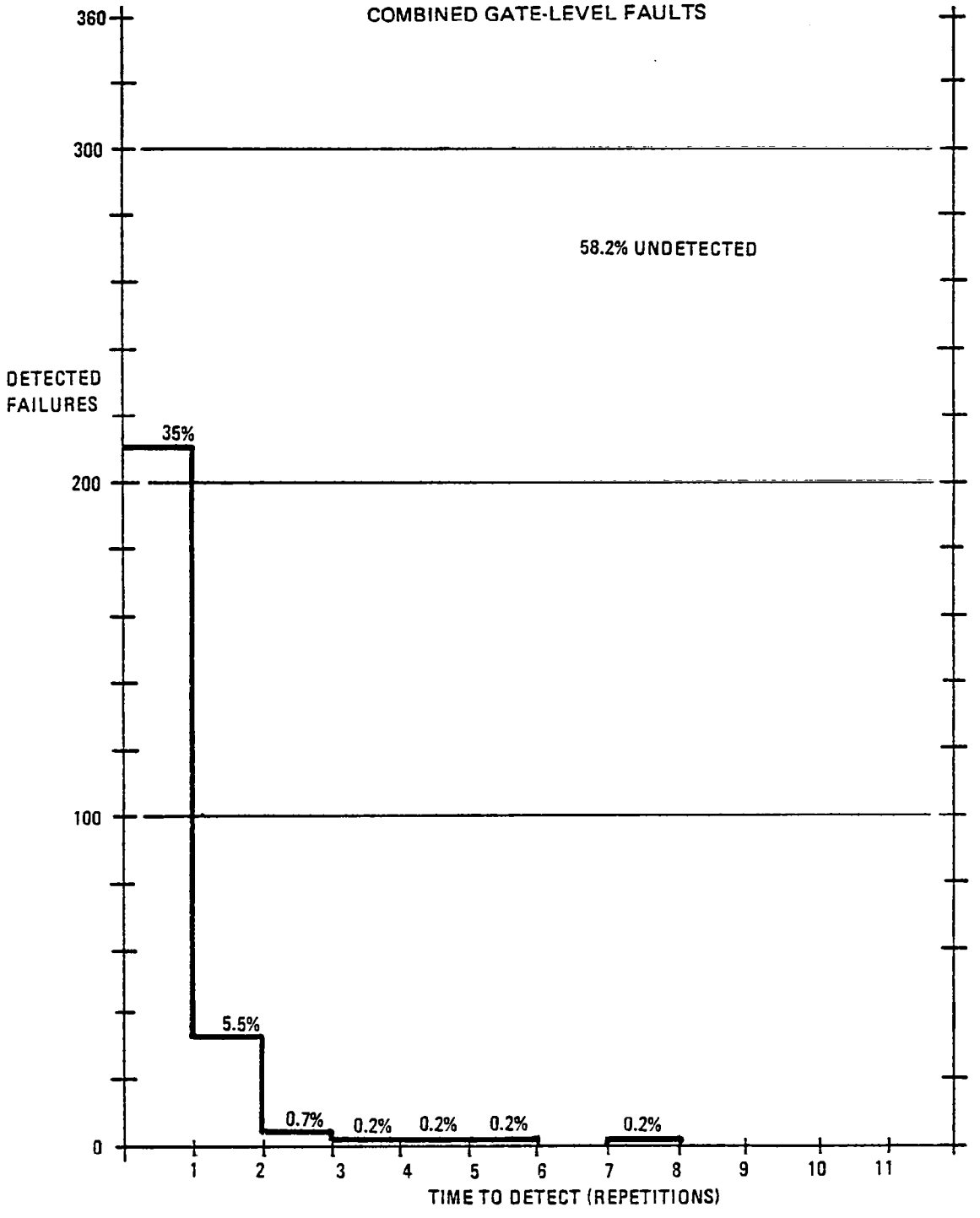
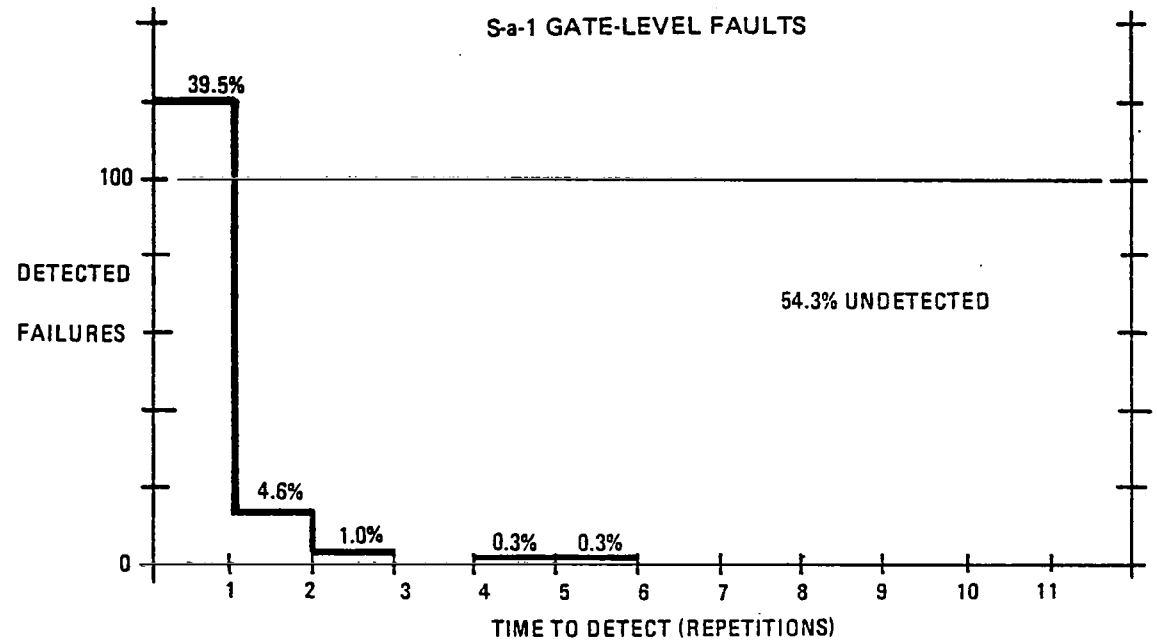


FIGURE 4a

FIB



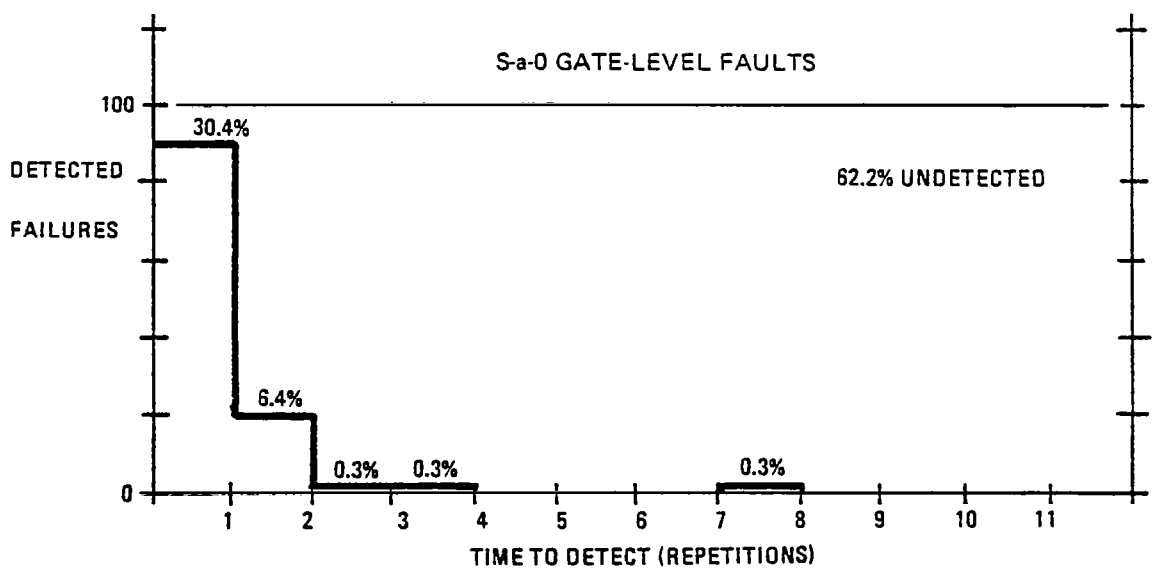
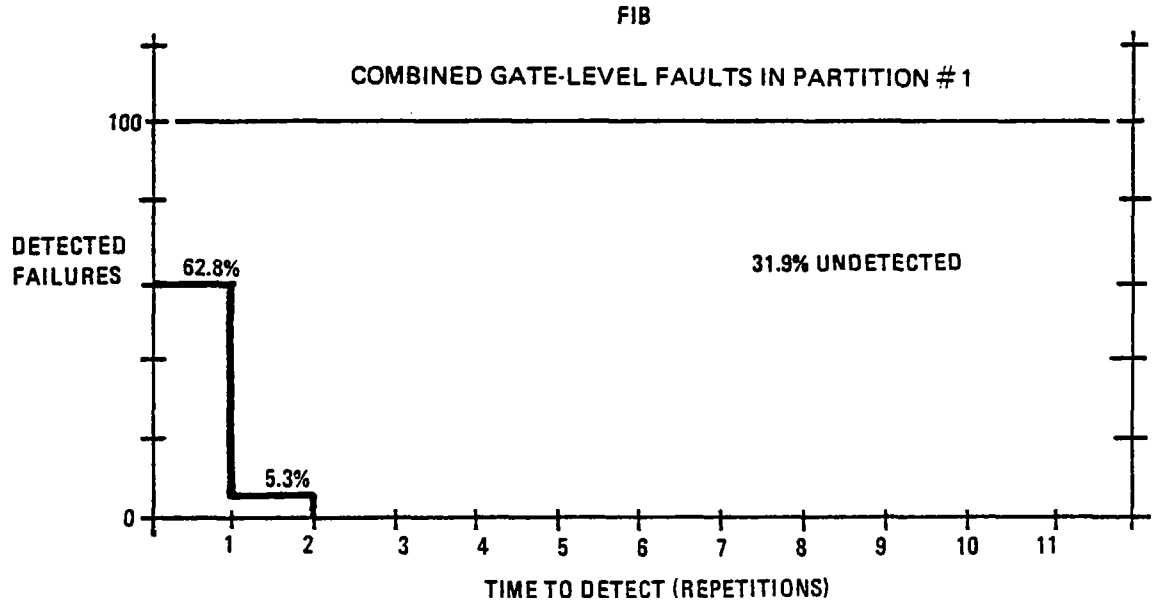


FIGURE 4b


FIB
COMBINED GATE-LEVEL FAULTS IN PARTITION #1
100
DETECTED FAILURES
62.8%
31.9% UNDETECTED
5.3%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

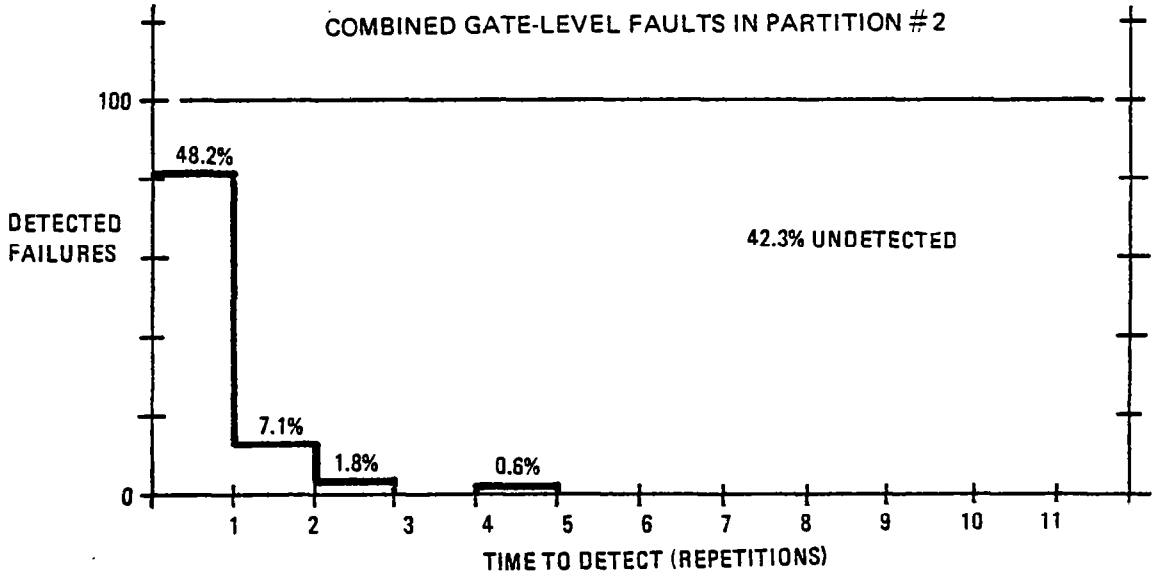
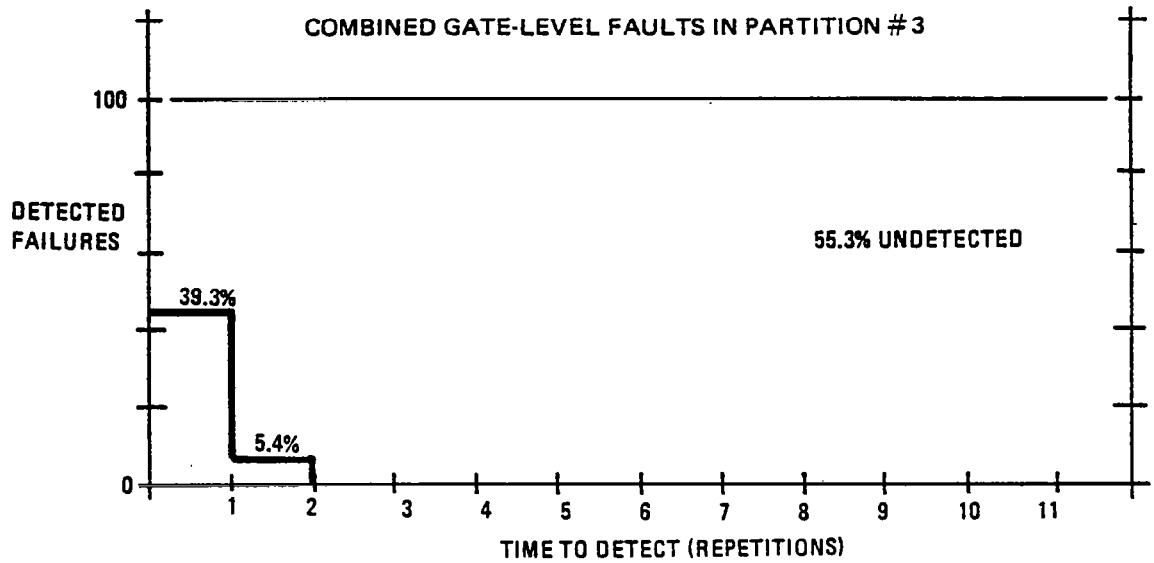

COMBINED GATE-LEVEL FAULTS IN PARTITION #2
100
48.2%
DETECTED FAILURES
42.3% UNDETECTED
7.1%
1.8%
0.6%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 4c

FIB



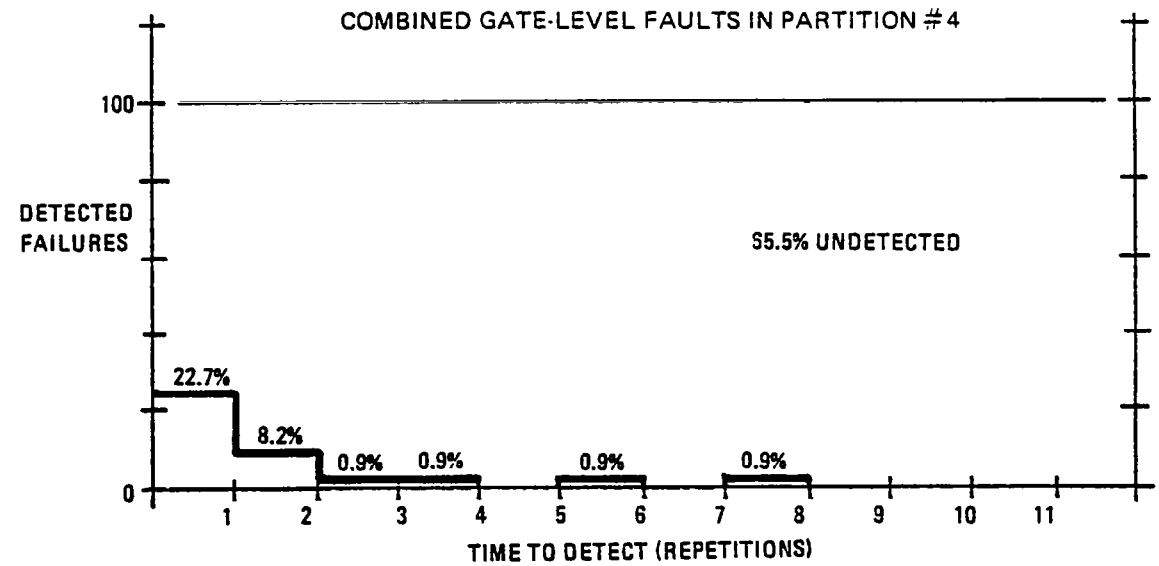
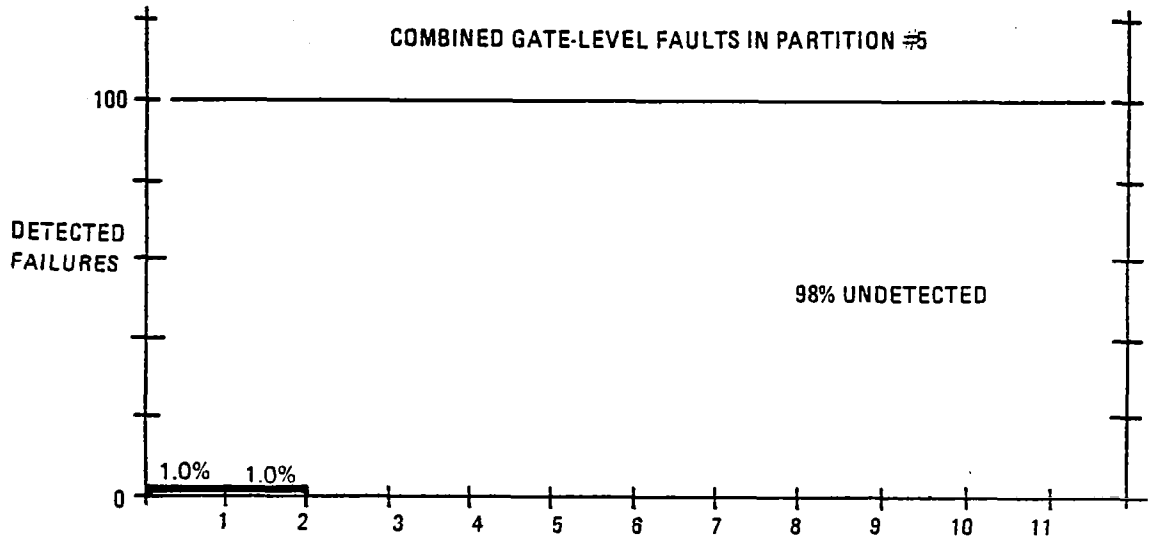


FIGURE 4d

FIB



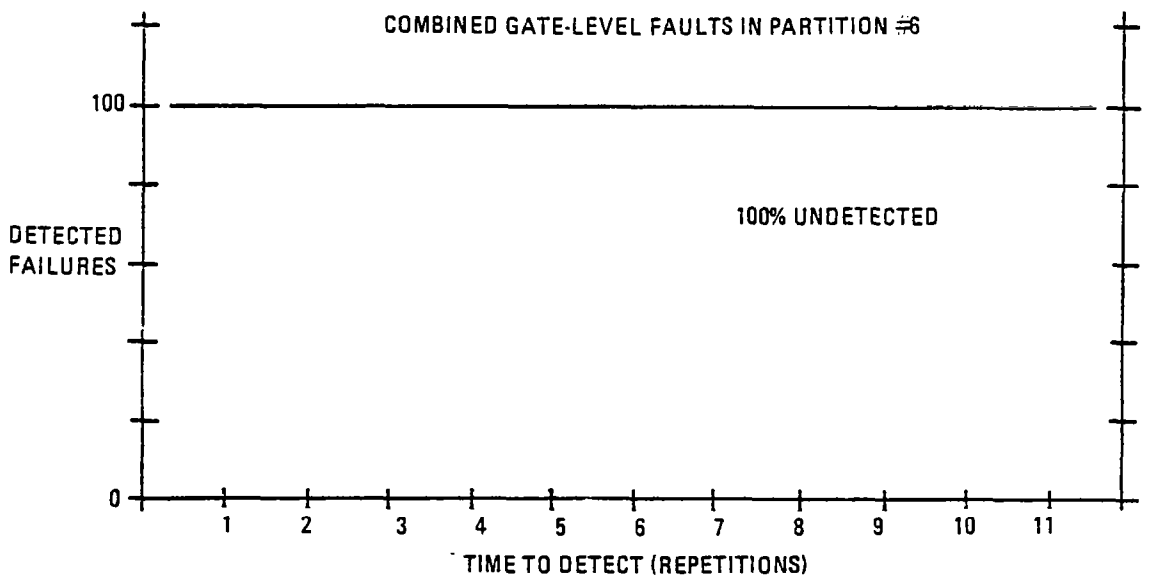


FIGURE 4e

FIB

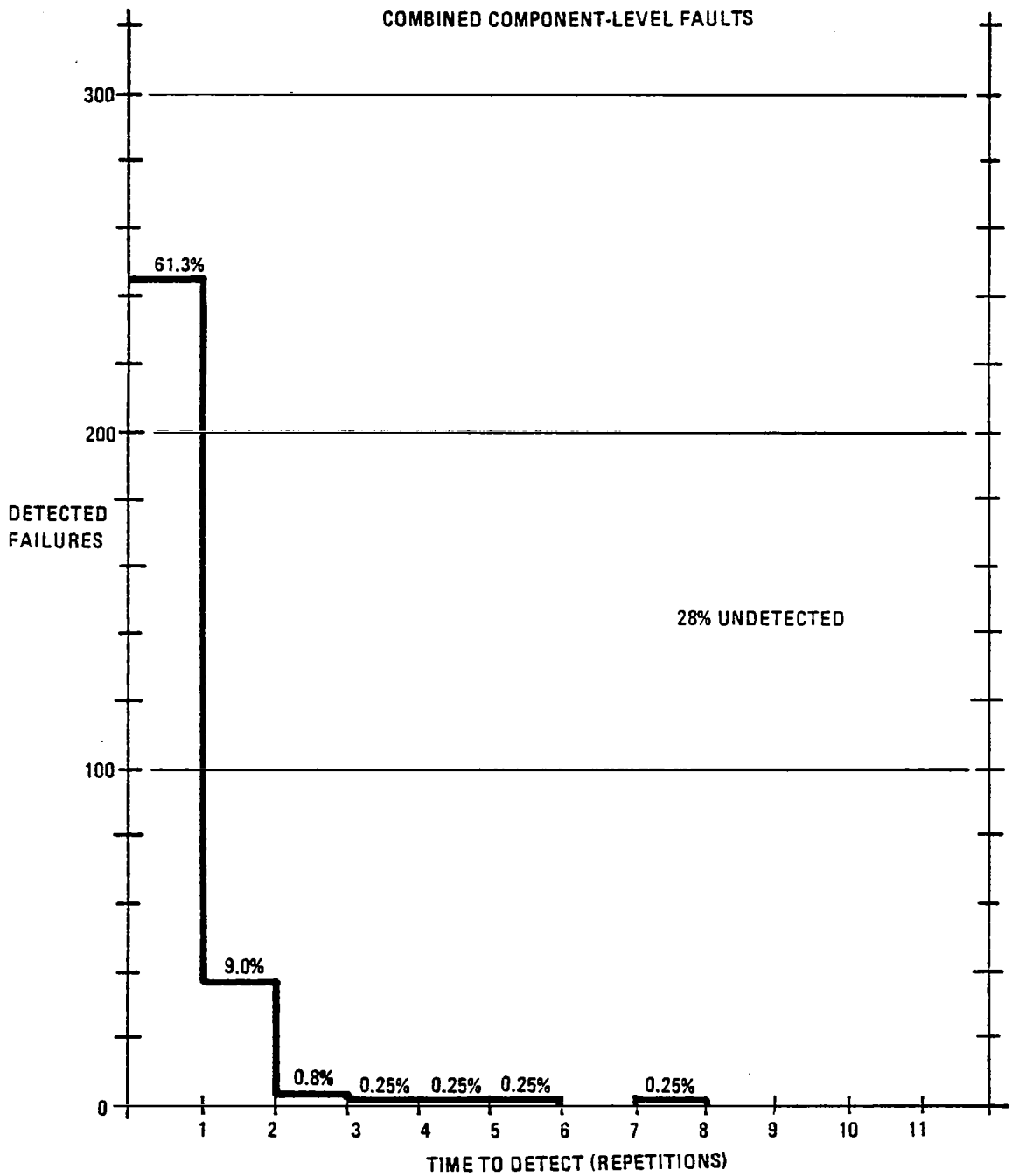
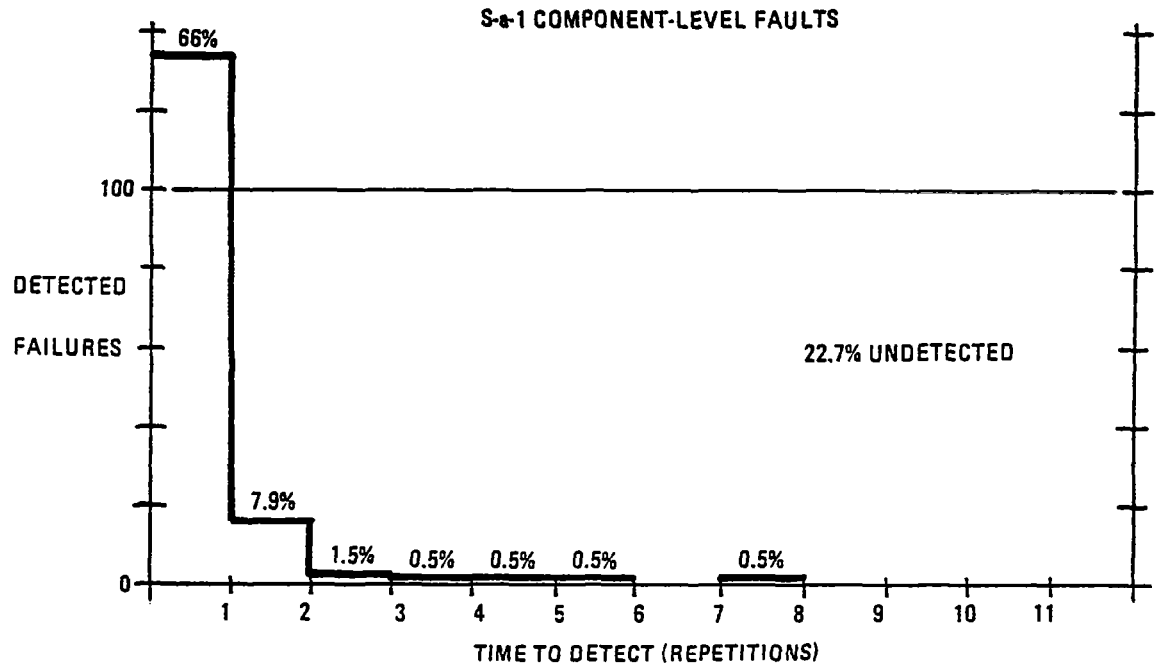


FIGURE 4f


S-a-1 COMPONENT-LEVEL FAULTS
66%
100
DETECTED
FAILURES
22.7% UNDETECTED
7.9%
1.5%
0.5%
0.5%
0.5%
0.5%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

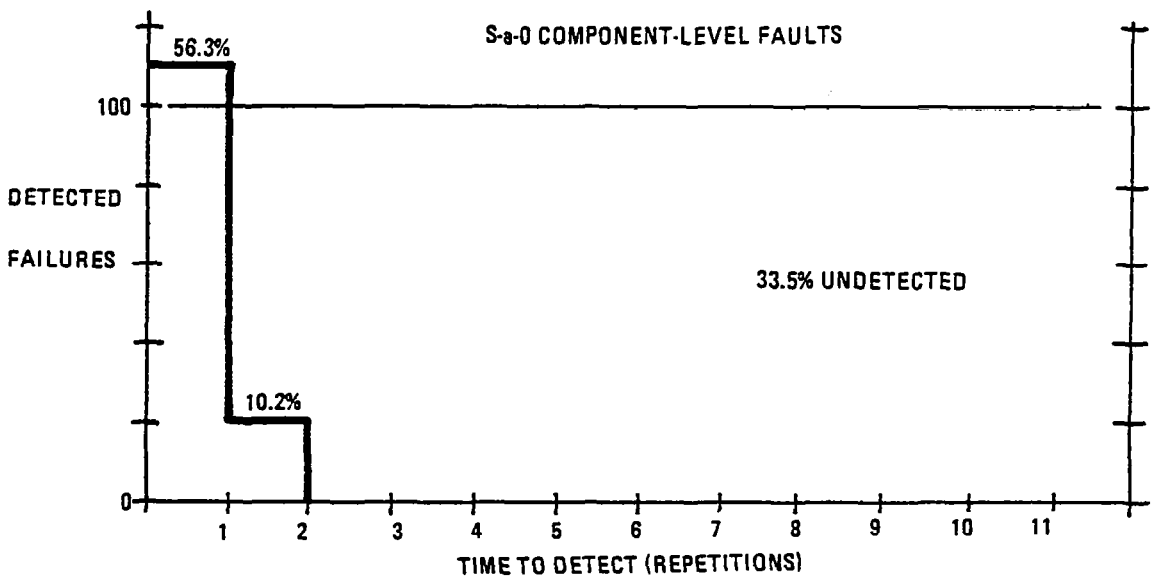
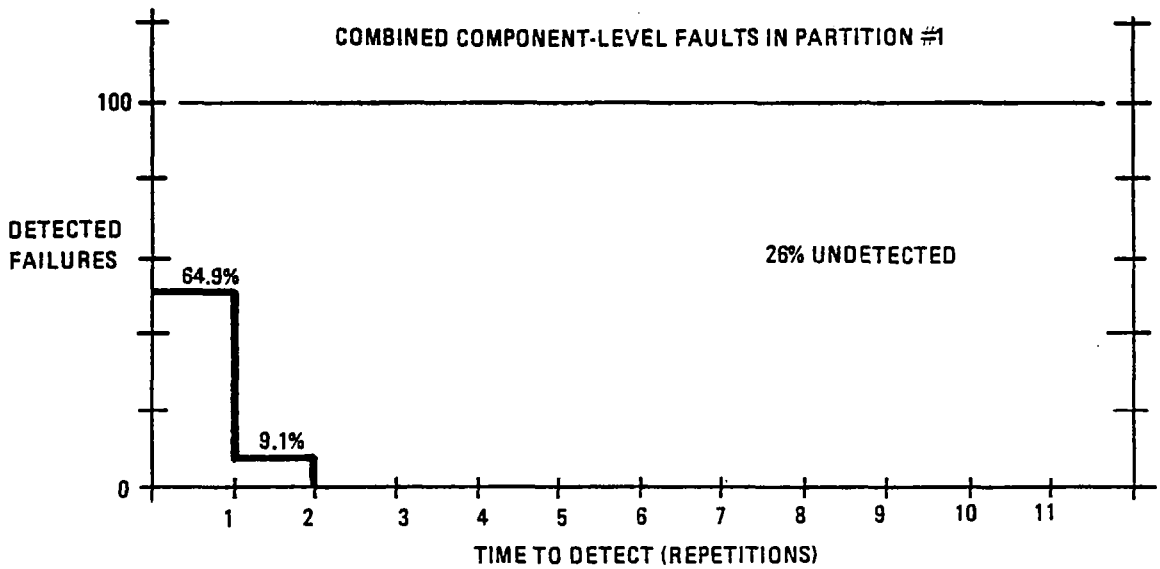

S-a-0 COMPONENT-LEVEL FAULTS
56.3%
100
DETECTED
FAILURES
33.5% UNDETECTED
10.2%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 4g

FIB



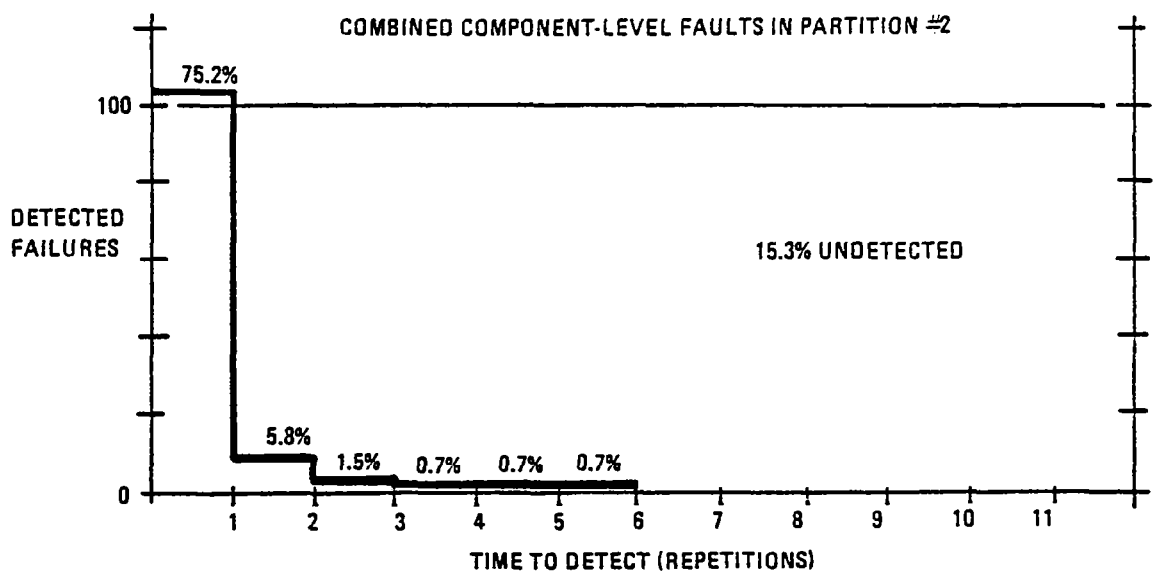
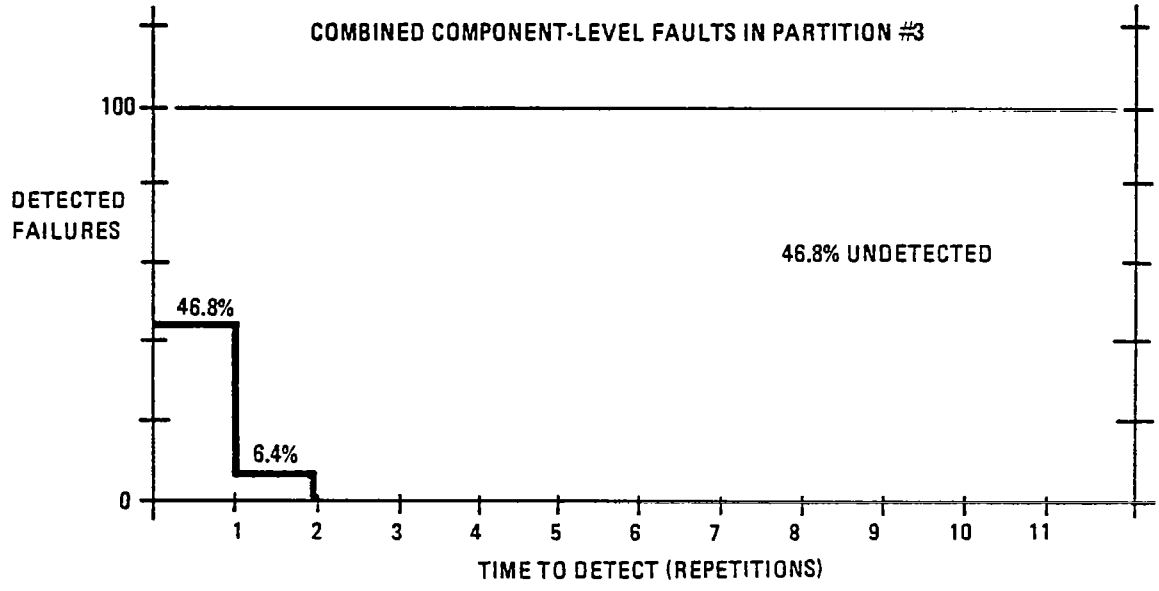


FIGURE 4h

FIB



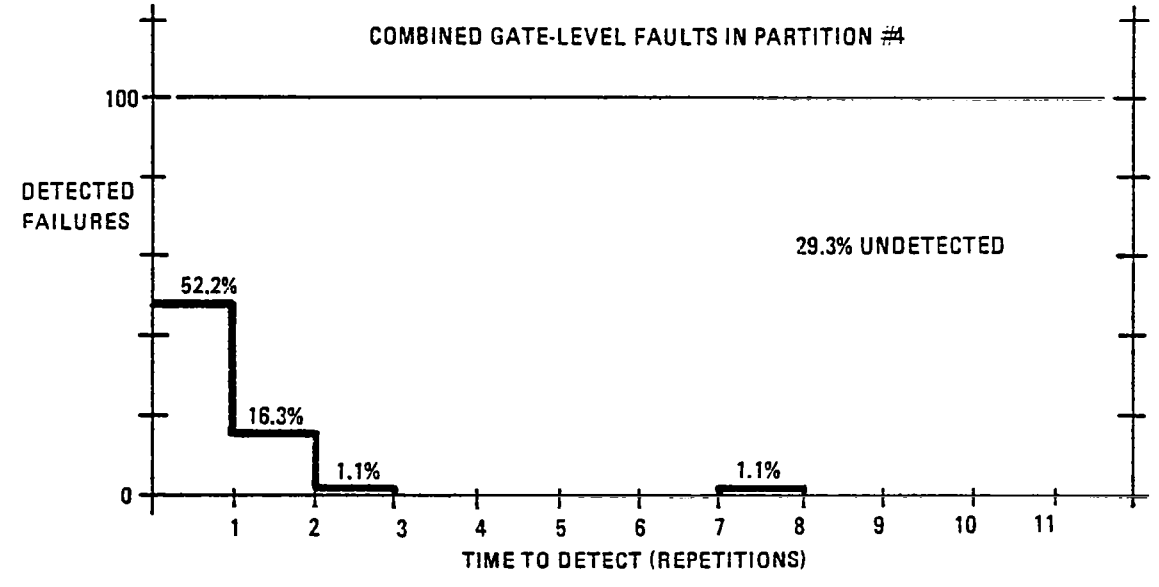


FIGURE 4i

QUAD LATENCY DATA

GATE-LEVEL FAULTS

PARTITION	DETECTED FAULTS								FAULTS INJECTED	
	m_1	n_1	m_2	n_2	m_3	n_3	m_4	n_4	m	n
P1	35	29	0	0	0	0	1	0	49	45
P2	40	56	4	4	2	0	2	1	90	78
P3	23	31	0	0	0	0	0	0	46	66
P4	15	27	3	3	0	0	0	1	53	57
P5	1	2	0	0	0	0	0	0	52	52
P6	0	0	0	0	0	0	0	0	6	6
TOTAL	114	145	7	7	2	0	3	2	296	304

m_1 = detected S-a-0 faults, ith cell

n_1 = detected S-a-1 faults, ith cell

COMPONENT-LEVEL FAULTS

PARTITION	DETECTED FAULTS								FAULTS INJECTED	
	m_1	n_1	m_2	n_2	m_3	n_3	m_4	n_4	m	n
P1	25	29	0	1	0	0	0	0	38	39
P2	50	68	2	0	0	0	0	3	58	79
P3	25	24	2	1	0	1	0	0	52	42
P4	33	33	7	2	0	0	0	0	49	43
P5										
P6										
TOTAL	133	154	11	4	0	1	0	3	197	203

TABLE 8

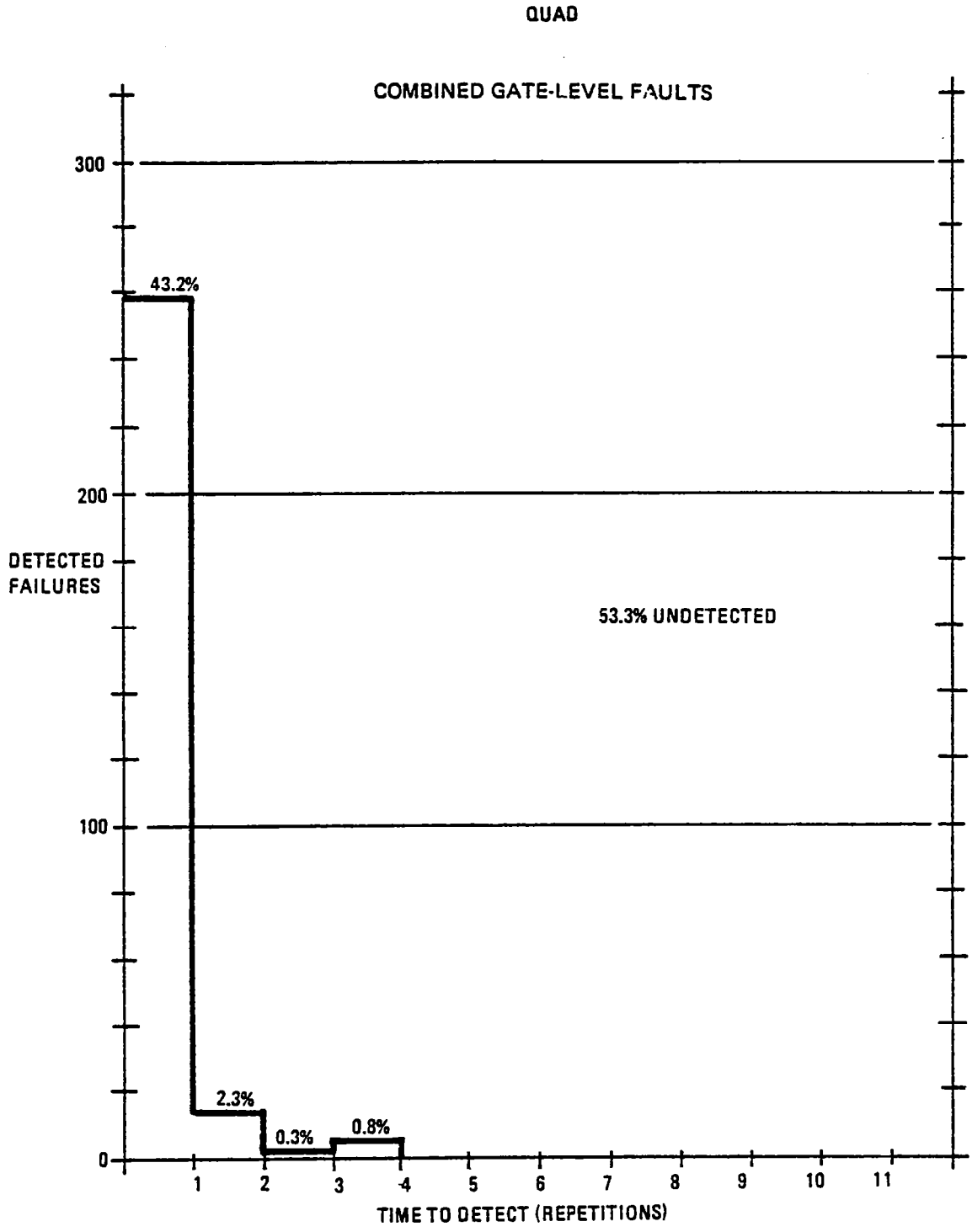

QUAD
COMBINED GATE-LEVEL FAULTS
300
200
100
0
DETECTED FAILURES
43.2%
53.3% UNDETECTED
2.3%
0.3%
0.8%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 5a

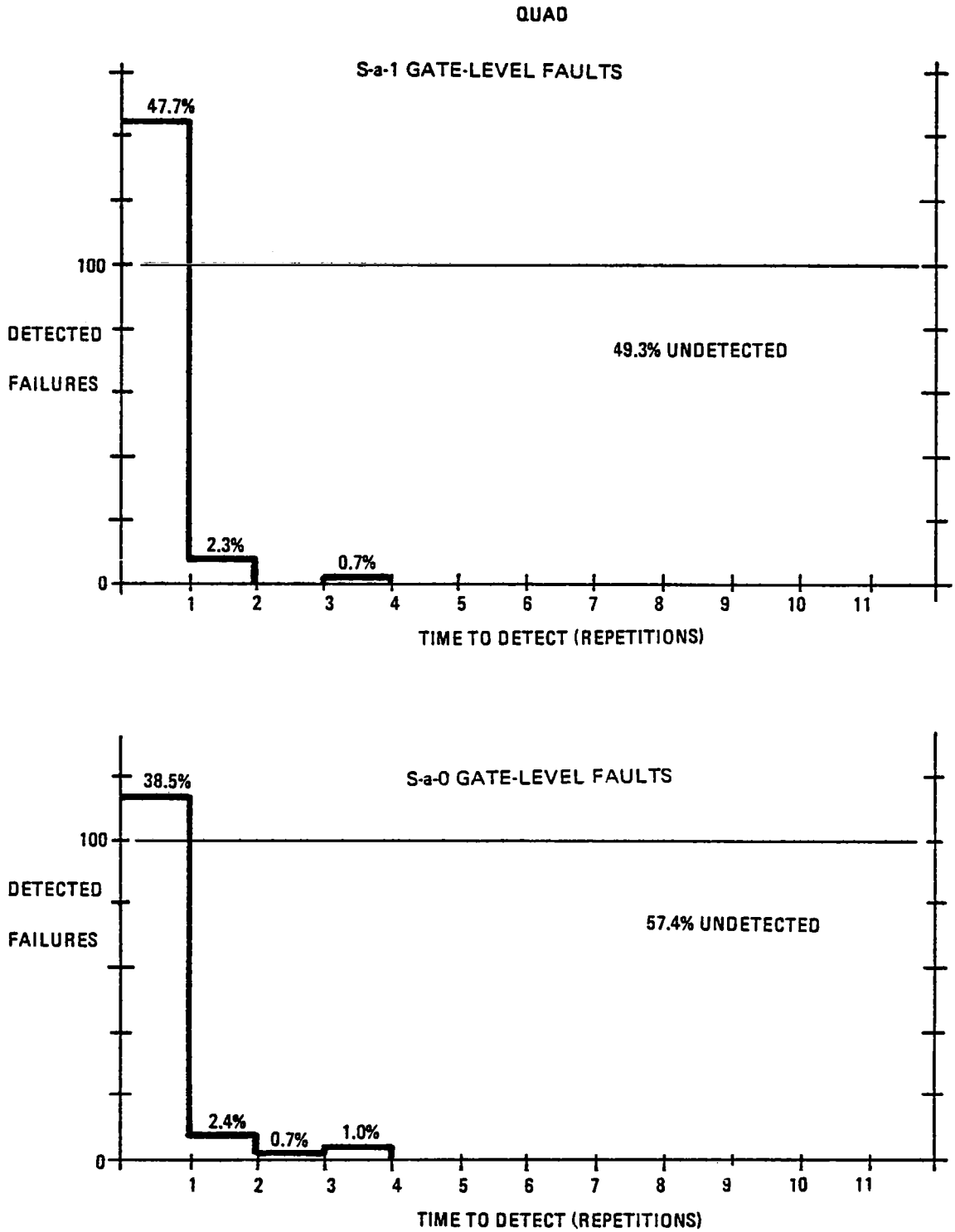

QUAD
S-a-1 GATE-LEVEL FAULTS
47.7%
100
DETECTED
FAILURES
49.3% UNDETECTED
2.3%
0.7%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)
S-a-0 GATE-LEVEL FAULTS
38.5%
100
DETECTED
FAILURES
57.4% UNDETECTED
2.4%
0.7%
1.0%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 5b

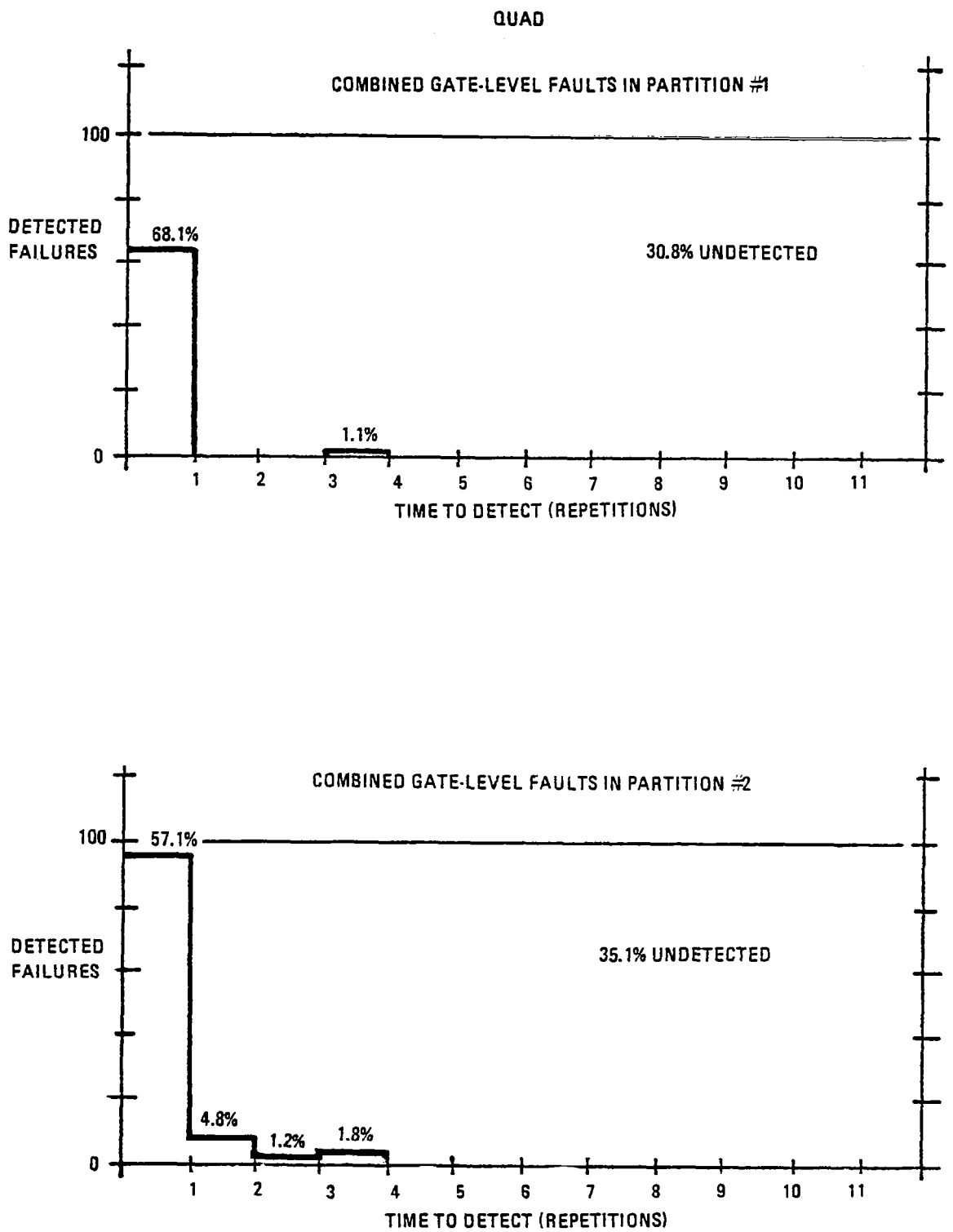
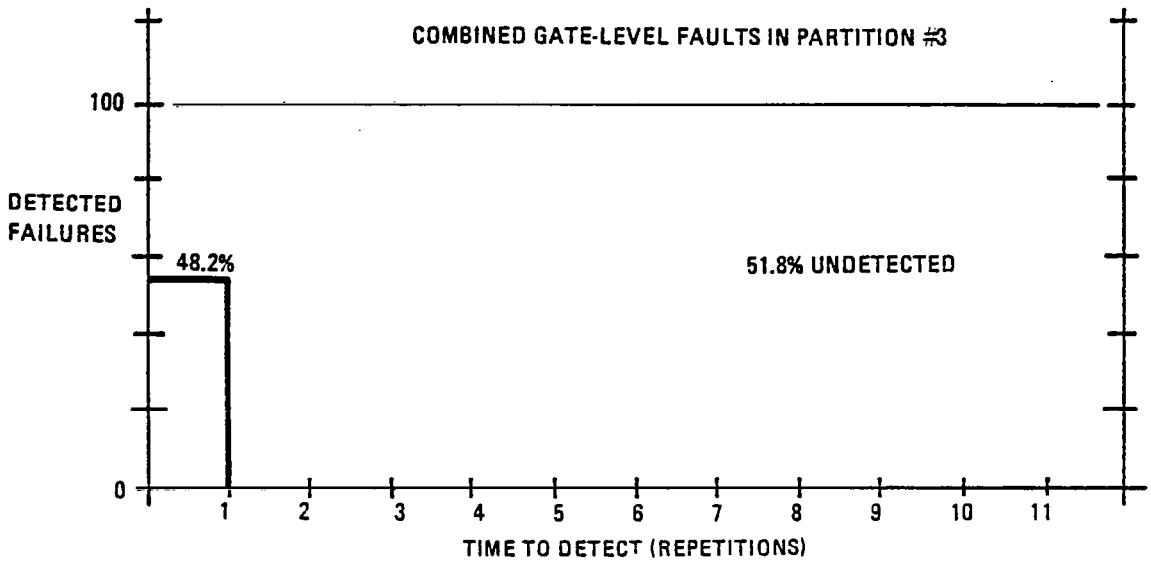

QUAD
COMBINED GATE-LEVEL FAULTS IN PARTITION #1
100
DETECTED FAILURES
68.1%
30.8% UNDETECTED
1.1%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)
COMBINED GATE-LEVEL FAULTS IN PARTITION #2
100
57.1%
DETECTED FAILURES
35.1% UNDETECTED
4.8%
1.2%
1.8%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 5c


COMBINED GATE-LEVEL FAULTS IN PARTITION #3
100
DETECTED FAILURES
48.2%
51.8% UNDETECTED
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

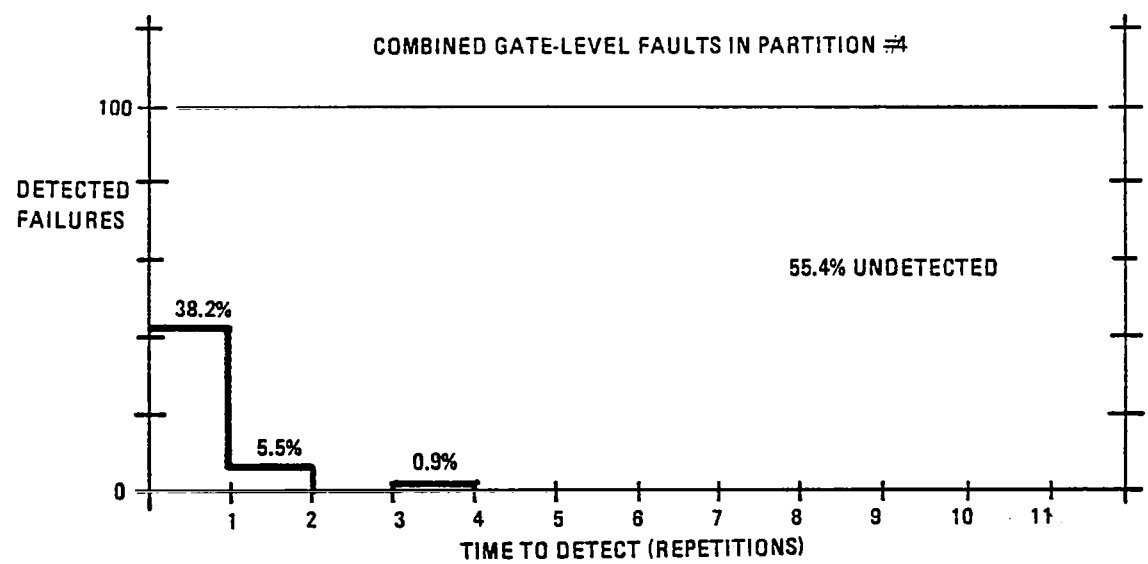
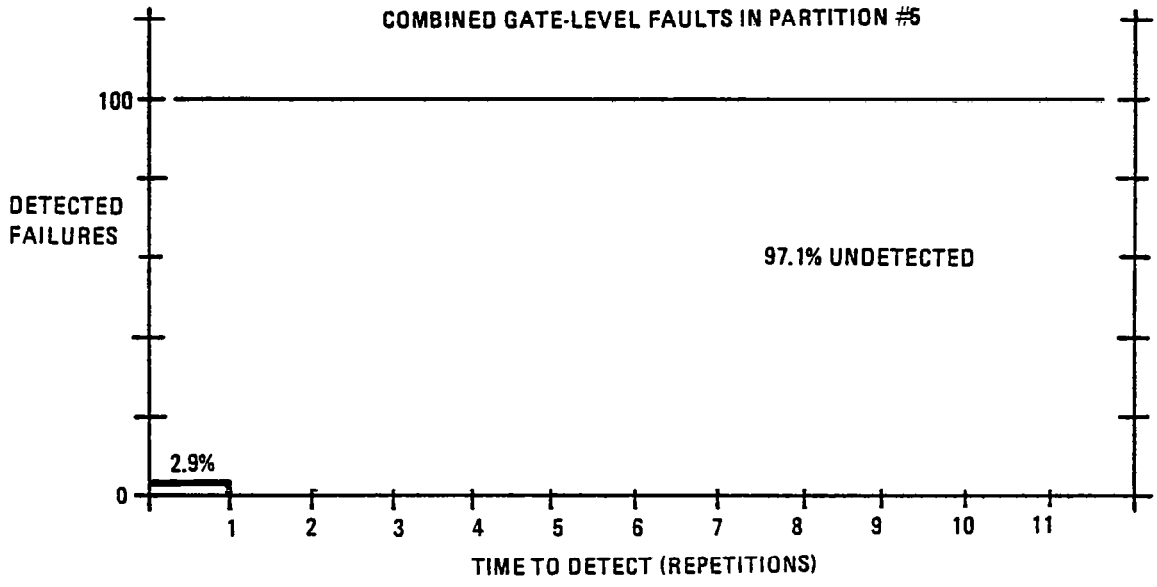

COMBINED GATE-LEVEL FAULTS IN PARTITION #4
100
DETECTED FAILURES
55.4% UNDETECTED
38.2%
5.5%
0.9%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 5d

QUAD



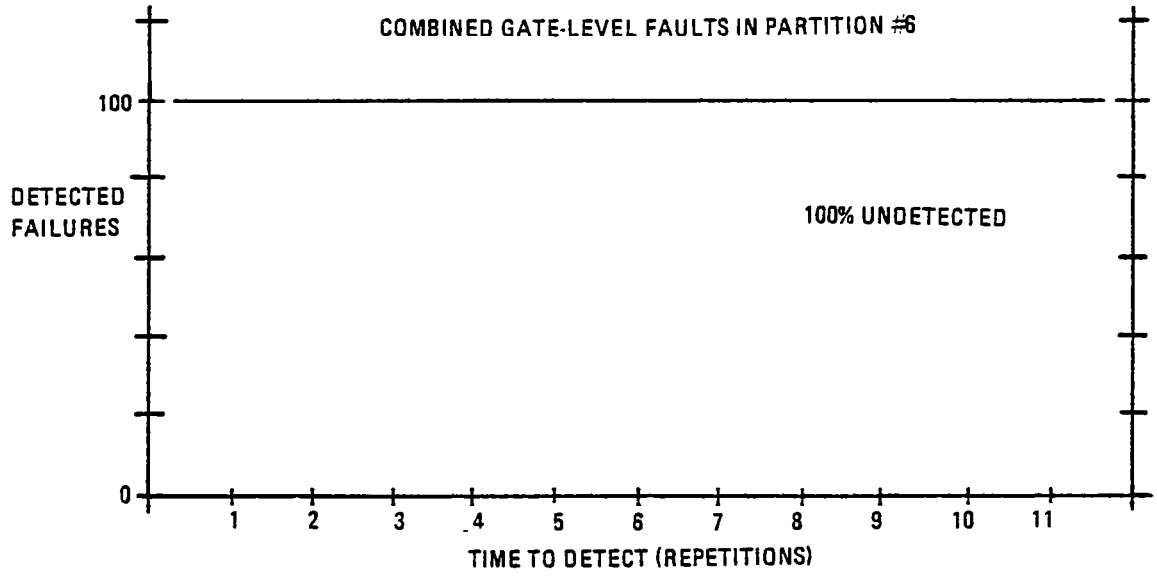


FIGURE 5e

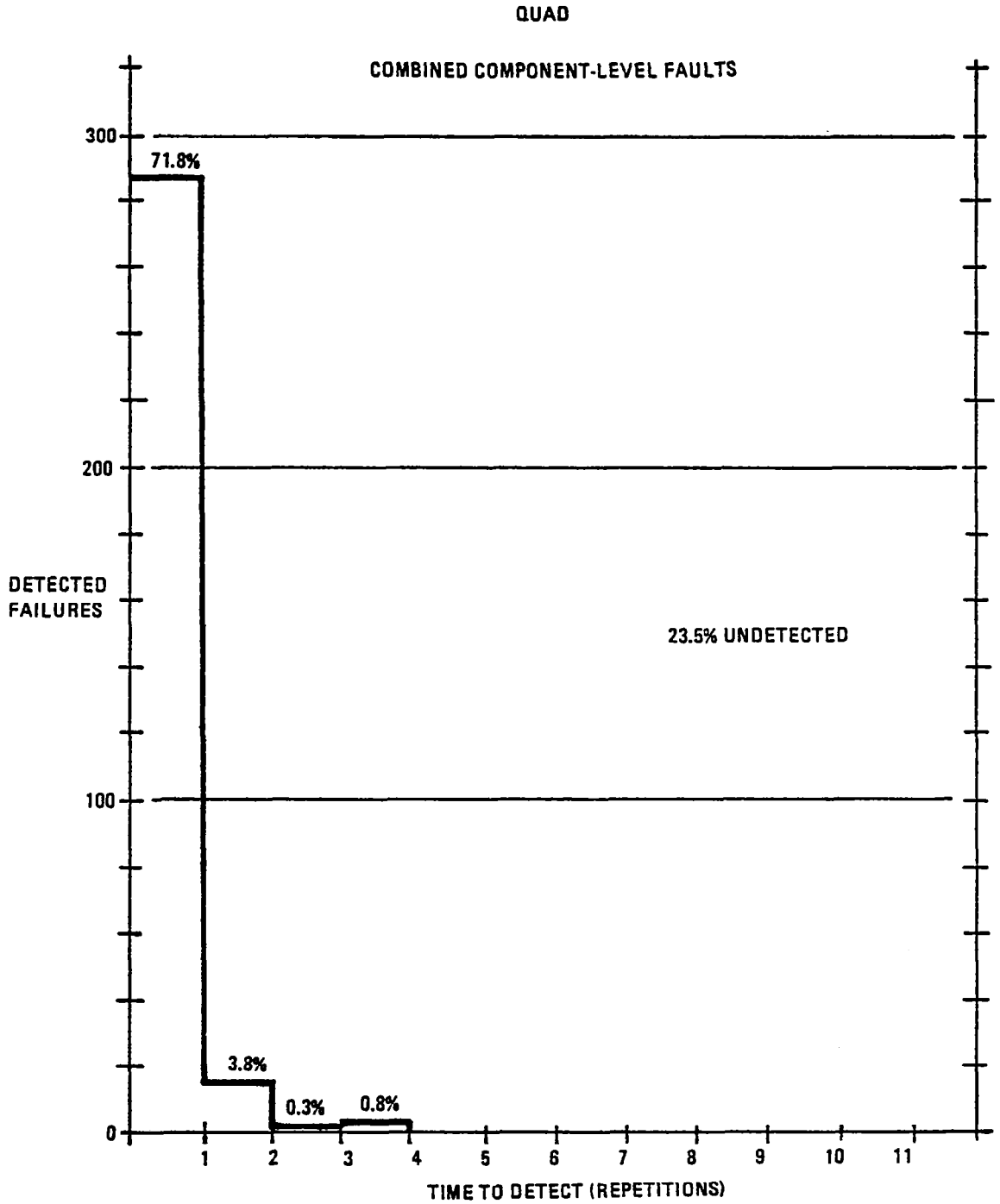
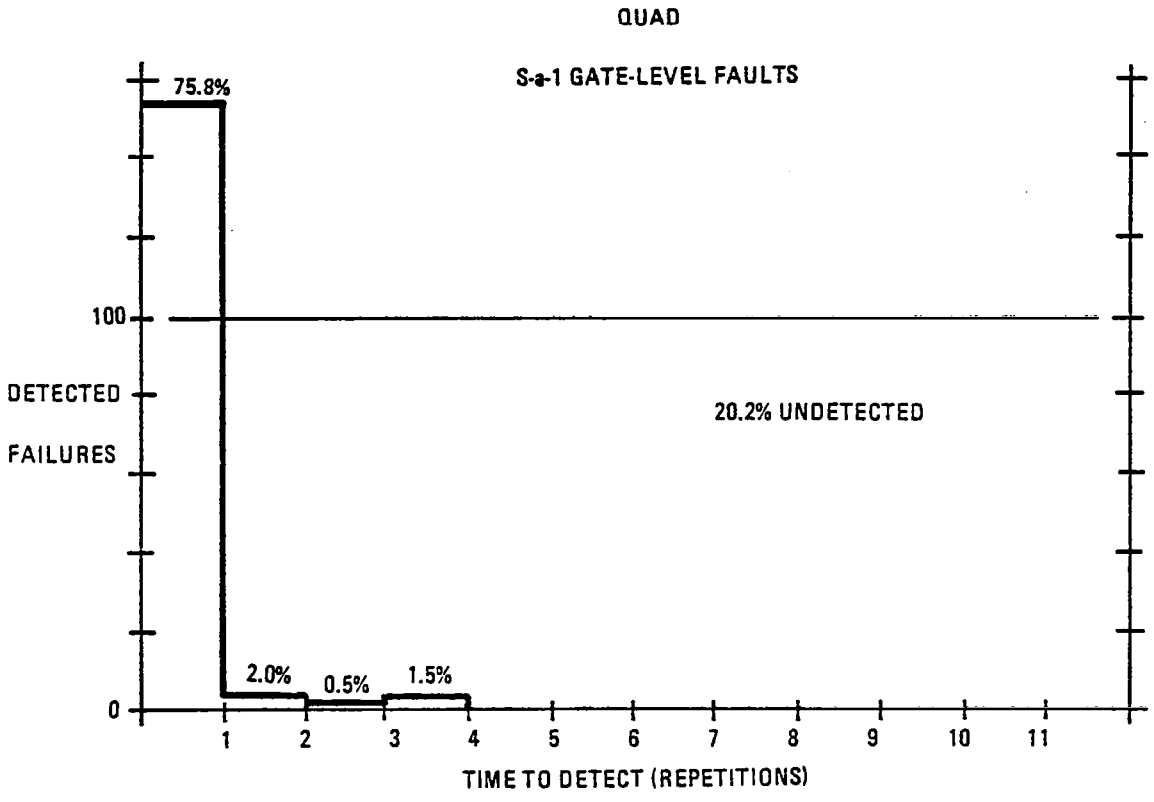

QUAD
COMBINED COMPONENT-LEVEL FAULTS
300
200
100
0
DETECTED FAILURES
71.8%
3.8%
0.3%
0.8%
23.5% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 5f


QUAD
S-a-1 GATE-LEVEL FAULTS
75.8%
100
DETECTED
FAILURES
20.2% UNDETECTED
2.0%
0.5%
1.5%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

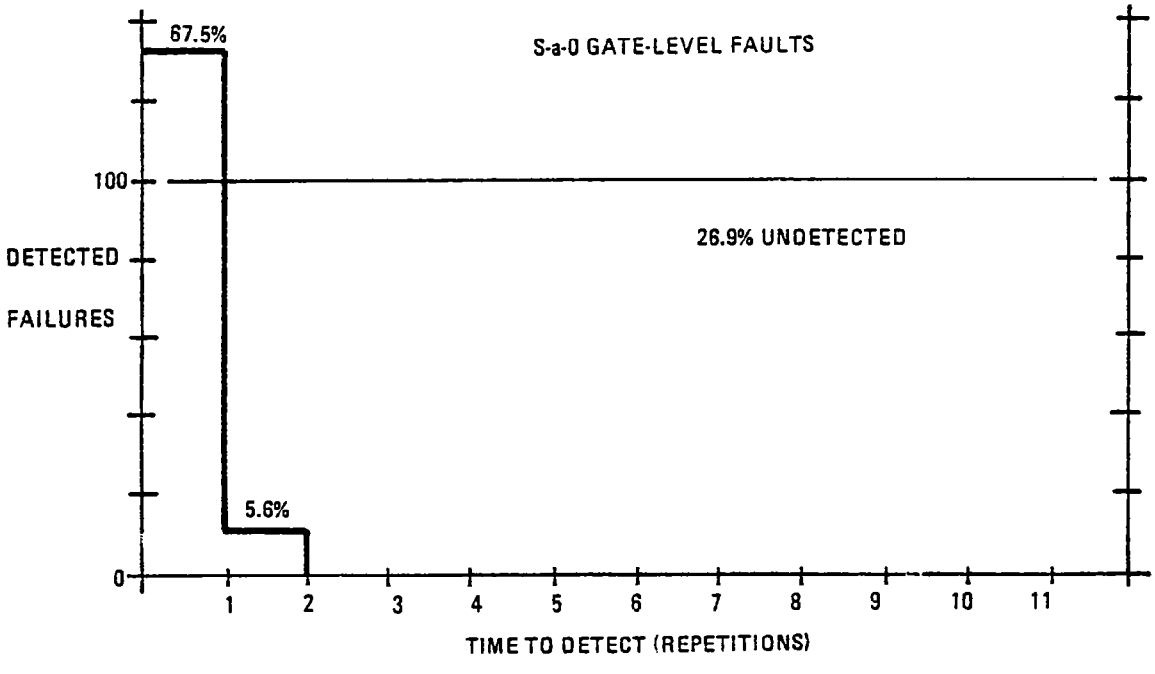
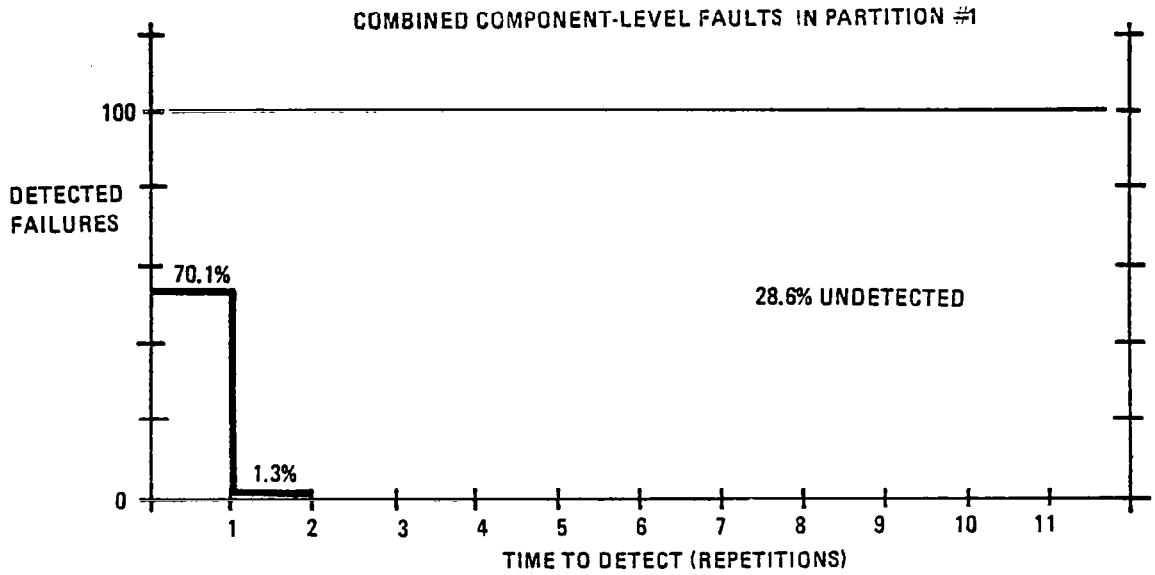

67.5%
S-a-0 GATE-LEVEL FAULTS
100
DETECTED
FAILURES
26.9% UNDETECTED
5.6%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 5g

QUAD



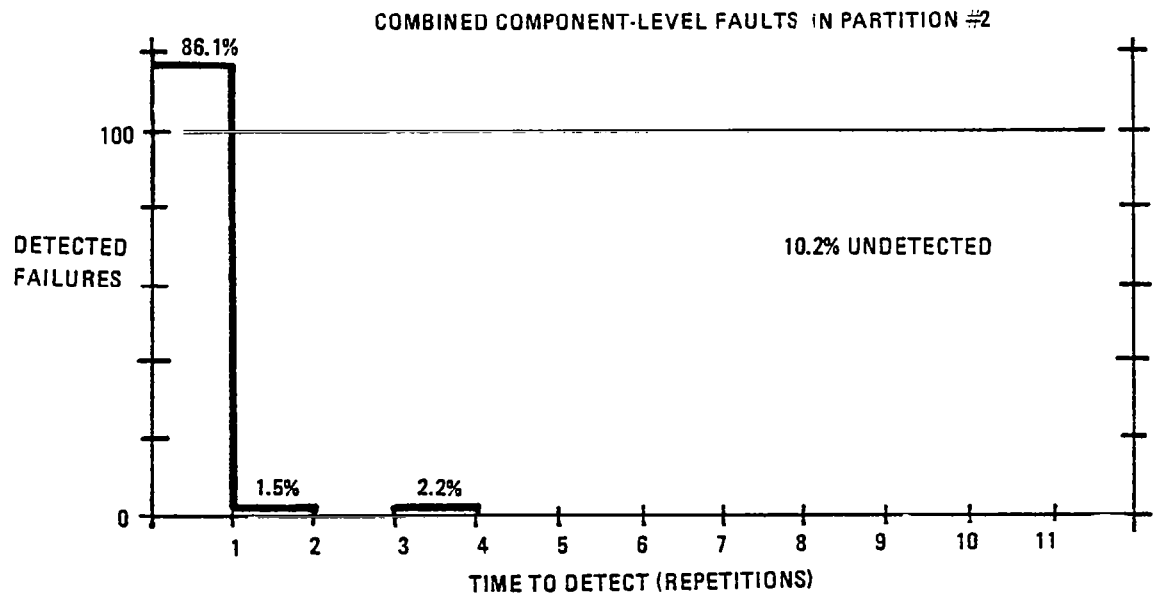
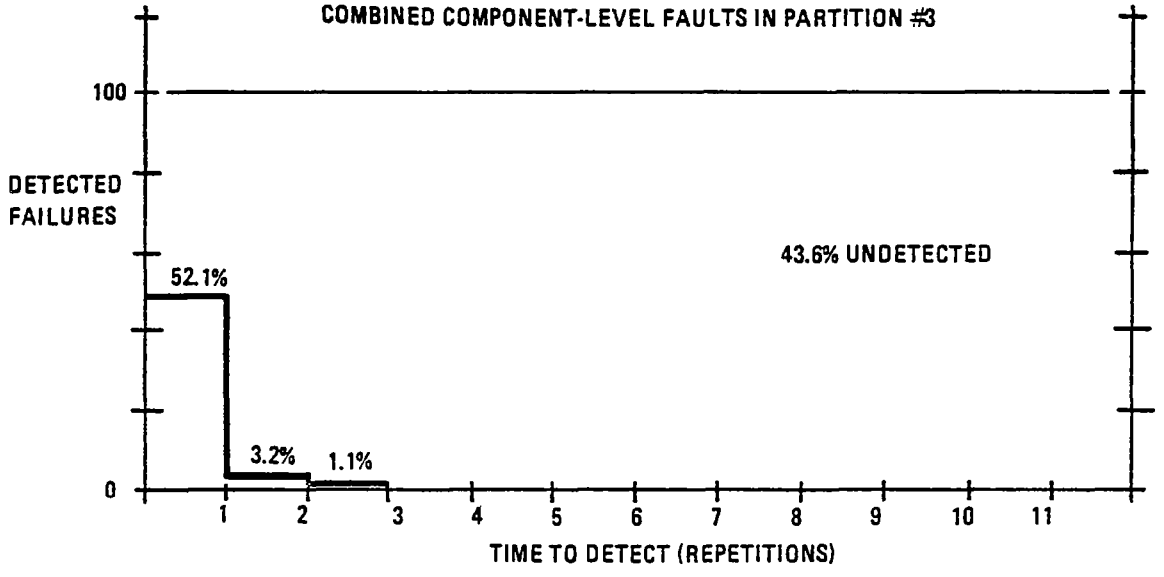


FIGURE 5h


COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #3
100
DETECTED FAILURES
0
52.1%
3.2%
1.1%
43.6% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

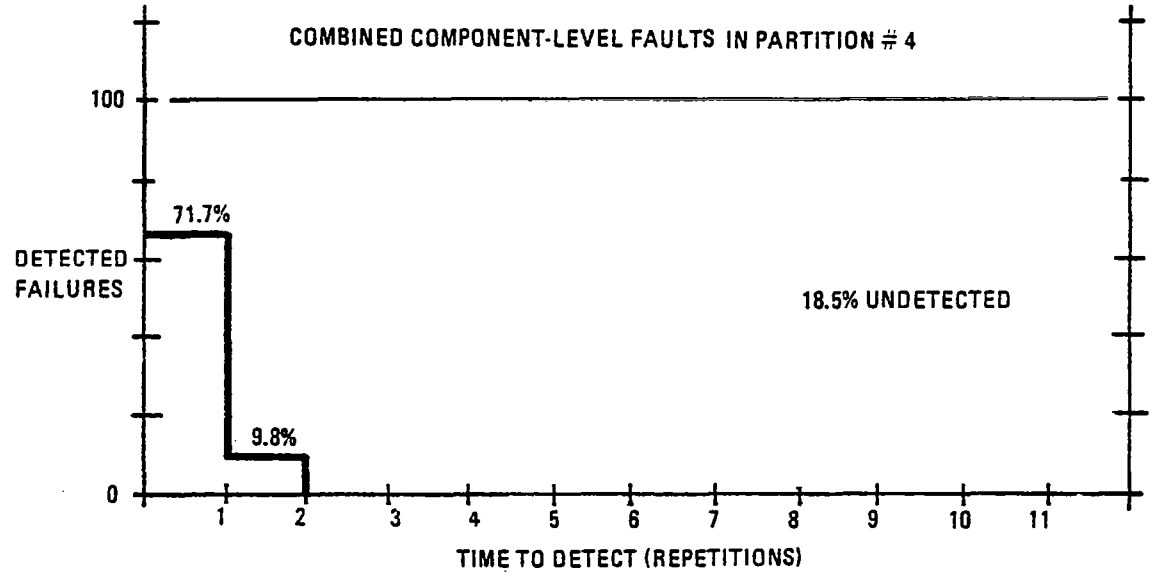

COMBINED COMPONENT-LEVEL FAULTS IN PARTITION # 4
100
DETECTED FAILURES
0
71.7%
9.8%
18.5% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 5i

SERCOM LATENCY DATA

GATE-LEVEL FAULTS

PARTITION	DETECTED FAULTS		FAULTS INJECTED	
	m_1	n_1	m	n
P1	60	45	82	76
P2	58	76	147	127
P3	39	43	87	98
P4	25	46	98	108
P5	1	2	75	75
P6	0	0	14	13
TOTAL	183	212	503	497

COMPONENT-LEVEL FAULTS

PARTITION	DETECTED FAULTS		FAULTS INJECTED	
	m_1	n_1	m	n
P1	27	32	38	39
P2	30	61	58	79
P3	26	24	52	42
P4	29	30	49	43
P5				
P6				
TOTAL	112	147	197	203

TABLE 9

m_1 = detected S-a-0 faults, ith cell

n_1 = detected S-a-1 faults, ith cell

LINCON LATENCY DATA

GATE-LEVEL FAULTS

PARTITION	DETECTED FAULTS		FAULTS INJECTED	
	m_1	n_1	m	n
P1	37	33	49	45
P2	65	69	90	78
P3	25	34	46	66
P4	20	23	53	57
P5	1	3	52	52
P6	0	0	6	6
TOTAL	148	162	296	304

COMPONENT-LEVEL FAULTS

PARTITION	DETECTED FAULTS		FAULTS INJECTED	
	m_1	n_1	m	n
P1	29	32	38	39
P2	52	70	58	79
P3	31	24	52	42
P4	34	34	49	43
P5				
P6				
TOTAL	146	160	197	203

TABLE 10

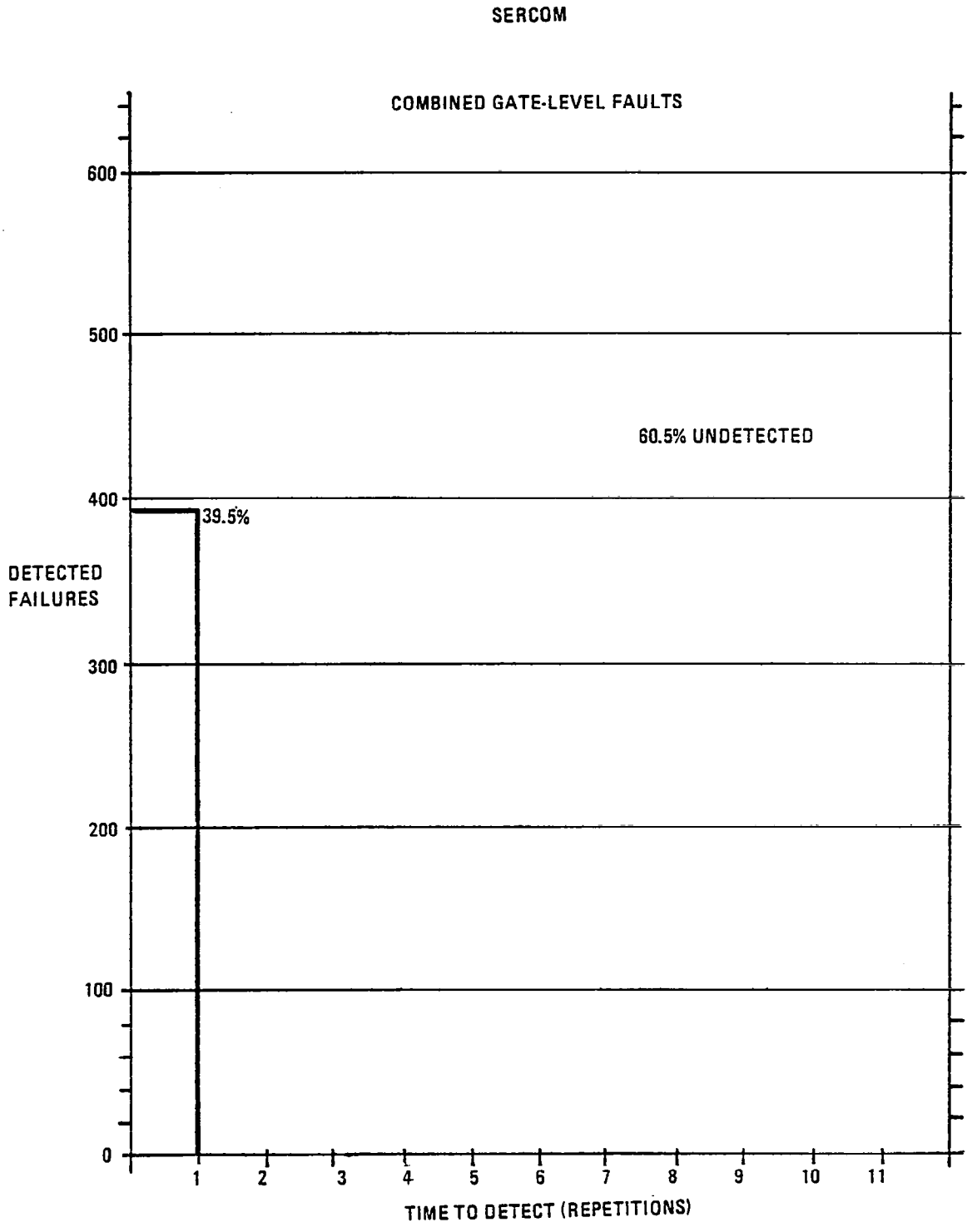

SERCOM
COMBINED GATE-LEVEL FAULTS
600
500
60.5% UNDETECTED
400
39.5%
DETECTED
FAILURES
300
200
100
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 6a

SERCOM

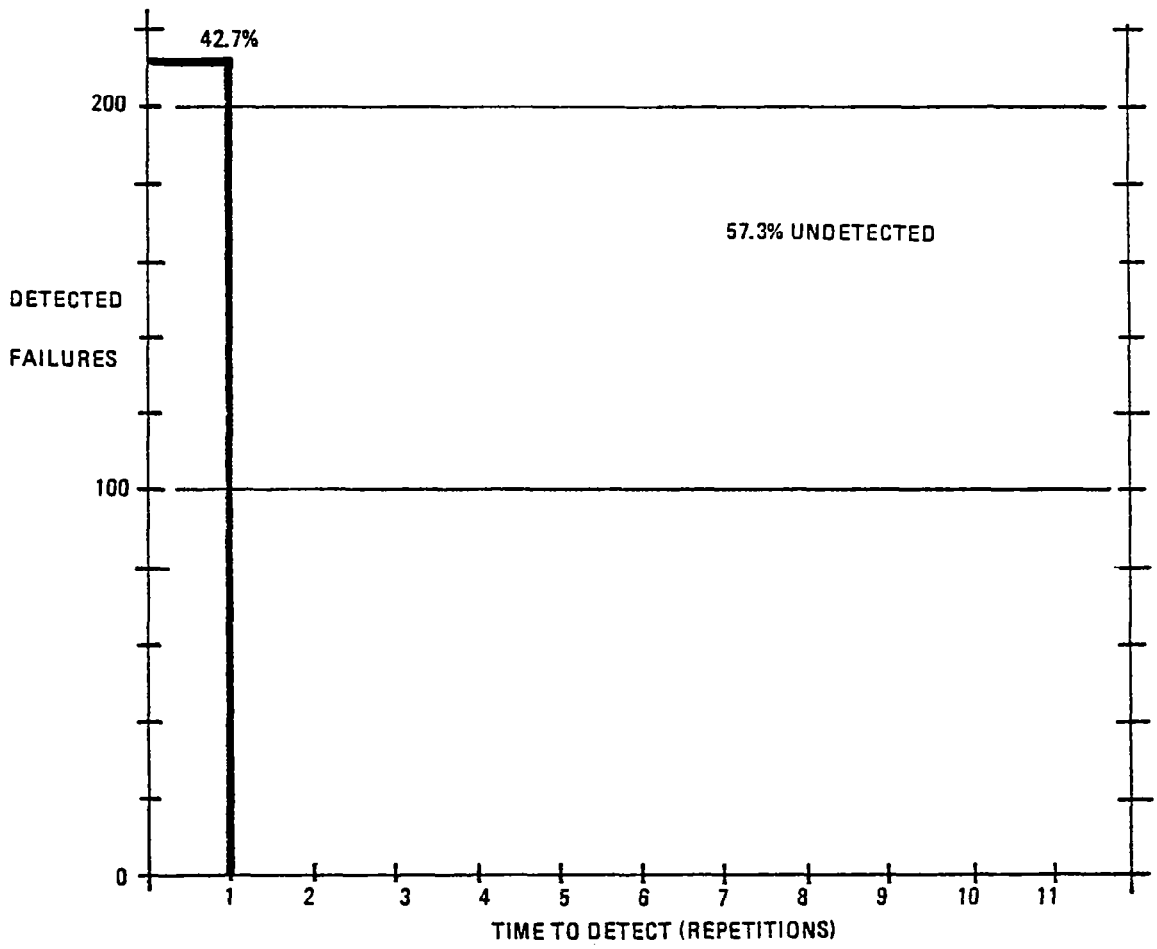


FIGURE 6b

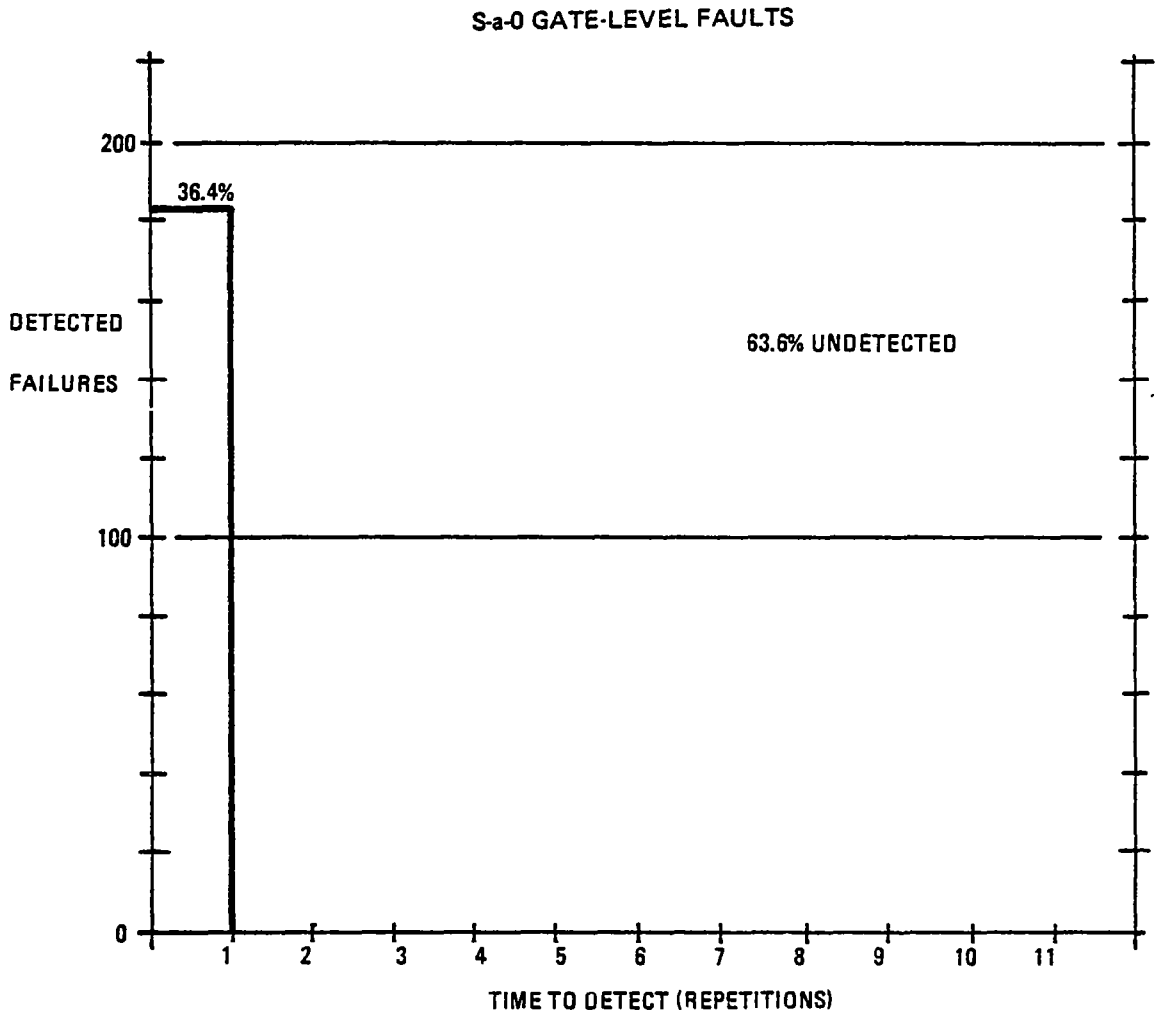
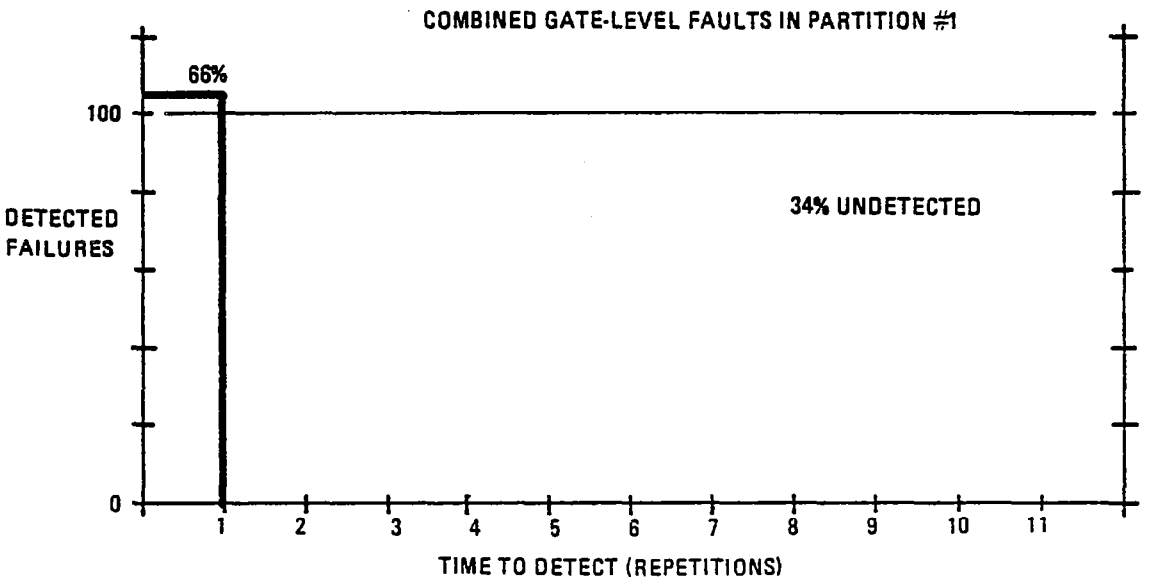

S-a-0 GATE-LEVEL FAULTS
200
100
0
DETECTED
FAILURES
36.4%
63.6% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 6c


COMBINED GATE-LEVEL FAULTS IN PARTITION #1
66%
100
DETECTED FAILURES
34% UNDETECTED
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

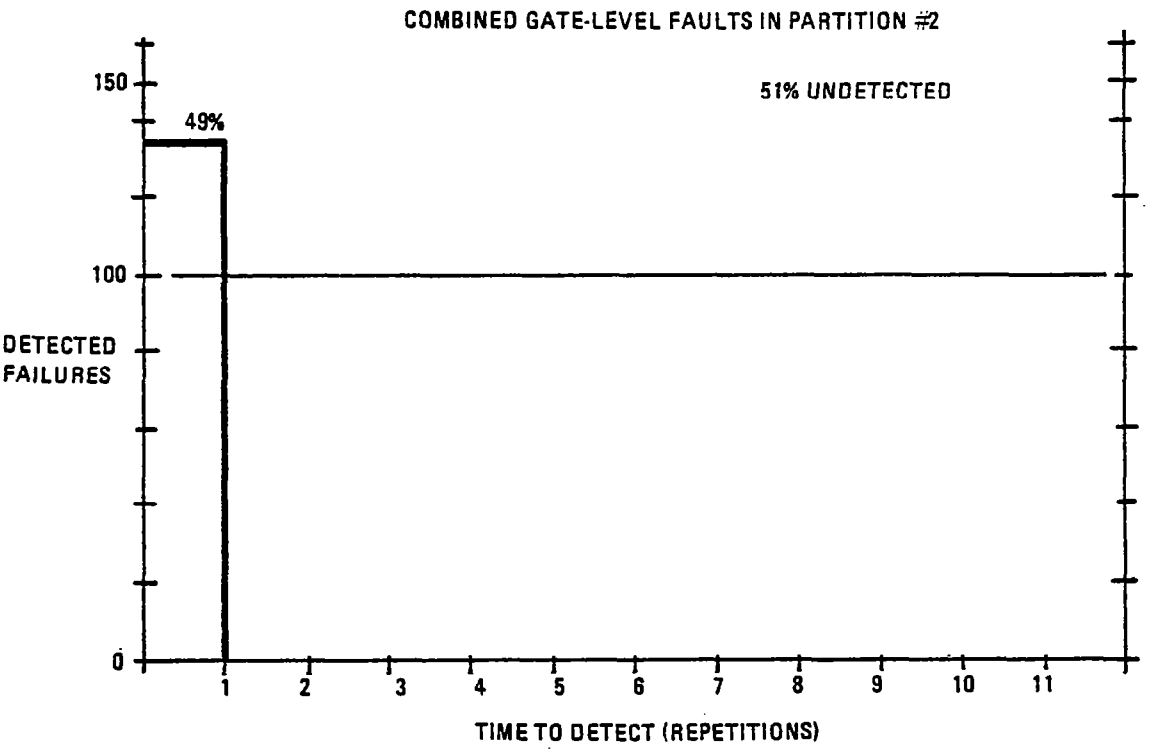
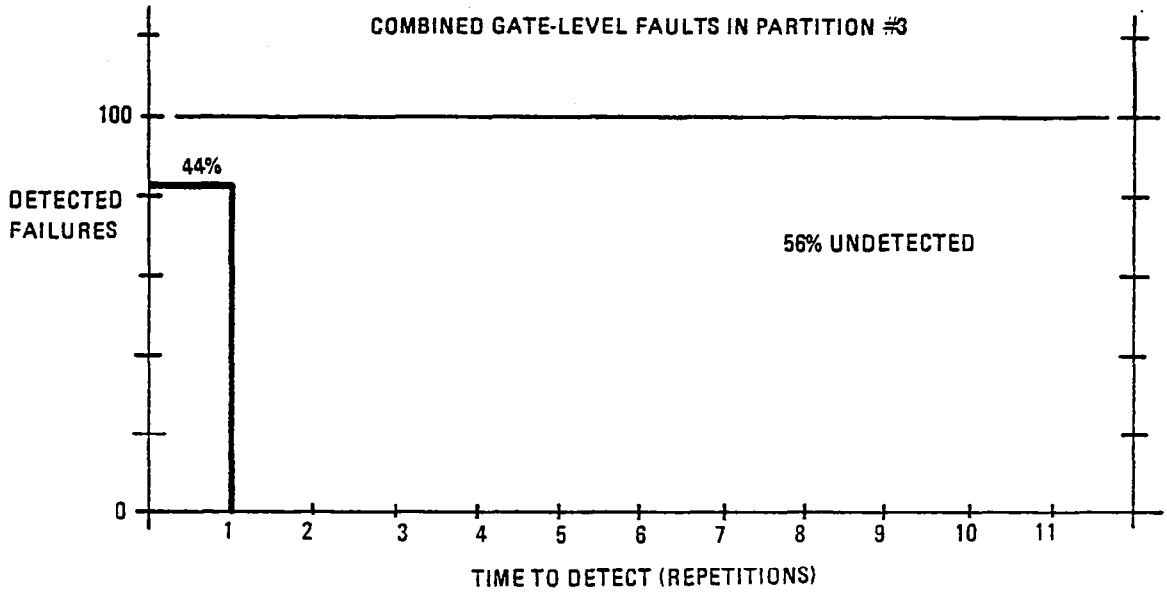

COMBINED GATE-LEVEL FAULTS IN PARTITION #2
51% UNDETECTED
150
49%
100
DETECTED FAILURES
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 6d

SERCOM



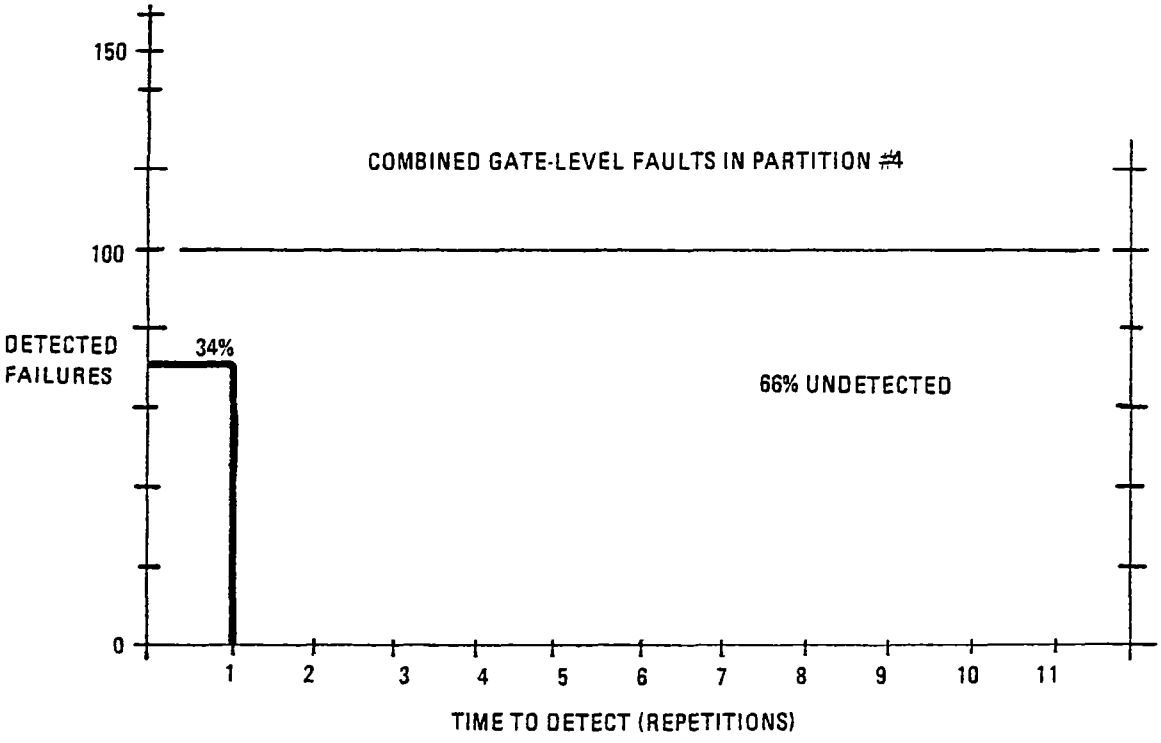
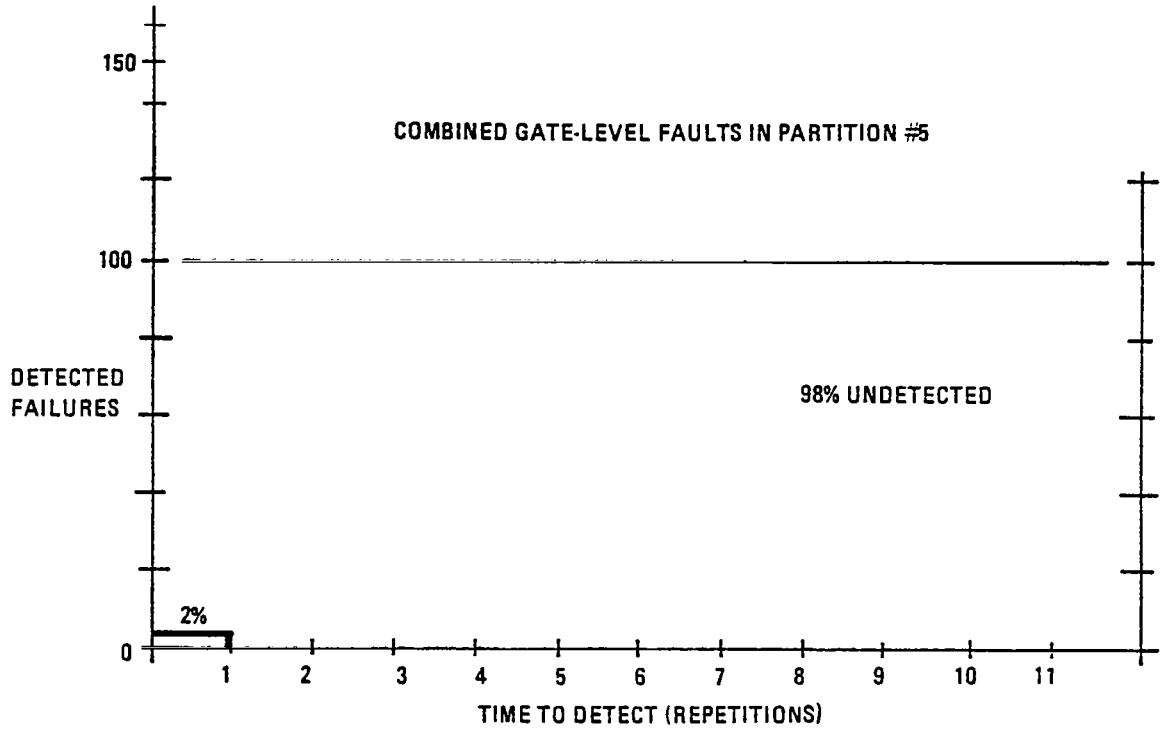


FIGURE 6e


150
100
0
DETECTED FAILURES
COMBINED GATE-LEVEL FAULTS IN PARTITION #5
98% UNDETECTED
2%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

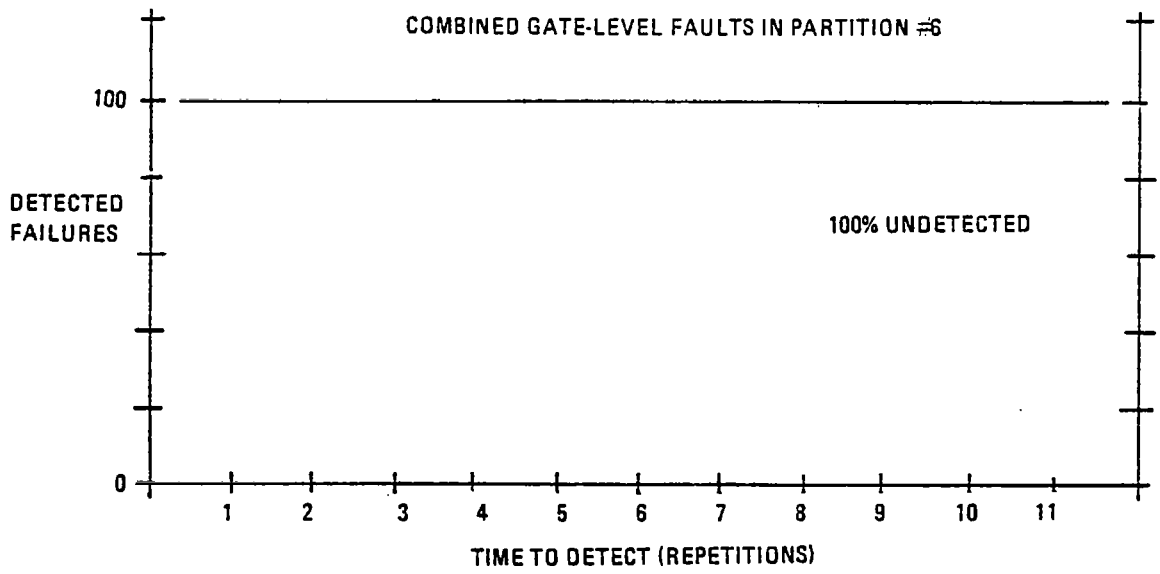

100
0
DETECTED FAILURES
COMBINED GATE-LEVEL FAULTS IN PARTITION #6
100% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 6f

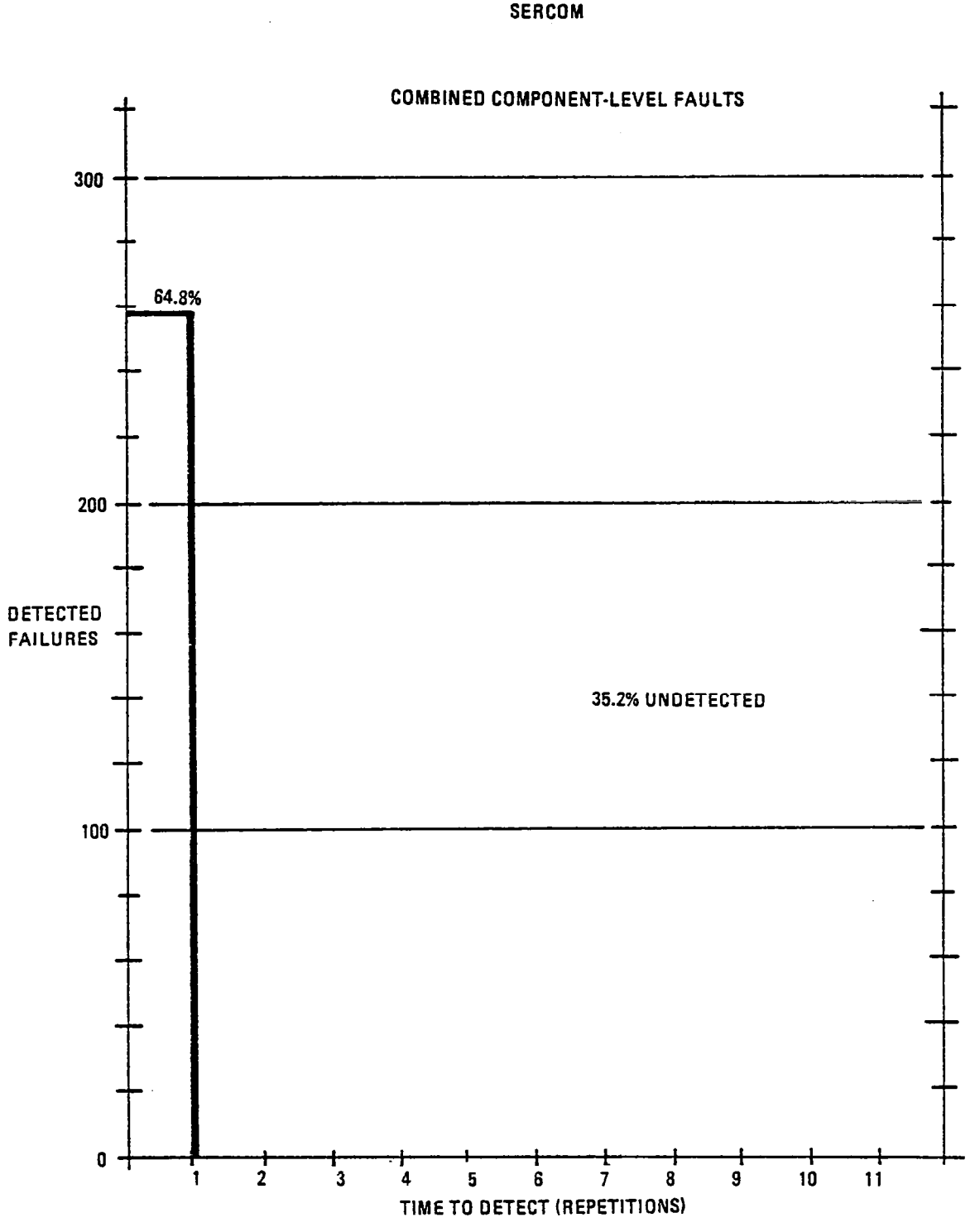
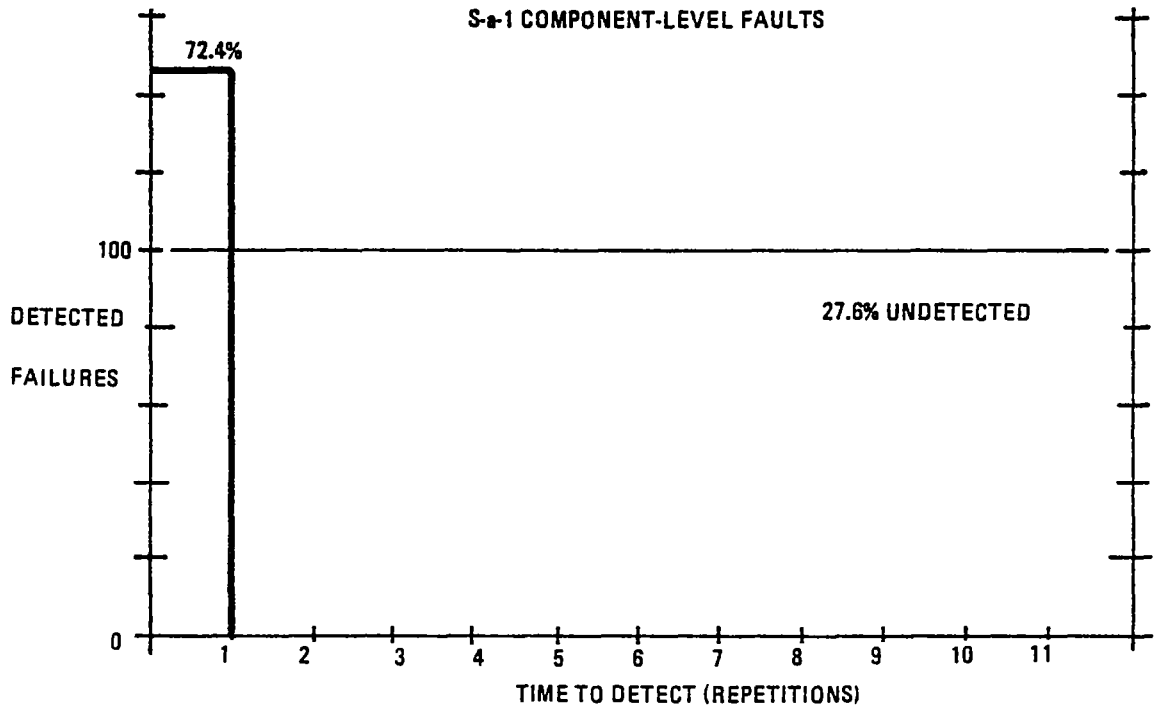

SERCOM
COMBINED COMPONENT-LEVEL FAULTS
300
200
100
0
64.8%
35.2% UNDETECTED
DETECTED FAILURES
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 6g

SERCOM



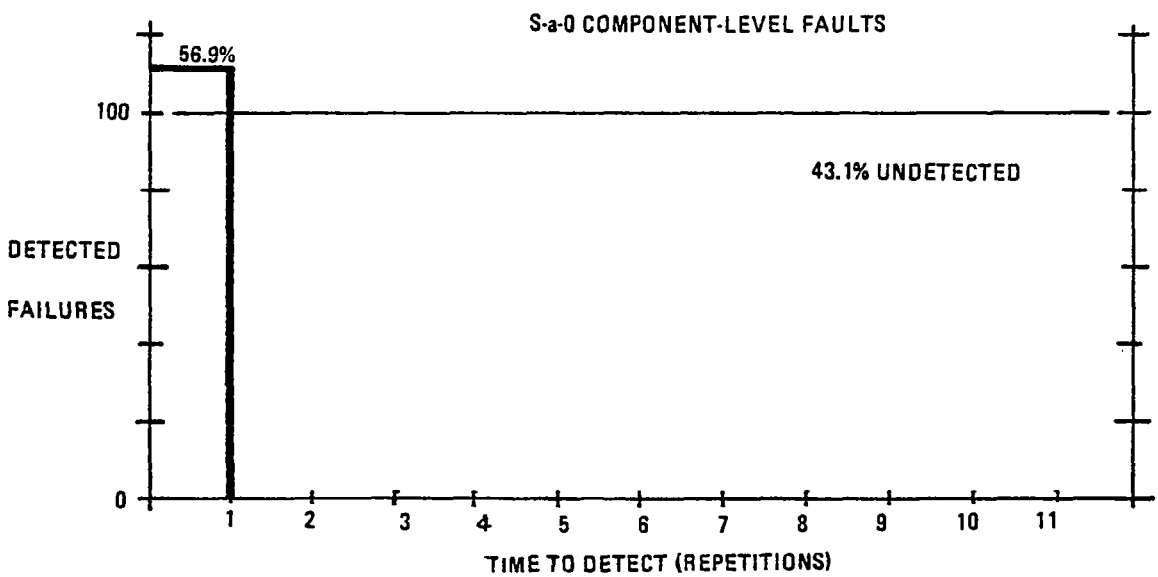
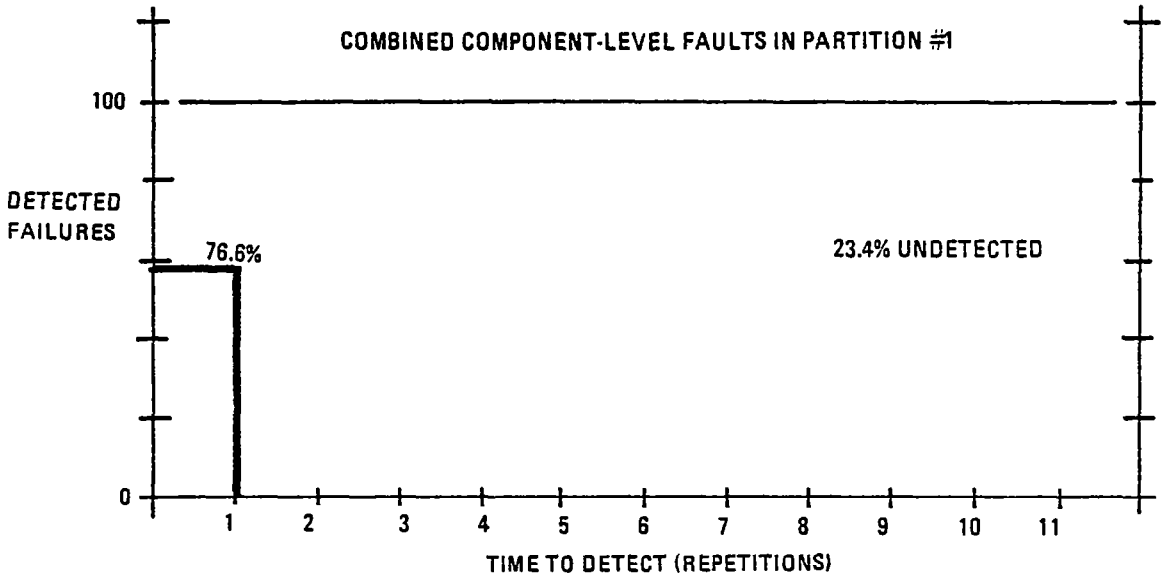


FIGURE 6h


COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #1
100
DETECTED FAILURES
76.6%
23.4% UNDETECTED
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

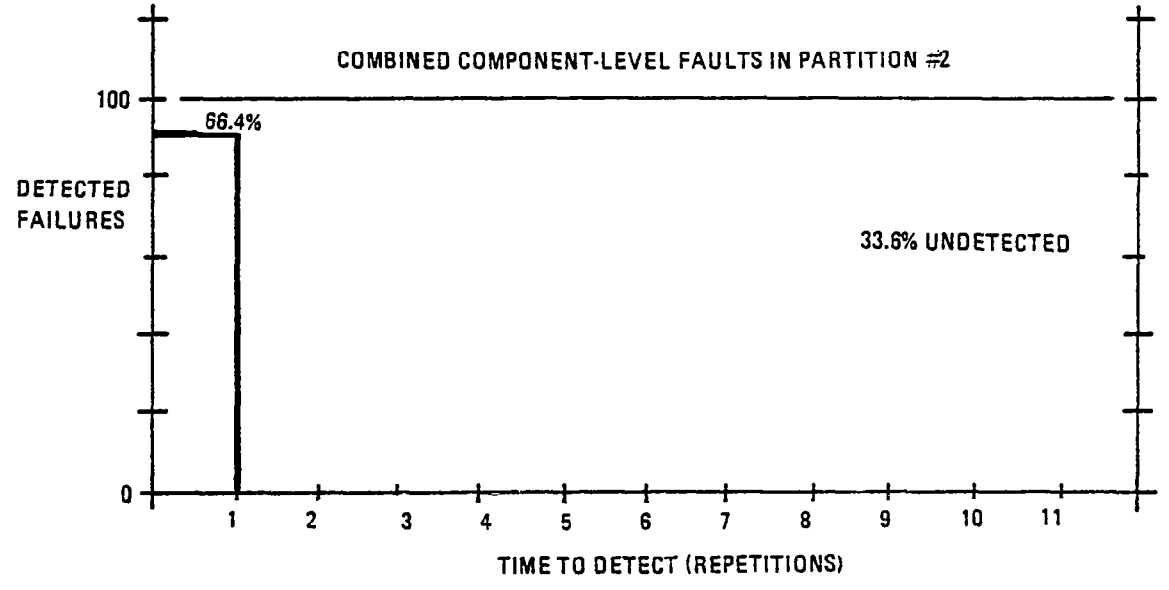
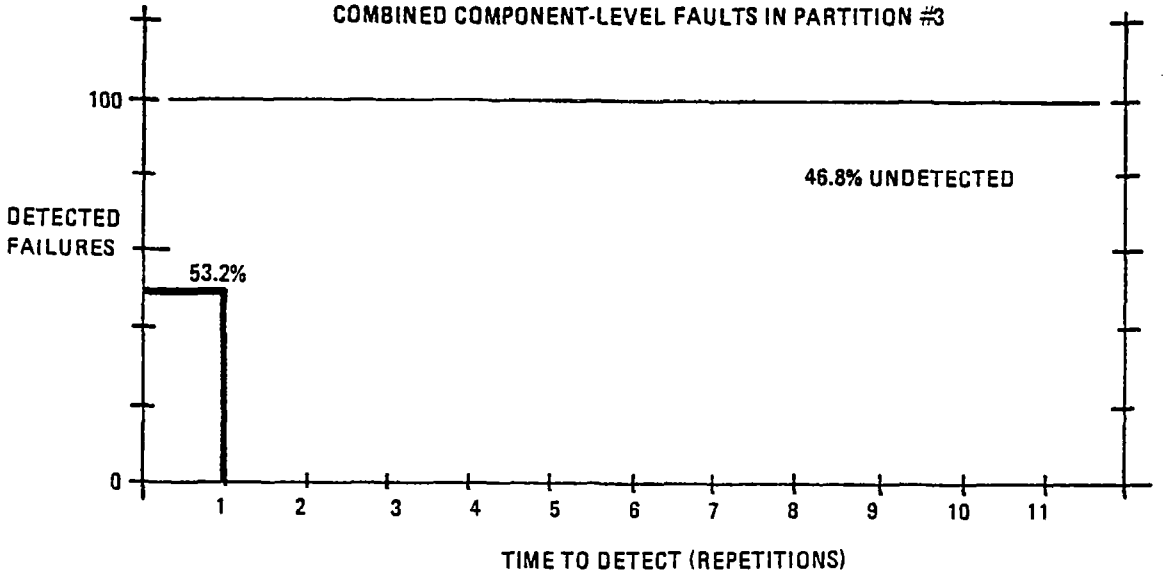

COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #2
100
66.4%
DETECTED FAILURES
33.6% UNDETECTED
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 6i


COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #3
100
DETECTED FAILURES
0
53.2%
46.8% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

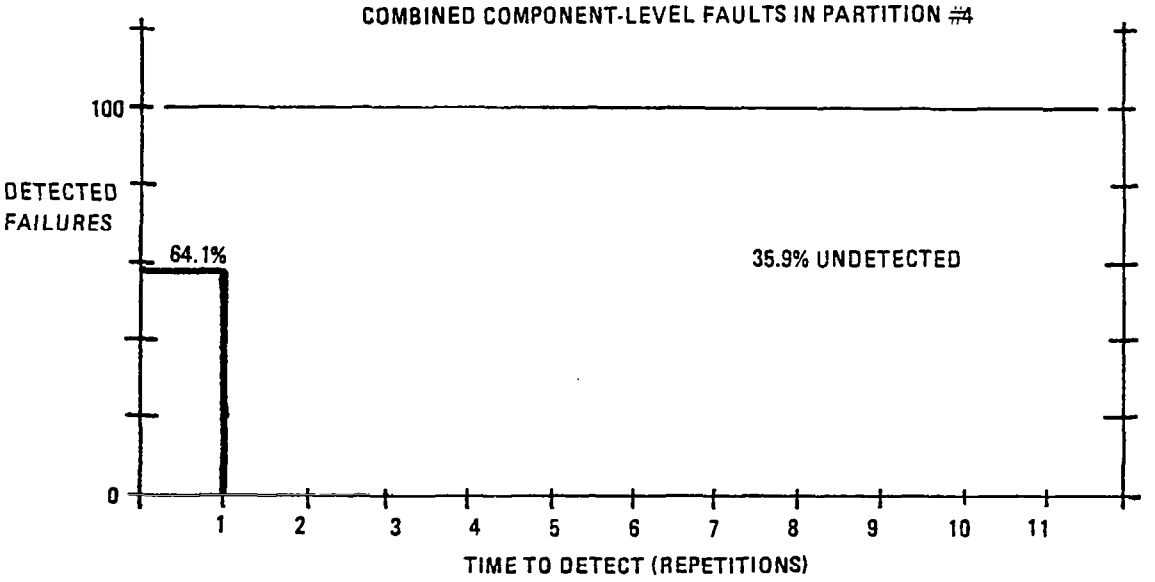

COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #4
100
DETECTED FAILURES
0
64.1%
35.9% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 6j

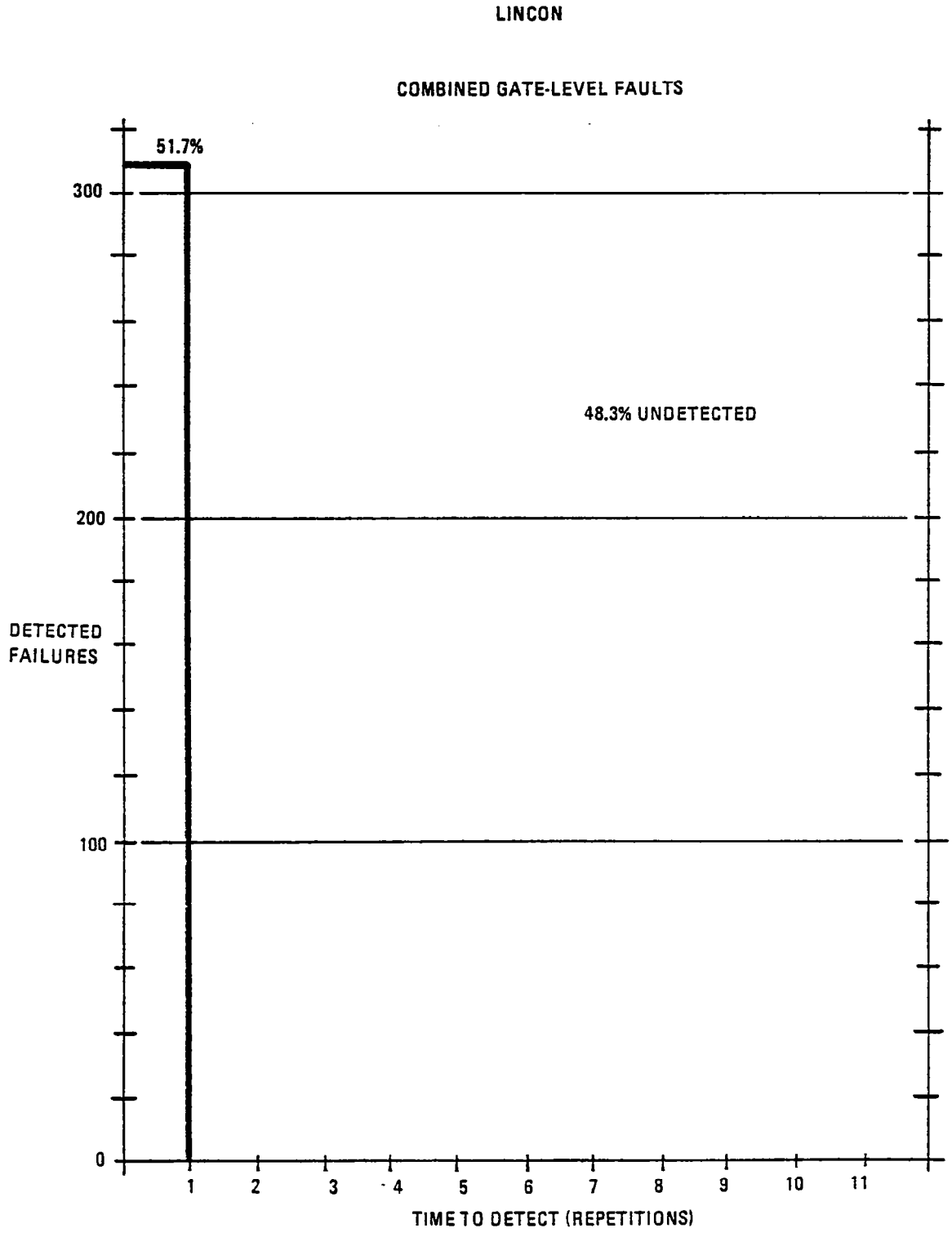

LINCON
COMBINED GATE-LEVEL FAULTS
51.7%
48.3% UNDETECTED
300
200
100
0
DETECTED FAILURES
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 7a

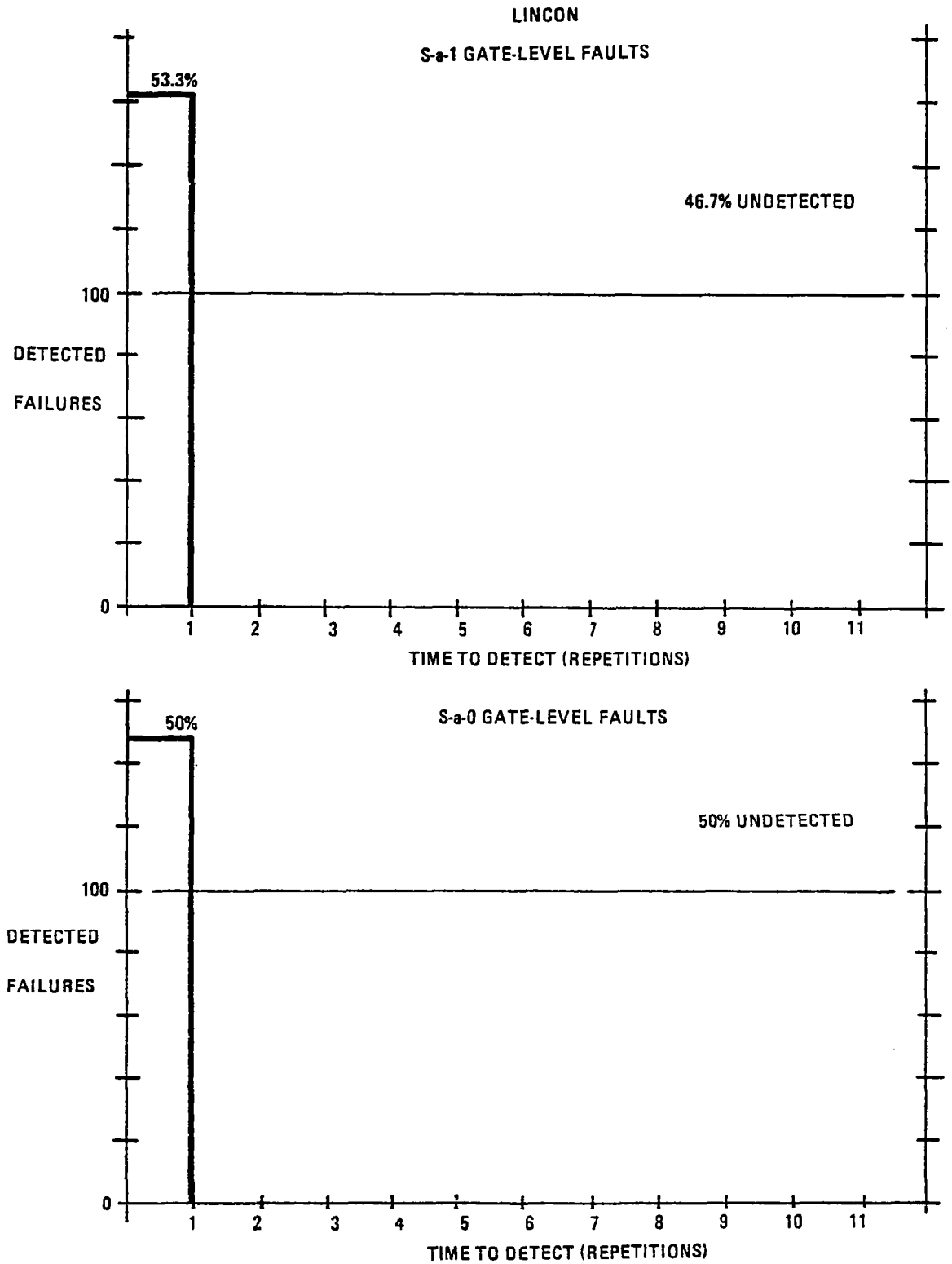

LINCON
S-a-1 GATE-LEVEL FAULTS
53.3%
46.7% UNDETECTED
100
DETECTED
FAILURES
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)
S-a-0 GATE-LEVEL FAULTS
50%
50% UNDETECTED
100
DETECTED
FAILURES
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 7b

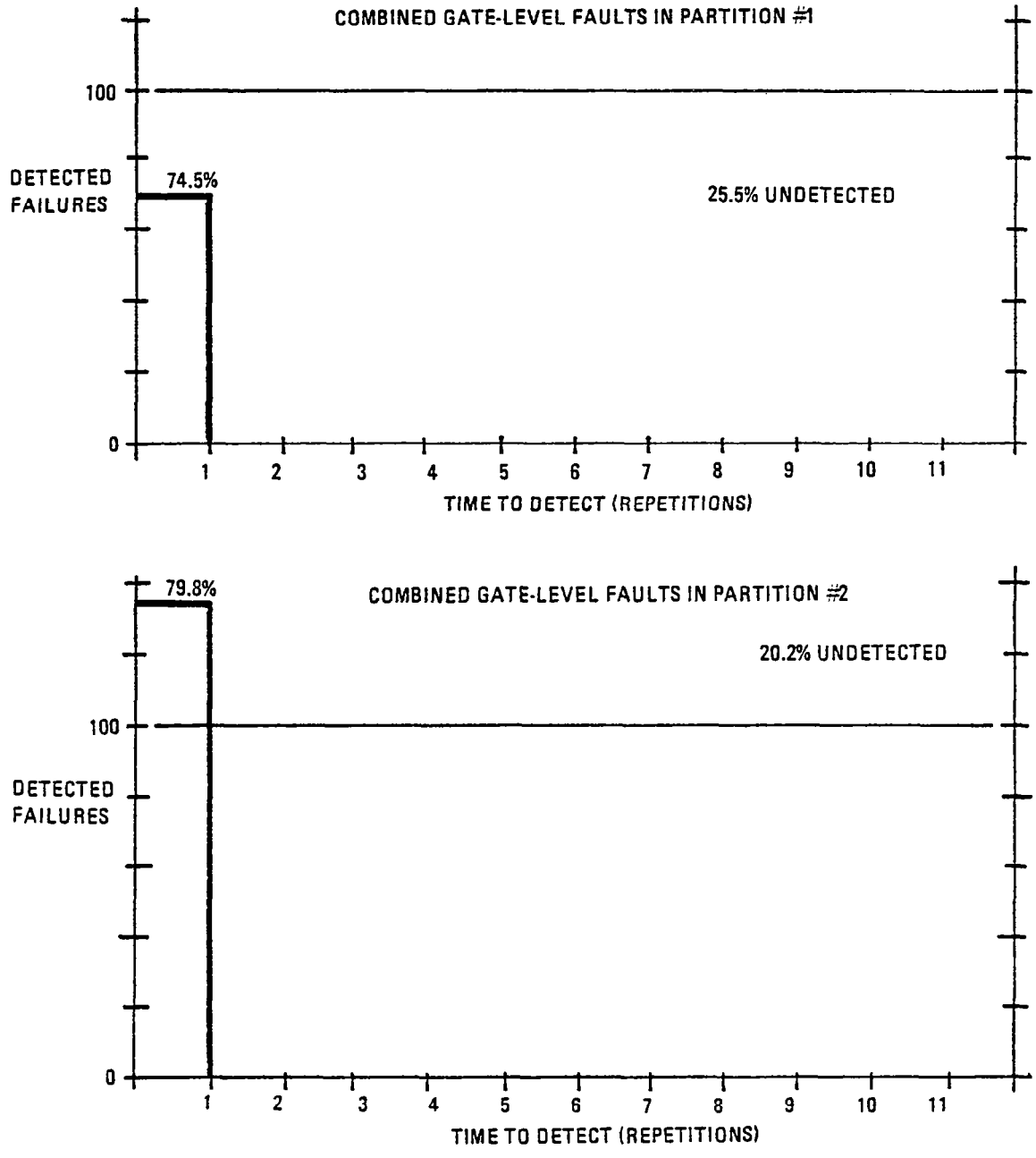

COMBINED GATE-LEVEL FAULTS IN PARTITION #1
100
DETECTED FAILURES
74.5%
25.5% UNDETECTED
0
1 2 3 4 5 6 7 8 9 10 11
TIME TO DETECT (REPETITIONS)
79.8%
COMBINED GATE-LEVEL FAULTS IN PARTITION #2
20.2% UNDETECTED
100
DETECTED FAILURES
0
1 2 3 4 5 6 7 8 9 10 11
TIME TO DETECT (REPETITIONS)

FIGURE 7c

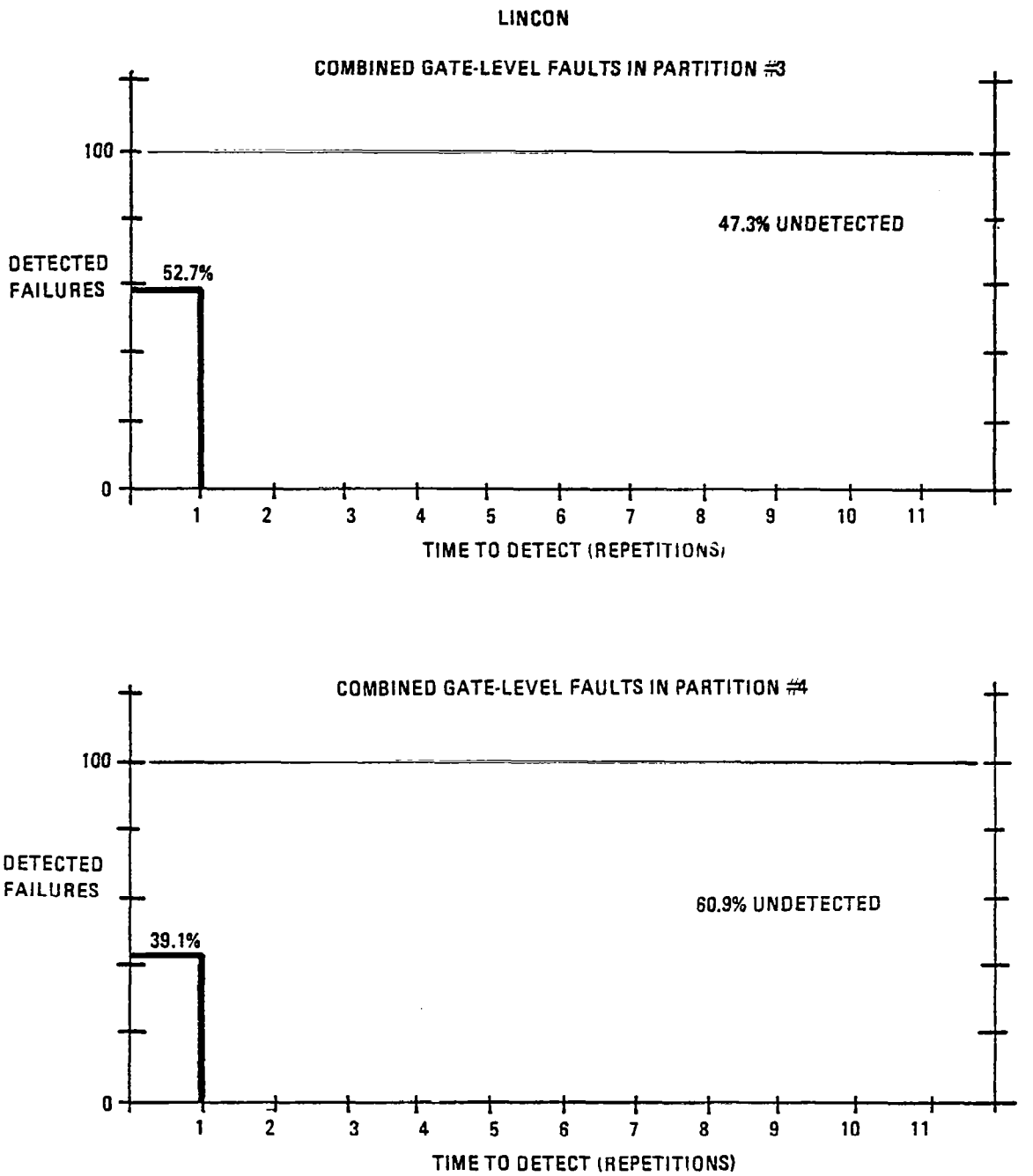

LINCON
COMBINED GATE-LEVEL FAULTS IN PARTITION #3
100
47.3% UNDETECTED
DETECTED FAILURES
52.7%
0
1 2 3 4 5 6 7 8 9 10 11
TIME TO DETECT (REPETITIONS)
COMBINED GATE-LEVEL FAULTS IN PARTITION #4
100
DETECTED FAILURES
60.9% UNDETECTED
39.1%
0
1 2 3 4 5 6 7 8 9 10 11
TIME TO DETECT (REPETITIONS)

FIGURE 7d

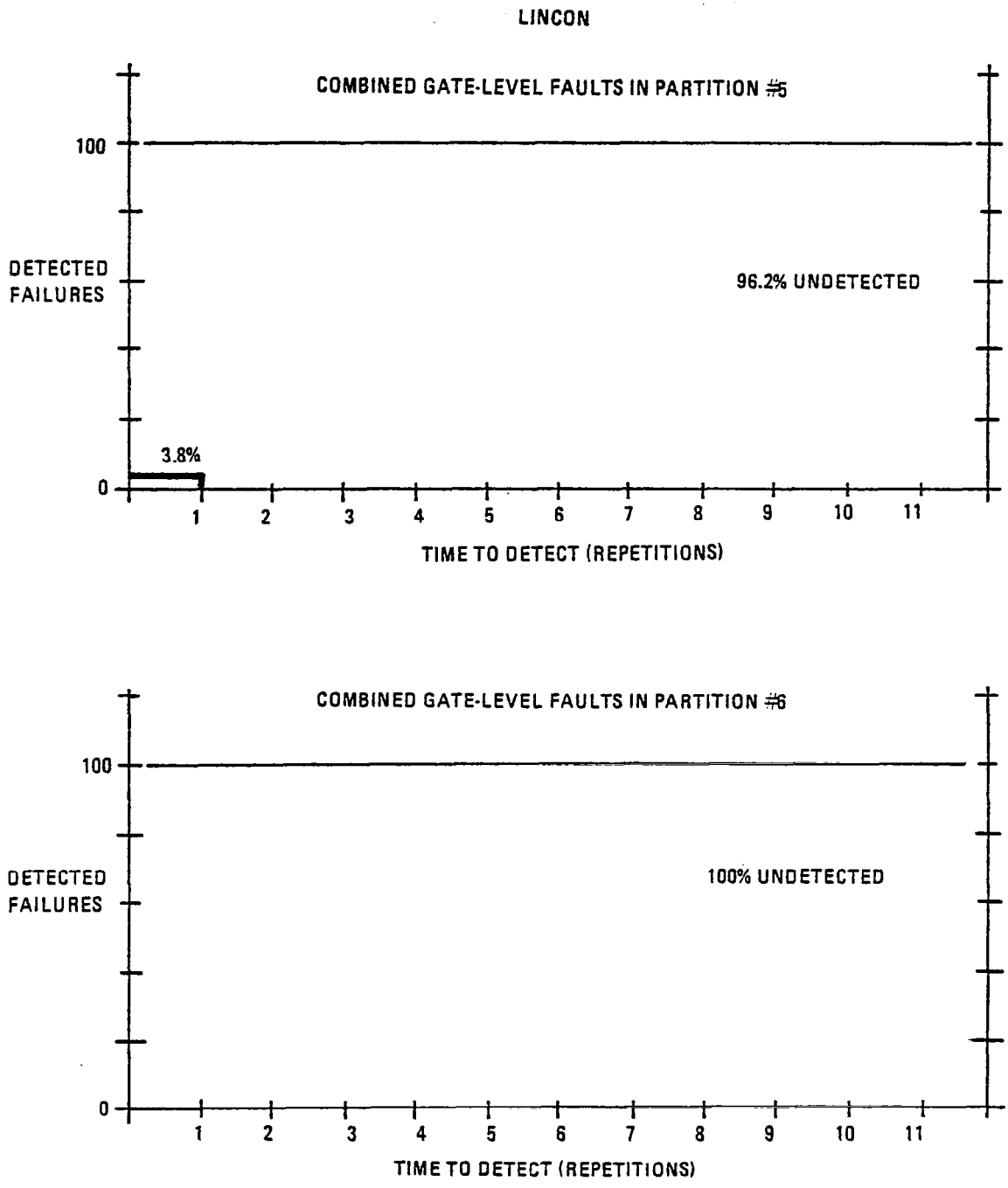

LINCON
COMBINED GATE-LEVEL FAULTS IN PARTITION #5
100
DETECTED FAILURES
96.2% UNDETECTED
3.8%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)
COMBINED GATE-LEVEL FAULTS IN PARTITION #6
100
DETECTED FAILURES
100% UNDETECTED
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 7e

LINCON

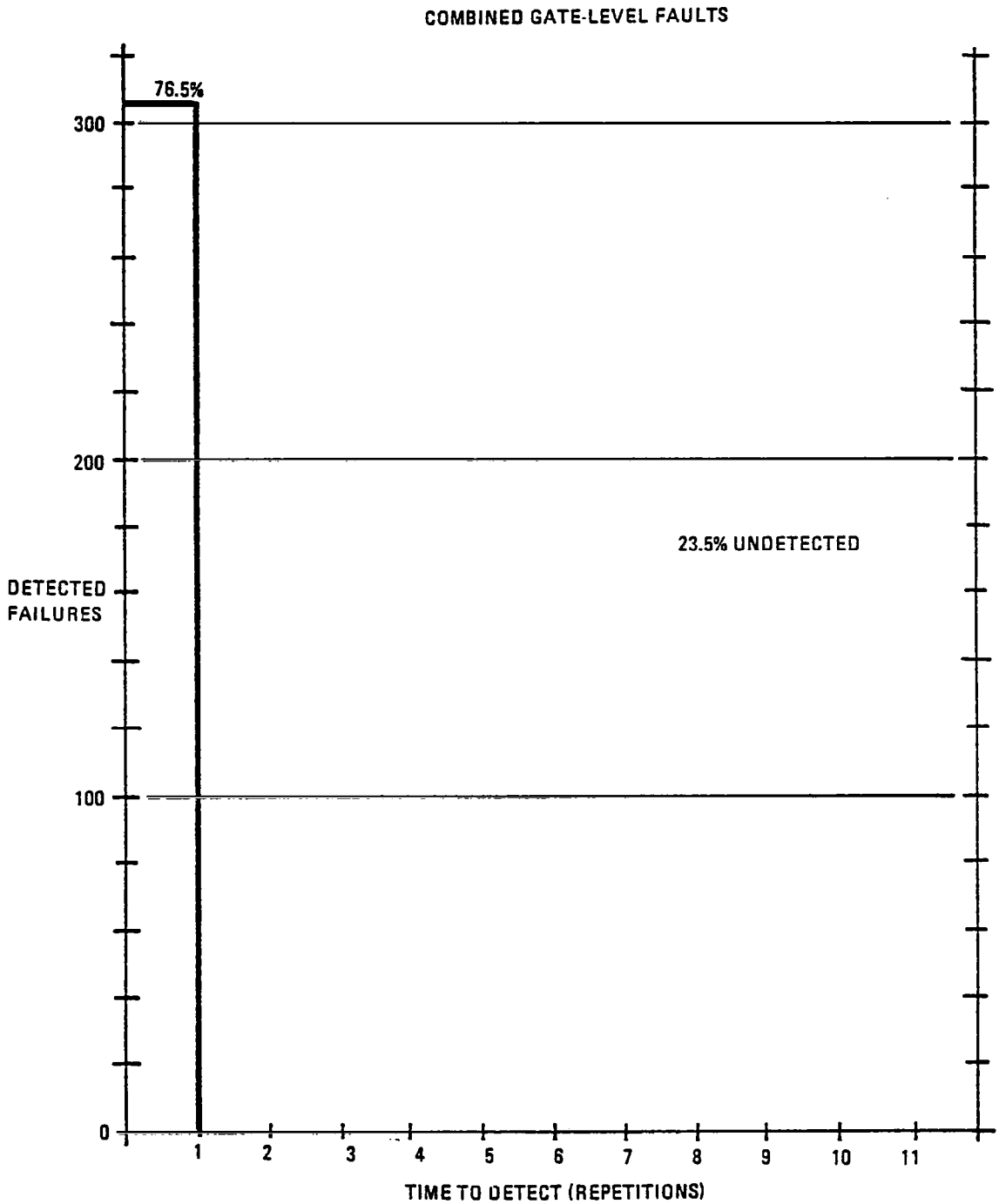
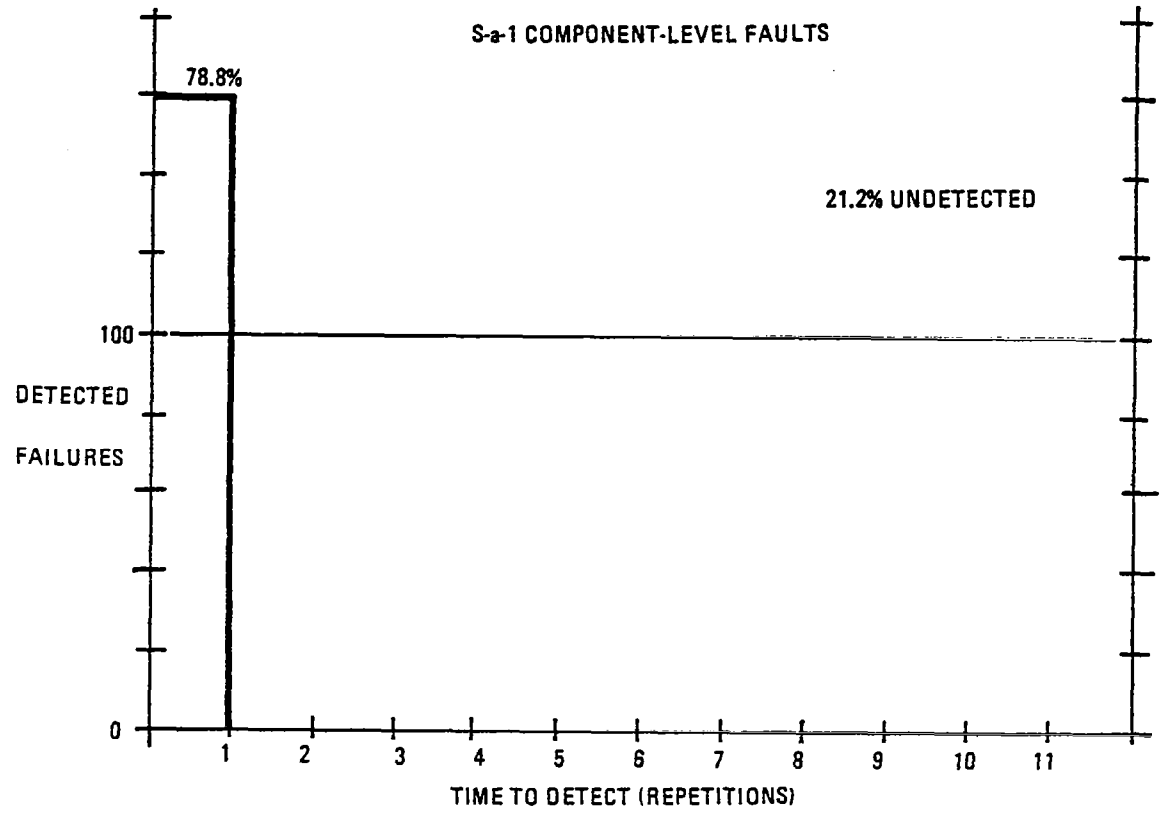


FIGURE 7f


S-a-1 COMPONENT-LEVEL FAULTS
78.8%
21.2% UNDETECTED
100
DETECTED
FAILURES
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

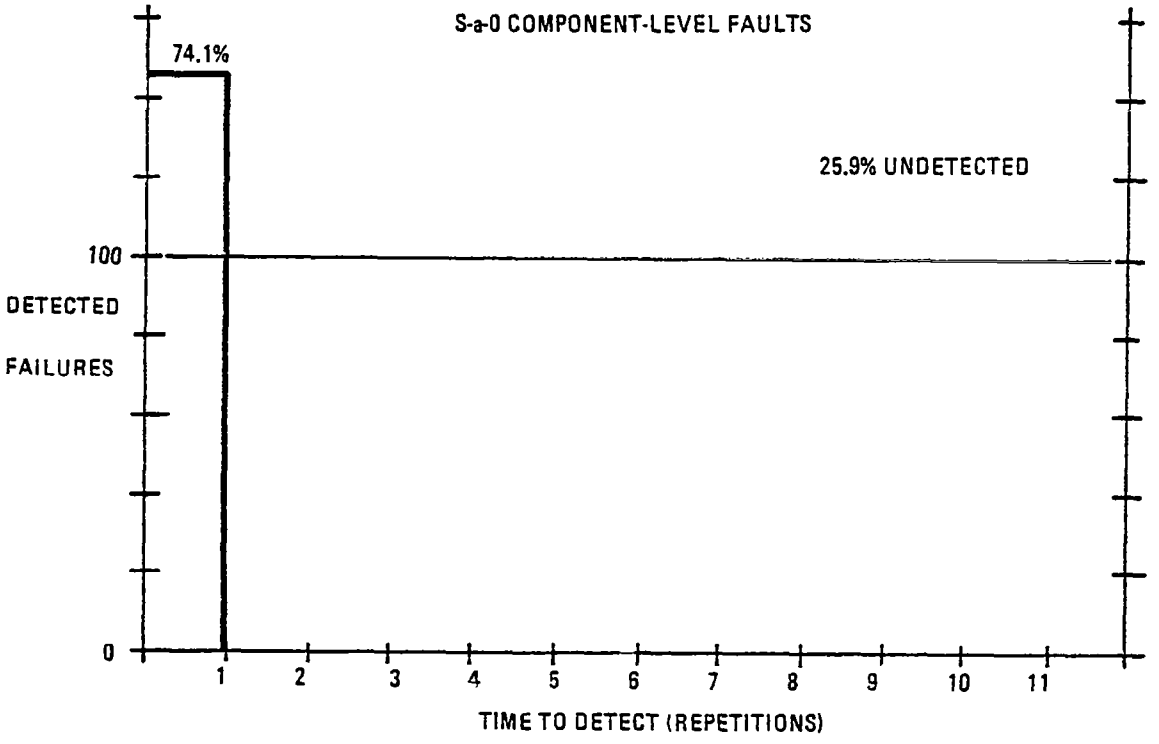

S-a-0 COMPONENT-LEVEL FAULTS
74.1%
25.9% UNDETECTED
100
DETECTED
FAILURES
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 7g

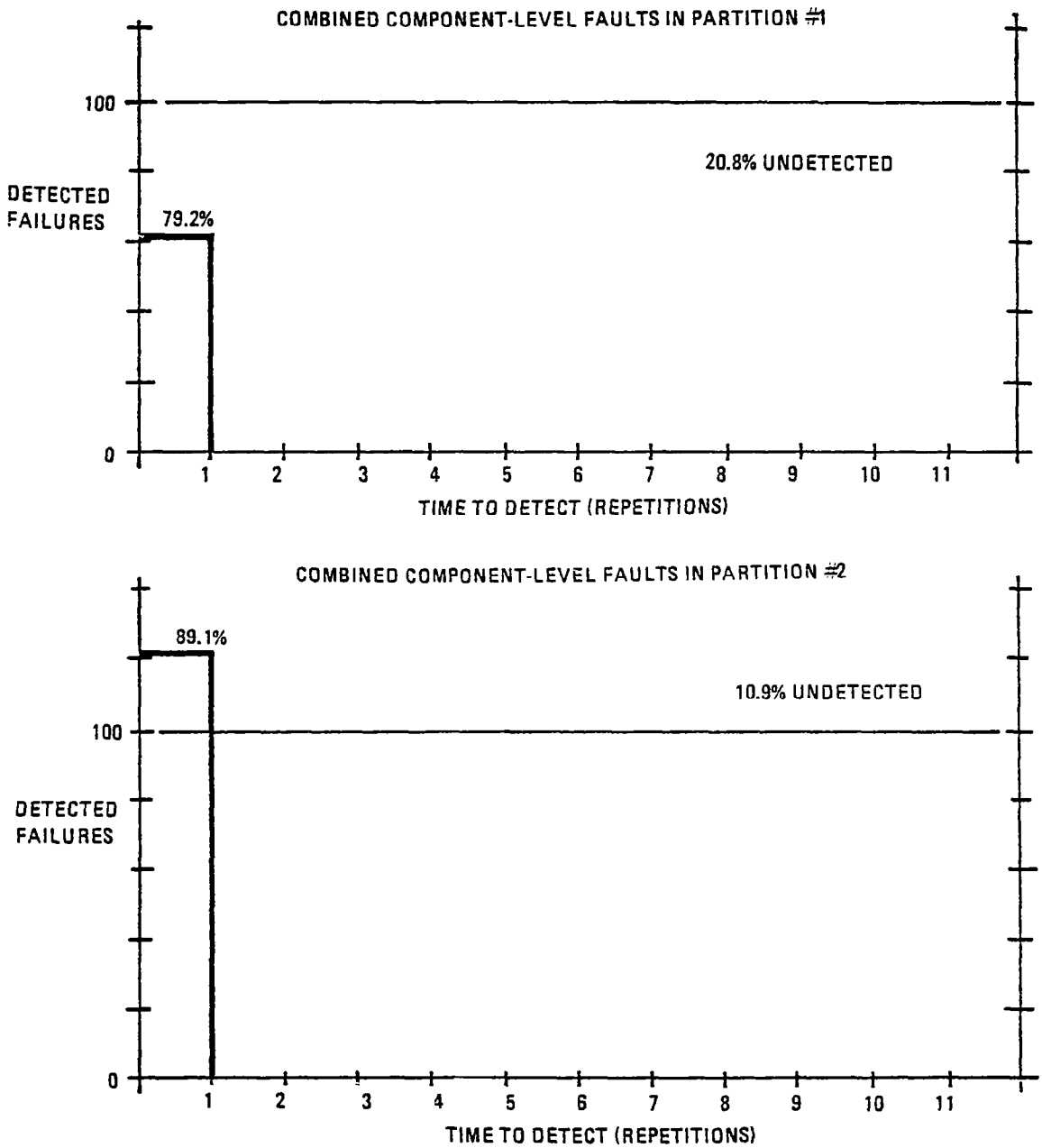
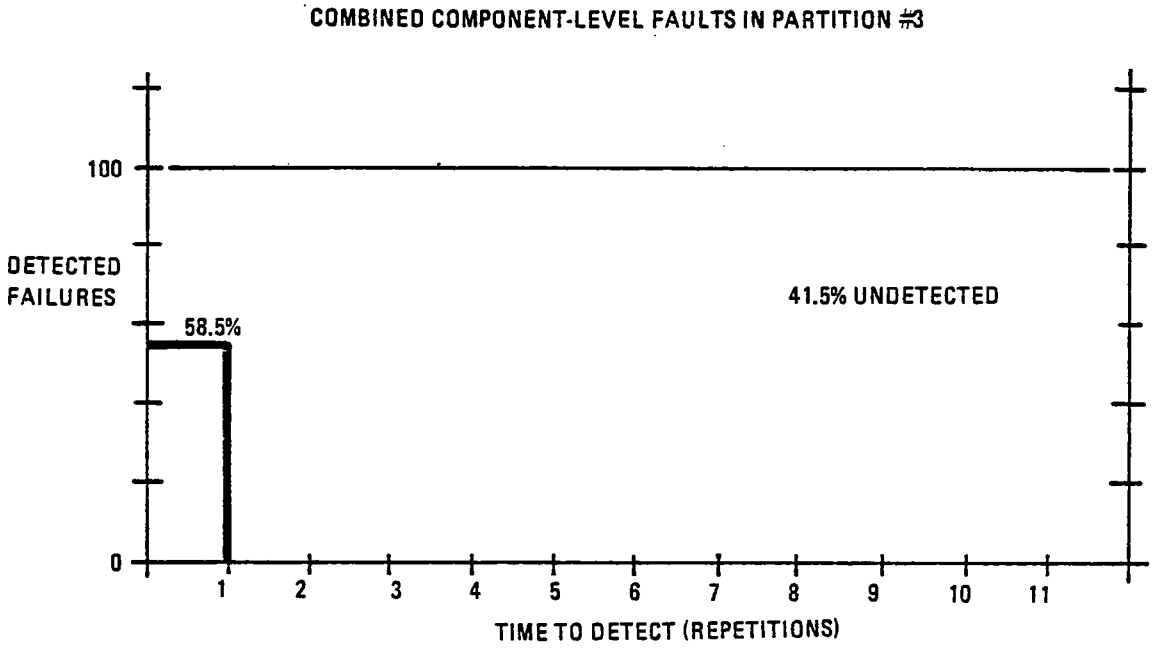

COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #1
100
20.8% UNDETECTED
DETECTED FAILURES
79.2%
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)
COMBINED COMPONENT-LEVEL FAULTS IN PARTITION #2
89.1%
10.9% UNDETECTED
100
DETECTED FAILURES
0
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 7h

LINCON



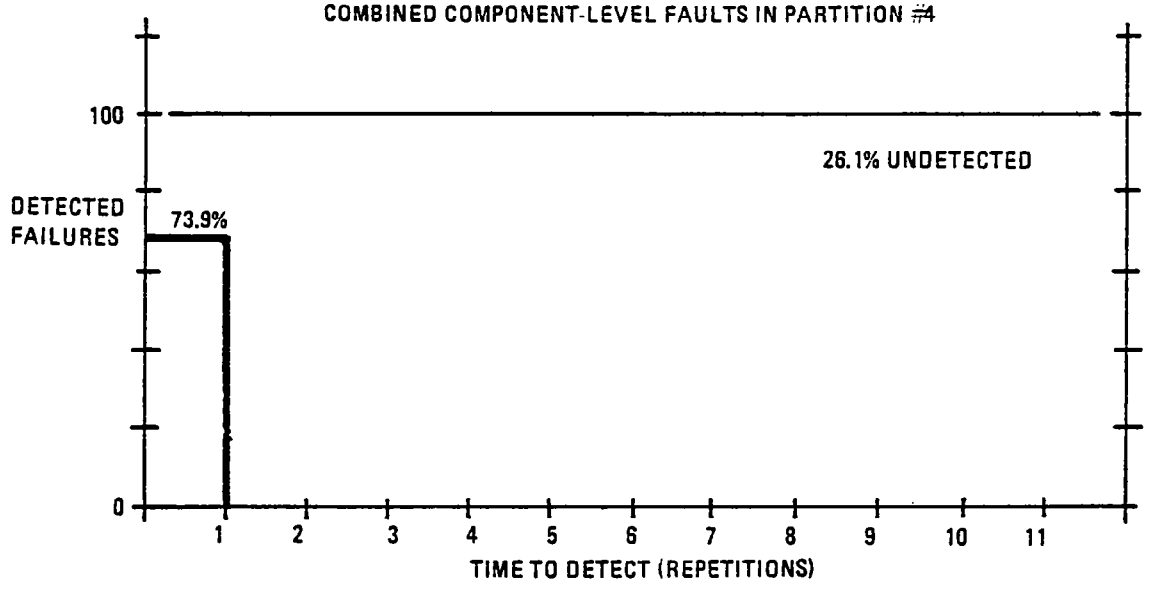


FIGURE 7i

GATE-LEVEL FAULTS

PROGRAM	S-a-Ø Faults				S-a-1 Faults			
	INJECTED	DETECTED	PERCENT DETECTED 1st REPETITION	PERCENT UNDETECTED	INJECTED	DETECTED	PERCENT DETECTED 1st REPETITION	PERCENT UNDETECTED
FETSTO	503	136	27.0	63.6	497	163	32.8	59.8
ADDSUB	296	84	28.4	64.2	304	117	38.5	54.9
FIB	296	90	30.4	62.2	304	120	39.5	54.3
SERCOM	503	183	36.4	63.6	497	212	42.7	57.3
QUAD	296	114	38.5	57.4	304	145	47.8	49.3
LINCON	296	148	50.0	50.0	304	162	53.3	46.7

COMPONENT-LEVEL FAULTS

PROGRAM	S-a-Ø Faults				S-a-1 Faults			
	INJECTED	DETECTED	PERCENT DETECTED 1st REPETITION	PERCENT UNDETECTED	INJECTED	DETECTED	PERCENT DETECTED 1st REPETITION	PERCENT UNDETECTED
FETSTO	197	92	46.7	40.1	203	113	55.7	31.0
ADDSUB	197	100	50.8	37.1	203	128	63.1	27.6
FIB	197	111	56.3	33.5	203	134	66.0	22.7
SERCOM	197	112	56.9	43.1	203	147	72.4	27.6
QUAD	197	133	67.5	26.9	203	154	75.9	20.2
LINCON	197	146	74.1	25.9	203	160	78.8	21.2

SUMMARY OF PHASE I RESULTS

TABLE 11

GATE-LEVEL FAULTS (86.4% DETECTION)

PARTITION	DETECTED FAULTS		FAULTS INJECTED		DETECTED FAULTS		TOTAL	TOTAL
	m_1	n_1	m	n	TEST KNOWN	WILD BRANCH	DETECTED	INJECTED
P1	14	14	16	16	1	27	28	32
P2	26	32	30	35	25	33	58	65
P3	20	19	20	19	17	22	39	39
P4	33	26	37	31	43	16	59	68
P5	2	7	8	9	7	2	9	17
P6	3	2	4	4	2	3	5	8
TOTAL	98	100	115	114	95	103	198	229*

m = S-a-0 faults

n = S-a-1 faults

* 71 faults were disqualified as indistinguishable

(COMPONENT-LEVEL FAULTS (97.7% DETECTION)

PARTITION	DETECTED FAULTS		FAULTS INJECTED		DETECTED FAULTS		TOTAL	TOTAL
	m_1	n_1	m	n	TEST KNOWN	WILD BRANCH	DETECTED	INJECTED
P1	15	19	15	20	5	29	34	35
P2	38	34	38	35	35	37	72	73
P3	20	21	21	22	16	25	41	43
P4	15	23	15	23	23	15	38	38
P5								
P6								
TOTAL	88	97	89	100	79	106	185	189*

* 11 faults were disqualified as indistinguishable.

SELF-TEST DATA

TABLE 12

TABLE 13

FAULT DETECTION BY THE INDIVIDUAL TESTS COMPRISING SELF-TEST

COMPONENT-LEVEL			GATE-LEVEL		
TEST # - 1	FREQ		TEST # - 1	FREQ	
0	106		0	103	
1	2		1	1	
2	6		2	1	
3	2		3	2	
5	7		4	1	
6	4		5	4	
7	4		6	2	
8	5		7	4	
9	1		8	9	
10	2		10	3	
11	1		12	1	
18	4		16	1	
19	2		18	9	
21	1		19	4	
22	2		21	2	
23	1		22	15	
28	2		24	1	
35	3		25	1	
37	1		28	1	
38	1		33	1	
44	2		34	1	
49	1		35	3	
69	2		36	1	
95	1		38	1	
102	1		39	1	
108	1		43	3	
110	1		53	1	
144	1		92	1	
187	2		98	1	
241	15[1]	(UNDETECTED)	100	1	
	184		102	1	
			103	1	
			112	1	
OUT OF RANGE TEST #	16		114	2	
			177	1	
	200		236	1	
			239	1	
			241	102[2]	(UNDETECTED)
			OUT OF RANGE TEST #	10	
				300	

(1) 11 were subsequently disqualified as indistinguishable

(2) 71 were subsequently disqualified as indistinguishable

GATE-LEVEL FAULTS

Partition	S-a-Ø Faults Percent Detected	S-a-1 Faults Percent Detected
#1	87.5	87.5
#2	36.7	91.4
#3	100.0	100.0
#4	89.2	83.9
#5	25.0	77.8
#6	75.0	50.0

COMPONENT-LEVEL FAULTS

Partition	S-a-Ø Faults Percent Detected	S-a-1 Faults Percent Detected
#1	100.0	95.0
#2	100.0	97.1
#3	95.2	95.5
#4	100.0	100.0

SELF-TEST COVERAGE SUMMARIES BY PARTITION

TABLE 14

T=Exact estimate
N=Approximate estimate
A=Empirical value

TEST			(1-a)	(a)	P	Po
FETSTO/GATE	COMBINED	T	0.5614	0.4386	0.7776	0.3845
		N	0.5359	0.4641	0.7807	0.383
		A				
	S-a-Ø	T	0.5168	0.4832	0.7413	0.3647
		N	0.5	0.5	0.7432	0.3638
		A				
	S-a-1	T	0.6137	0.3863	0.8099	0.4049
		N	0.575	0.425	0.815	0.402
		A				
FETSTO/COMP	COMBINED	T	0.5280	0.4720	0.7927	0.6465
		N	0.5093	0.4907	0.7946	0.6450
		A				
	S-a-Ø	T	0.6603	0.3397	0.7698	0.6066
		N	0.6061	0.3939	0.7797	0.5990
		A				
	S-a-1	T	0.3594	0.6406	0.8070	0.6898
		N	0.3571	0.6429	0.8071	0.6897
		A				
FIB/GATE	COMBINED	T	0.3177	0.6823	0.8366	0.4184
		N	0.3167	0.6833	0.8367	0.4183
		A				
	S-a-Ø	T	0.2909	0.7091	0.8035	0.3784
		N	0.2903	0.7097	0.8036	0.3784
		A				
	S-a-1	T	0.3466	0.6534	0.8632	0.4573
		N	0.3448	0.6552	0.8633	0.4572
		A				
FIB/COMP	COMBINED	T	0.2958	0.7042	0.8507	0.7200
		N	0.2951	0.7049	0.8507	0.7200
		A				
	S-a-Ø	T	0	1.000	0.8473	0.6650
		N	0	1.000	0.8473	0.5650
		A				
	S-a-1	T	0.4468	0.5532	0.8531	0.7738
		N	0.4390	0.5610	0.8535	0.7734
		A				
ADDSUB/GATE	COMBINED	T	0.5309	0.4691	0.8254	0.4058
		N	0.5116	0.4884	0.8272	0.4050
		A				
	S-a-Ø	T	0.6051	0.3949	0.7874	0.3604
		N	0.5686	0.4314	0.7925	0.3581
		A				
	S-a-1	T	0.4353	0.5647	0.8536	0.4509
		N	0.4286	0.5714	0.8540	0.4507
		A				
ADDSUB/COMP	COMBINED	T	0.3401	0.6599	0.8413	0.6776
		N	0.3385	0.6615	0.8413	0.6775
		A				
	S-a-Ø	T	0.3153	0.6847	0.8064	0.6295
		N	0.3143	0.6857	0.8065	0.6294
		A				
	S-a-1	T	0.3693	0.6307	0.8706	0.7242
		N	0.3667	0.6333	0.8707	0.7241
		A				

URN MODEL DISTRIBUTIONS
AND PARAMETER ESTIMATES

TABLE 15

*occupancy probabilities

Cell* #1	Cell* #2	Cell* #3	Cell* #4	Cell* #5	Cell* #6	Cell* #7	Cell* #8	Cell* #9
0.299	0.038	0.021	0.012	0.007	0.004	0.002	0.001	0.617
0.299	0.039	0.021	0.011	0.006	0.003	0.002	0.001	0.617
0.299	0.048	0.021	0	0	0.006	0.002	0.007	0.617
0.270	0.046	0.024	0.012	0.006	0.003	0.002	0.001	0.6362
0.270	0.047	0.023	0.012	0.006	0.003	0.001	0.001	0.6362
0.270	0.044	0.038	0	0	0.008	0	0.004	0.636
0.328	0.030	0.018	0.011	0.007	0.004	0.003	0.002	0.5976
0.328	0.032	0.018	0.010	0.006	0.003	0.002	0.001	0.598
0.328	0.052	0.004	0	0	0.004	0.004	0.010	0.598
0.5125	0.063	0.033	0.018	0.009	0.005	0.003	0.001	0.3550
0.5125	0.065	0.033	0.017	0.009	0.004	0.002	0.001	0.355
0.513	0.085	0.025	0.003	0	0	0.013	0.008	0.355
0.467	0.047	0.031	0.021	0.014	0.009	0.006	0.004	0.4010
0.467	0.052	0.032	0.019	0.012	0.007	0.004	0.003	0.4010
0.467	0.061	0.036	0.005	0	0	0.025	0.005	0.401
0.557	0.085	0.031	0.011	0.004	0.001	0.001	0.000	0.3103
0.557	0.086	0.031	0.011	0.004	0.001	0.001	0.000	0.3103
0.557	0.108	0.015	0	0	0	0	0.010	0.310
0.350	0.047	0.015	0.005	0.001	0	0	0	0.5816
0.350	0.047	0.015	0.005	0.001	0	0	0	0.5817
0.350	0.055	0.007	0.002	0.002	0.002	0	0.002	0.582
0.304	0.053	0.015	0.004	0.001	0	0	0	0.6216
0.304	0.053	0.015	0.004	0.001	0	0	0	0.6216
0.304	0.064	0.003	0.003	0	0	0	0.003	0.622
0.395	0.041	0.014	0.005	0.002	0.001	0	0	0.5427
0.395	0.041	0.014	0.005	0.002	0.001	0	0	0.5428
0.395	0.046	0.010	0	0.003	0.003	0	0	0.543
0.613	0.076	0.022	0.007	0.002	0.001	0	0	0.280
0.613	0.076	0.022	0.007	0.002	0.001	0	0	0.280
0.613	0.090	0.008	0.003	0.003	0.003	0	0.003	0.280
0.5635	0.1015	0	0	0	0	0	0	0.335
0.5635	0.1015	0	0	0	0	0	0	0.335
0.563	0.102	0	0	0	0	0	0	0.335
0.660	0.063	0.028	0.013	0.006	0.003	0.001	0.001	0.2266
0.660	0.064	0.028	0.012	0.005	0.002	0.001	0	0.2266
0.660	0.079	0.015	0.005	0.005	0.005	0	0.005	0.227
0.335	0.033	0.018	0.009	0.005	0.003	0.001	0.001	0.5950
0.335	0.034	0.018	0.009	0.005	0.002	0.001	0.001	0.5950
0.335	0.027	0.030	0.003	0.005	0.003	0.002	0	0.595
0.284	0.030	0.018	0.011	0.007	0.004	0.002	0.001	0.6419
0.284	0.032	0.018	0.010	0.006	0.003	0.002	0.001	0.6419
0.284	0.024	0.034	0	0.007	0.007	0.003	0	0.642
0.385	0.037	0.016	0.007	0.003	0.001	0.001	0	0.5493
0.385	0.038	0.016	0.007	0.003	0.001	0.001	0	0.5493
0.385	0.030	0.026	0.007	0.003	0	0	0	0.549
0.570	0.071	0.024	0.008	0.003	0.001	0	0	0.3225
0.570	0.071	0.024	0.008	0.003	0.001	0	0	0.3225
0.570	0.068	0.035	0	0	0.005	0	0	0.323
0.508	0.083	0.026	0.008	0.003	0.001	0	0	0.3705
0.508	0.084	0.026	0.008	0.003	0.001	0	0	0.3706
0.508	0.081	0.036	0	0	0.005	0	0	0.371
0.631	0.059	0.022	0.008	0.003	0.001	0	0	0.2759
0.631	0.059	0.022	0.008	0.003	0.001	0	0	0.2759
0.631	0.054	0.034	0	0	0.005	0	0	0.276

URN MODEL DISTRIBUTIONS
AND PARAMETER ESTIMATES

TABLE 15

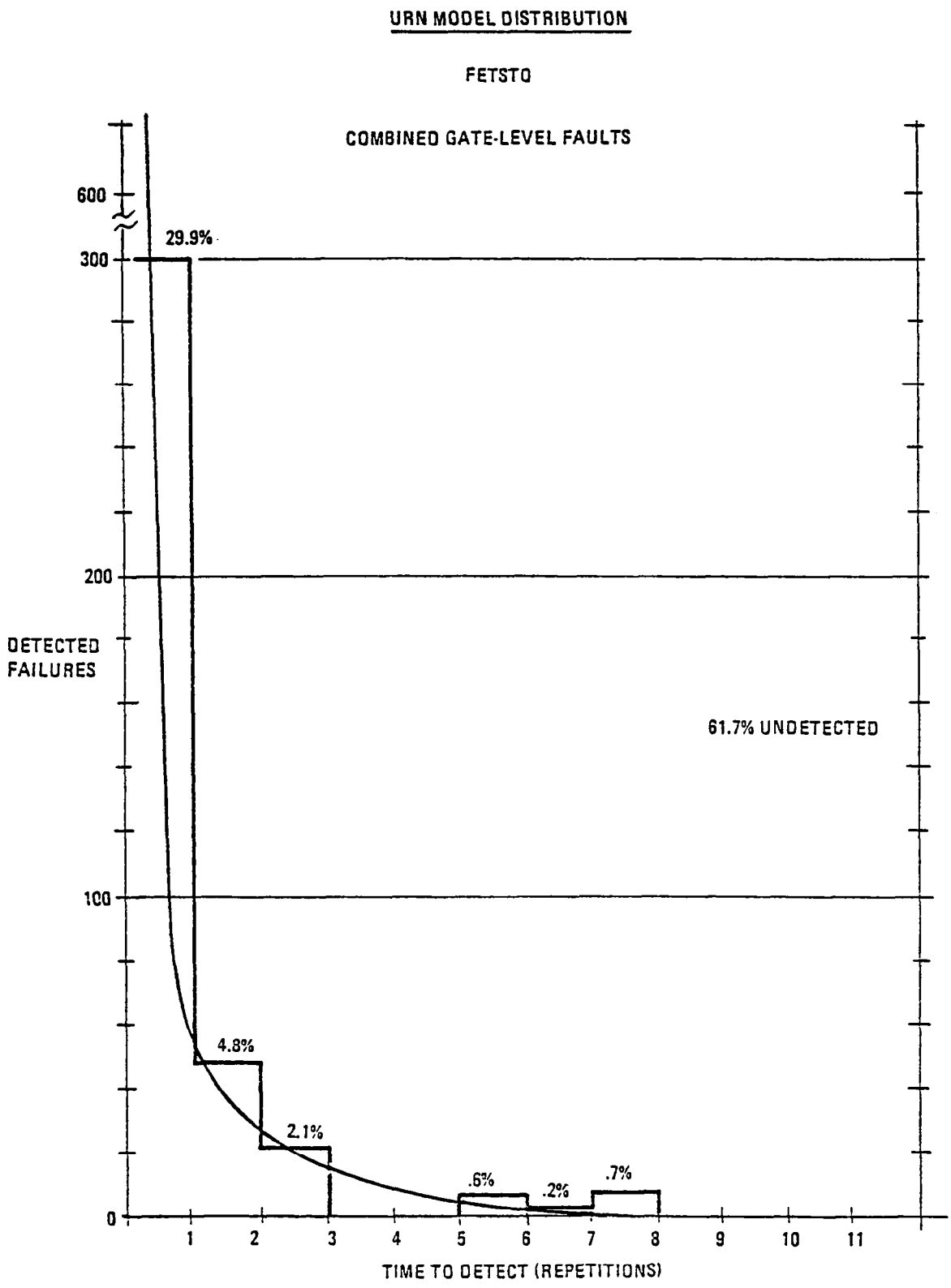

URN MODEL DISTRIBUTION
FETSTO
COMBINED GATE-LEVEL FAULTS
600
300
200
100
0
29.9%
4.8%
2.1%
.6%
.2%
.7%
61.7% UNDETECTED
DETECTED FAILURES
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 8a

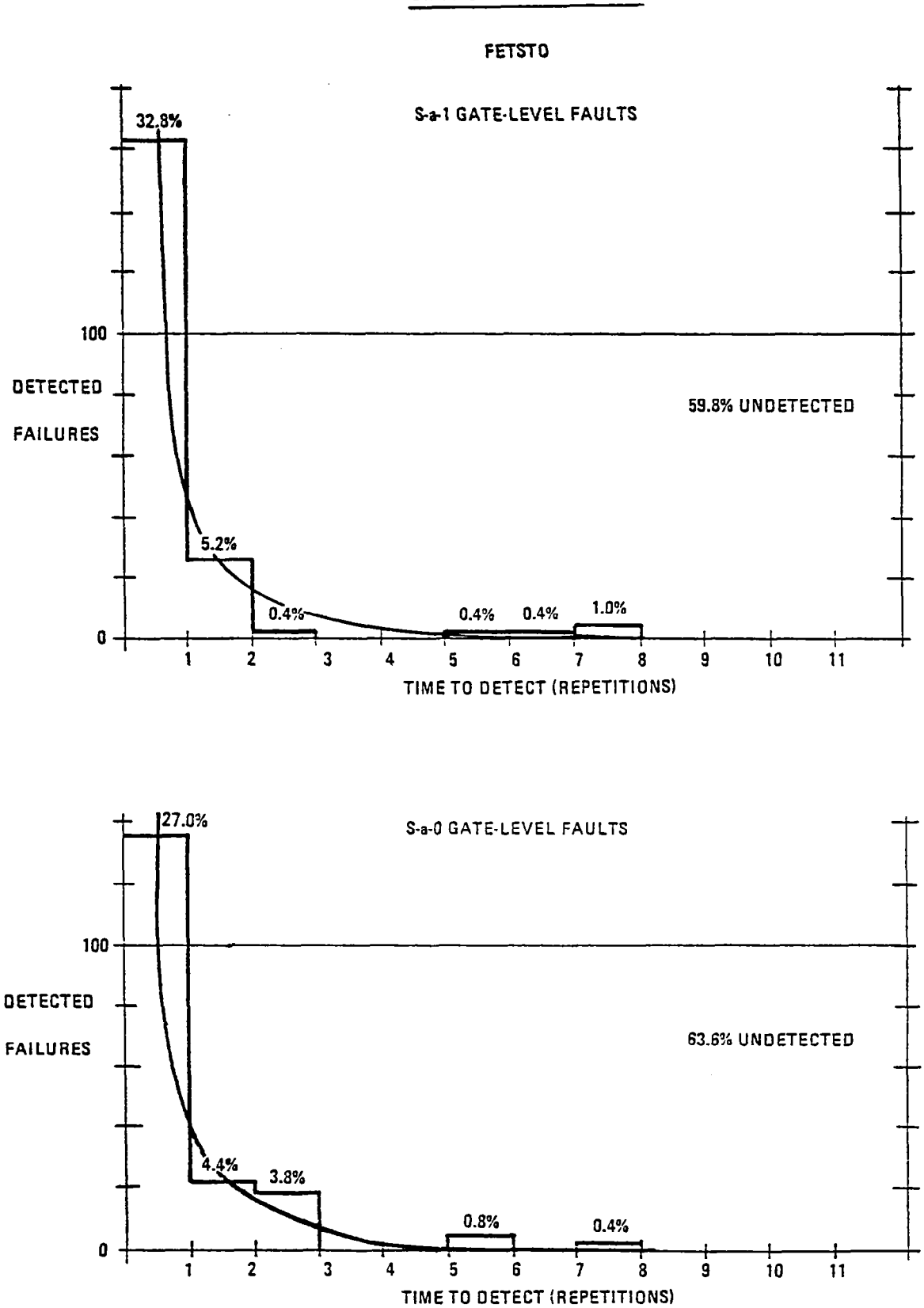

URN MODEL DISTRIBUTION
FETSTO
S-a-1 GATE-LEVEL FAULTS
32.8%
5.2%
0.4%
0.4%
0.4%
1.0%
59.8% UNDETECTED
100
0
DETECTED
FAILURES
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)
S-a-0 GATE-LEVEL FAULTS
27.0%
4.4%
3.8%
0.8%
0.4%
63.6% UNDETECTED
100
0
DETECTED
FAILURES
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 8b

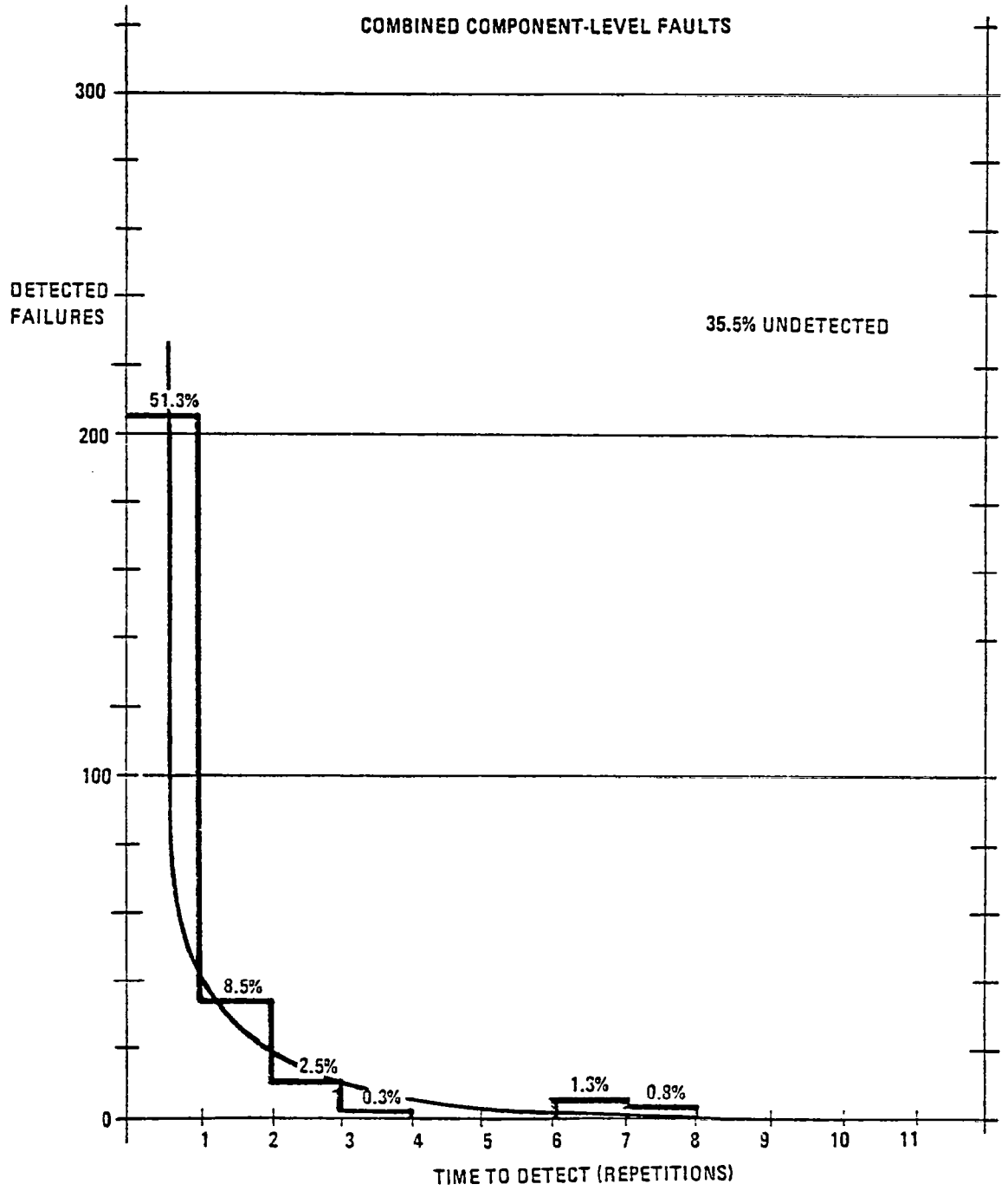

URN MODEL DISTRIBUTION
FETSTO
COMBINED COMPONENT-LEVEL FAULTS
300
200
100
0
DETECTED FAILURES
35.5% UNDETECTED
51.3%
8.5%
2.5%
0.3%
1.3%
0.8%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 8c

URN MODEL DISTRIBUTION

FETSTO

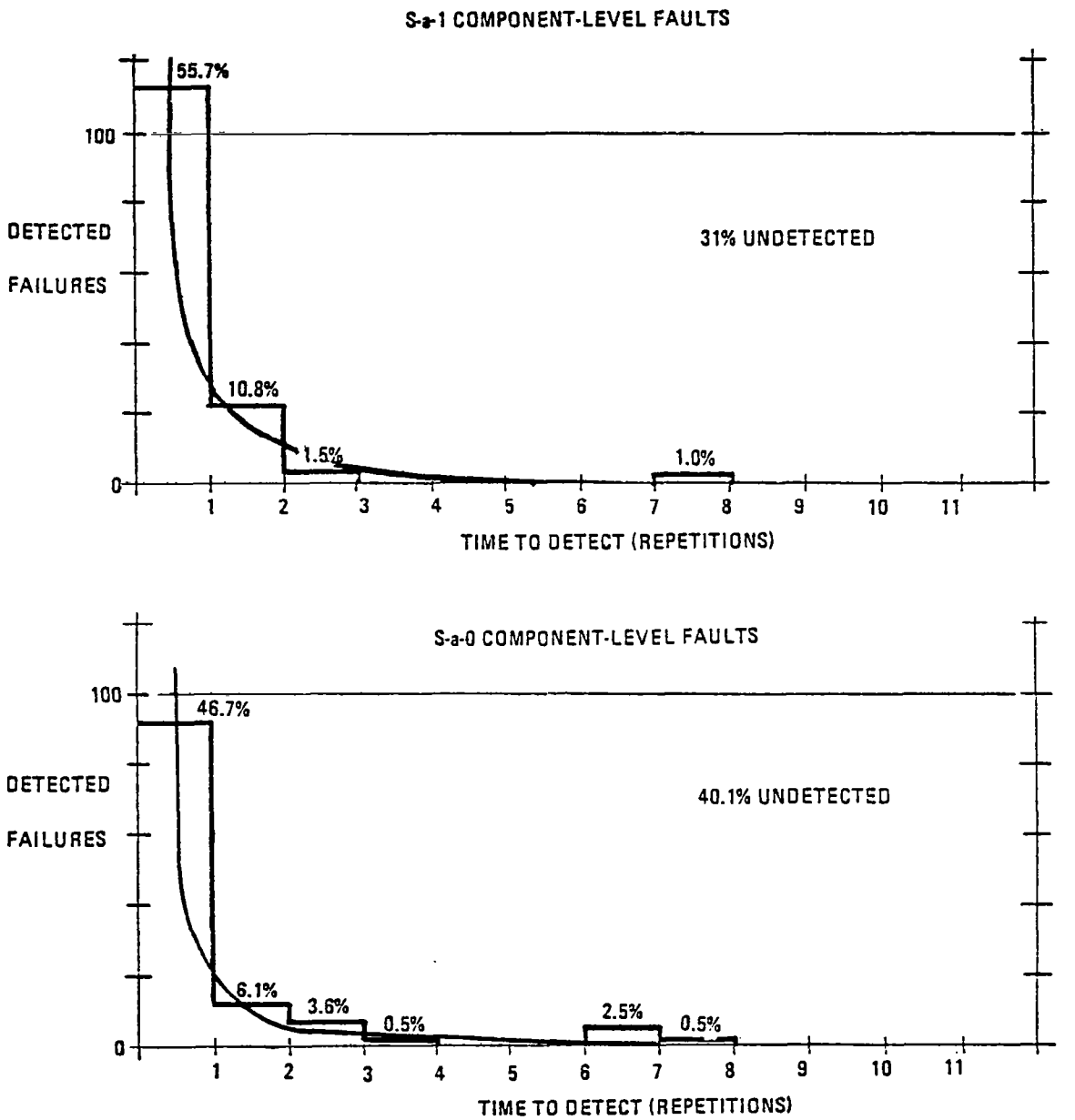


FIGURE 8d

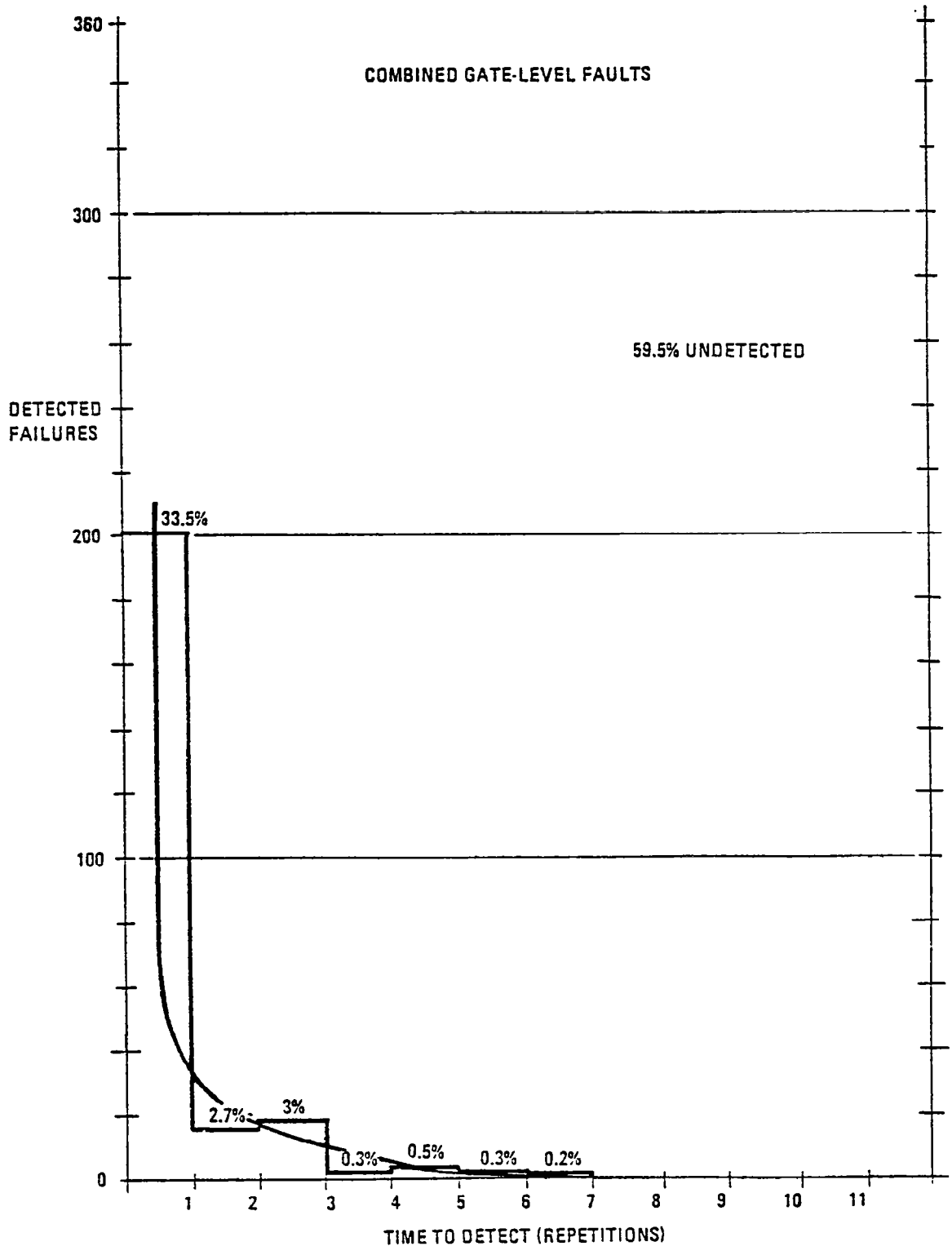
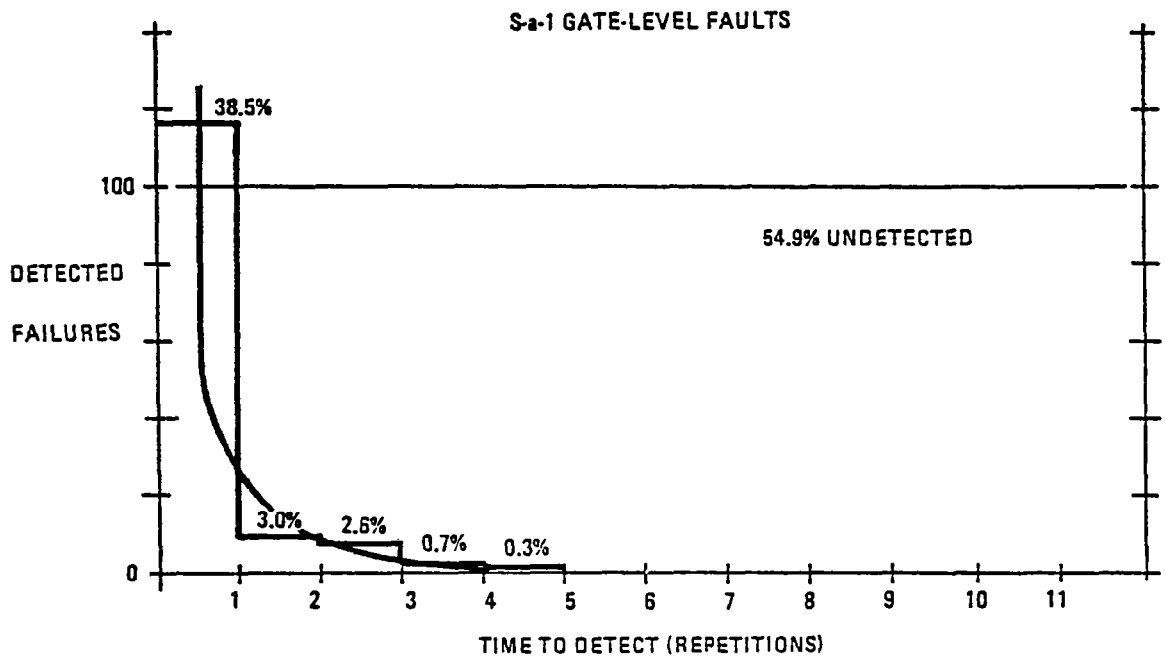

URN MODEL DISTRIBUTION
ADDSUB
COMBINED GATE-LEVEL FAULTS
59.5% UNDETECTED
DETECTED FAILURES
360
300
200
100
0
33.5%
2.7%
3%
0.3%
0.5%
0.3%
0.2%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 9a

URN MODEL DISTRIBUTION

ADDSUB



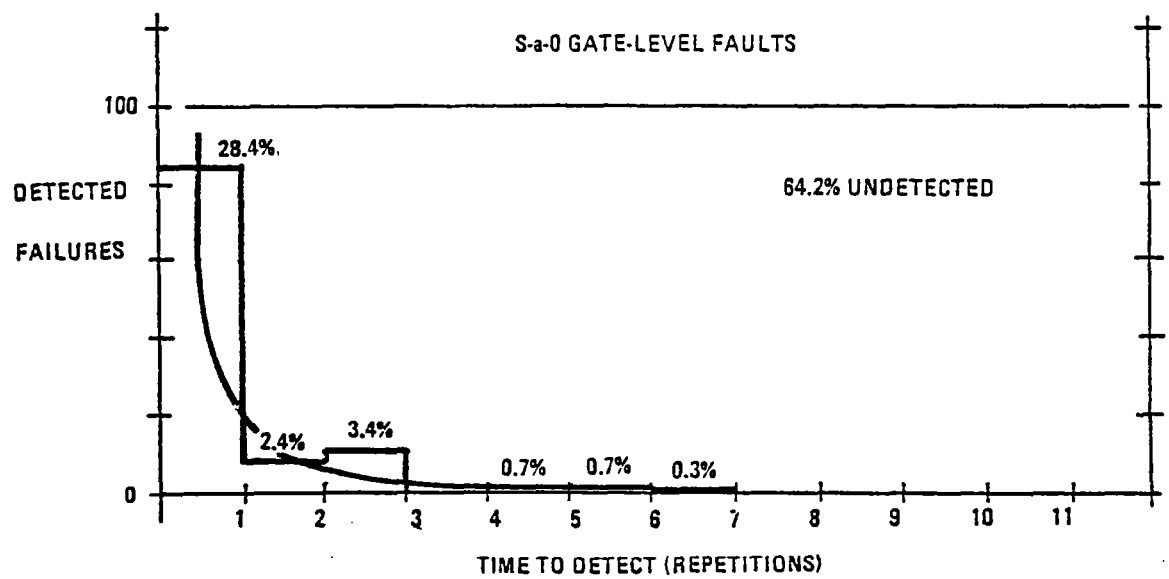


FIGURE 9b

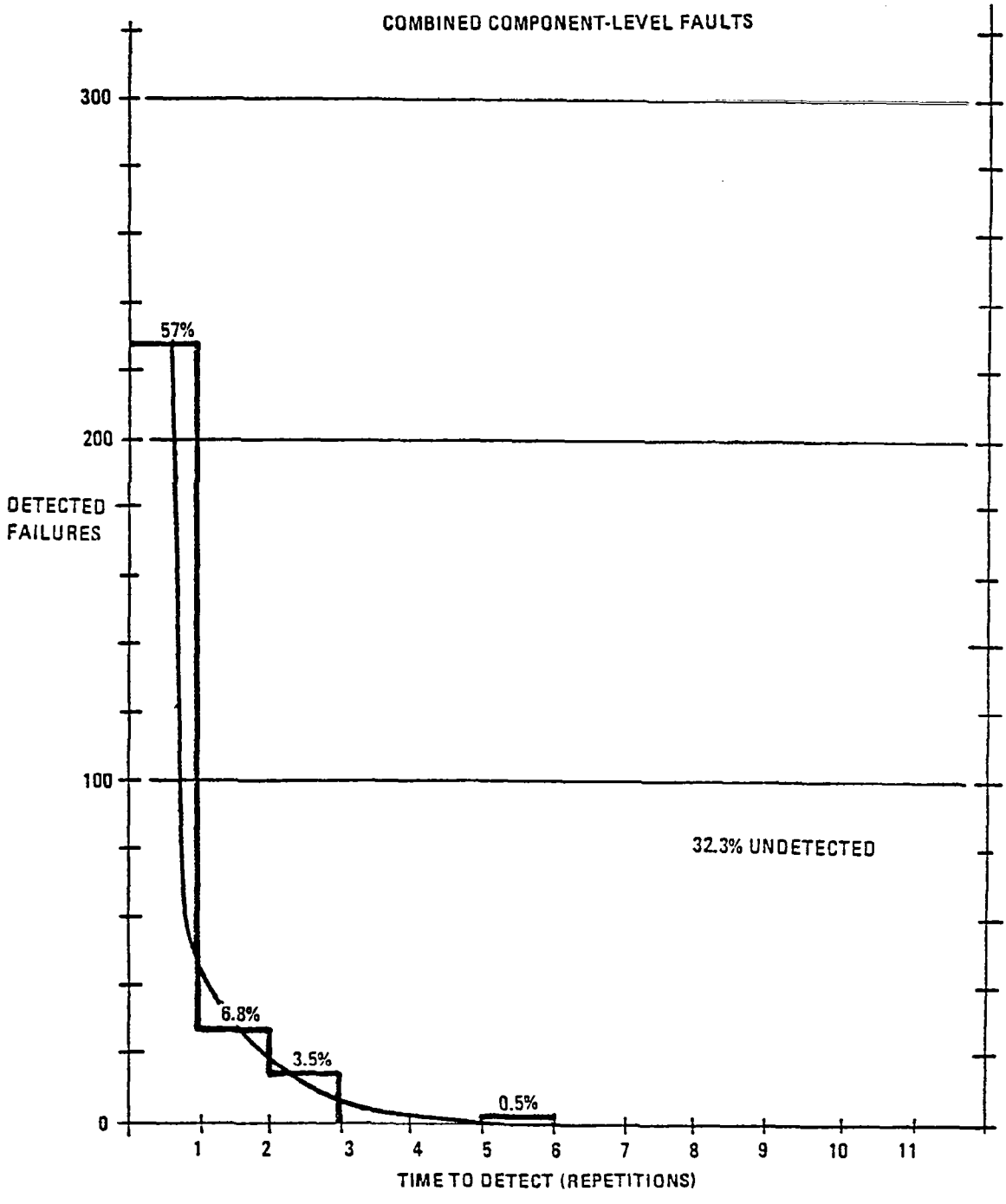
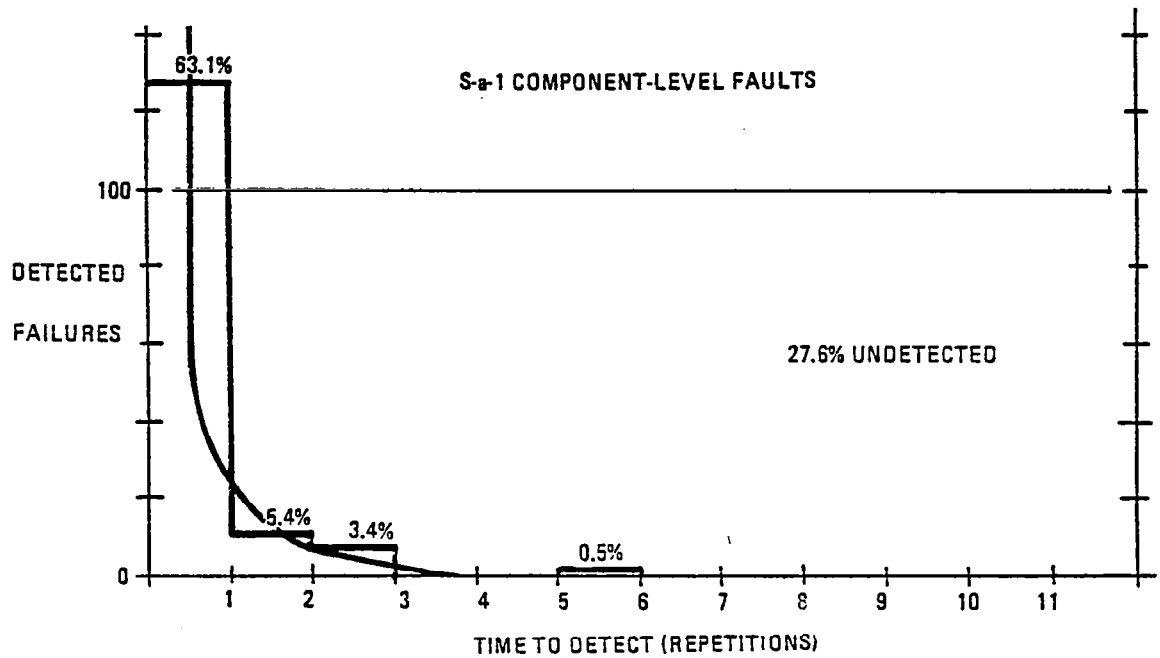

URN MODEL DISTRIBUTION
ADDSUB
COMBINED COMPONENT-LEVEL FAULTS
300
200
100
0
DETECTED FAILURES
57%
6.8%
3.5%
0.5%
32.3% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 9c

URN MODEL DISTRIBUTION

ADDSUB



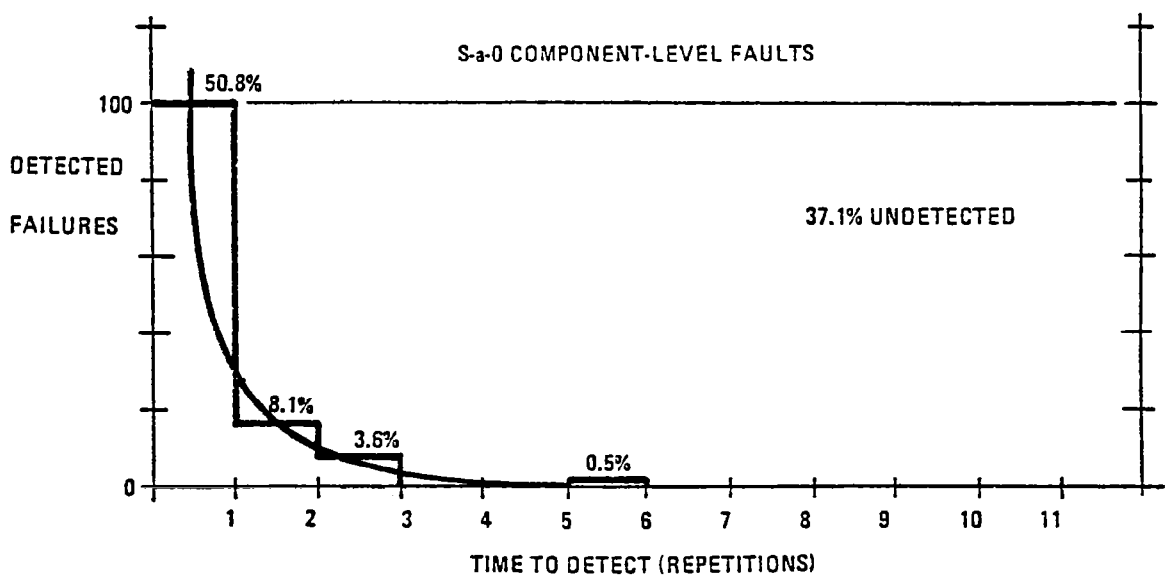


FIGURE 9d

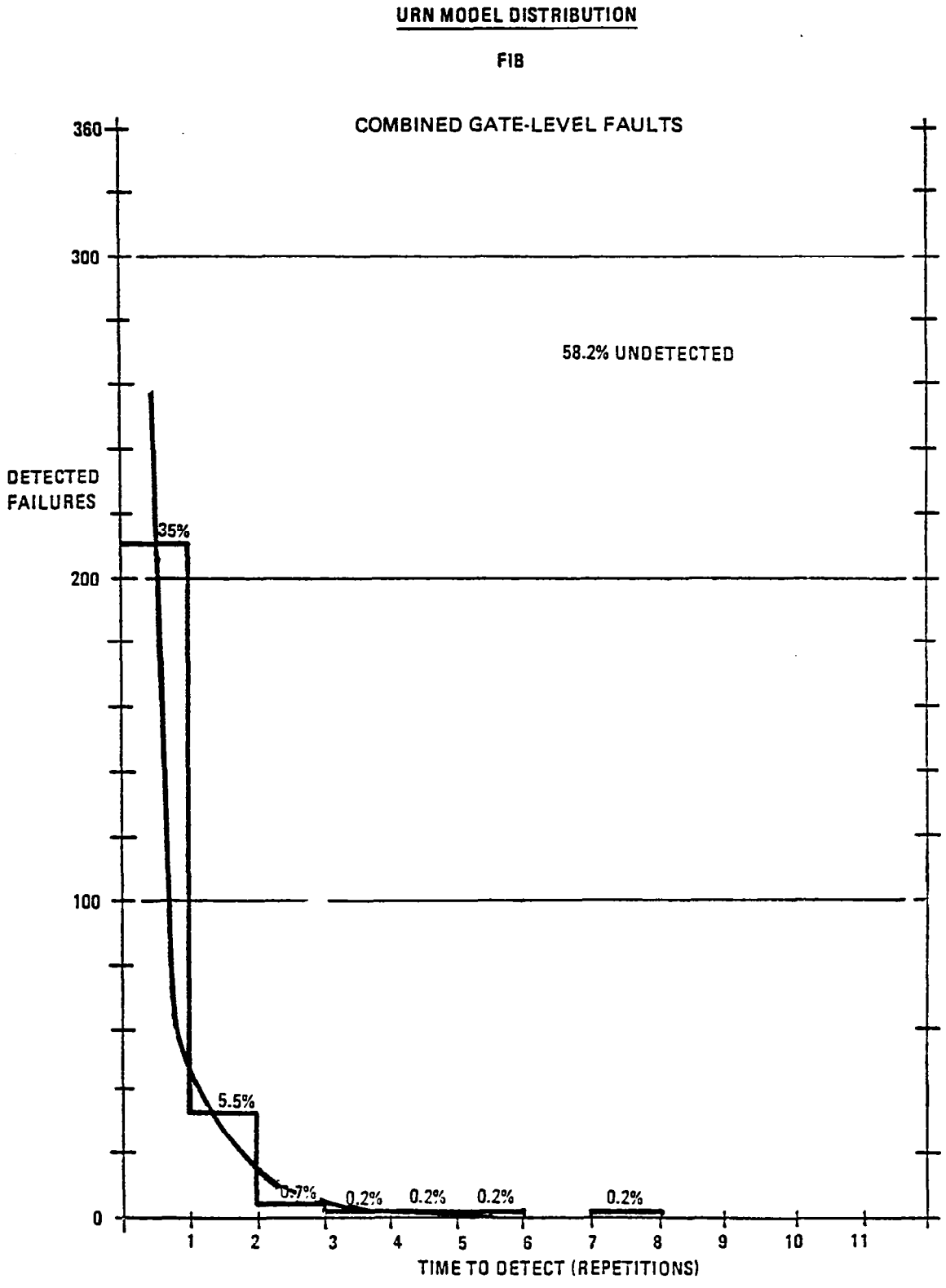
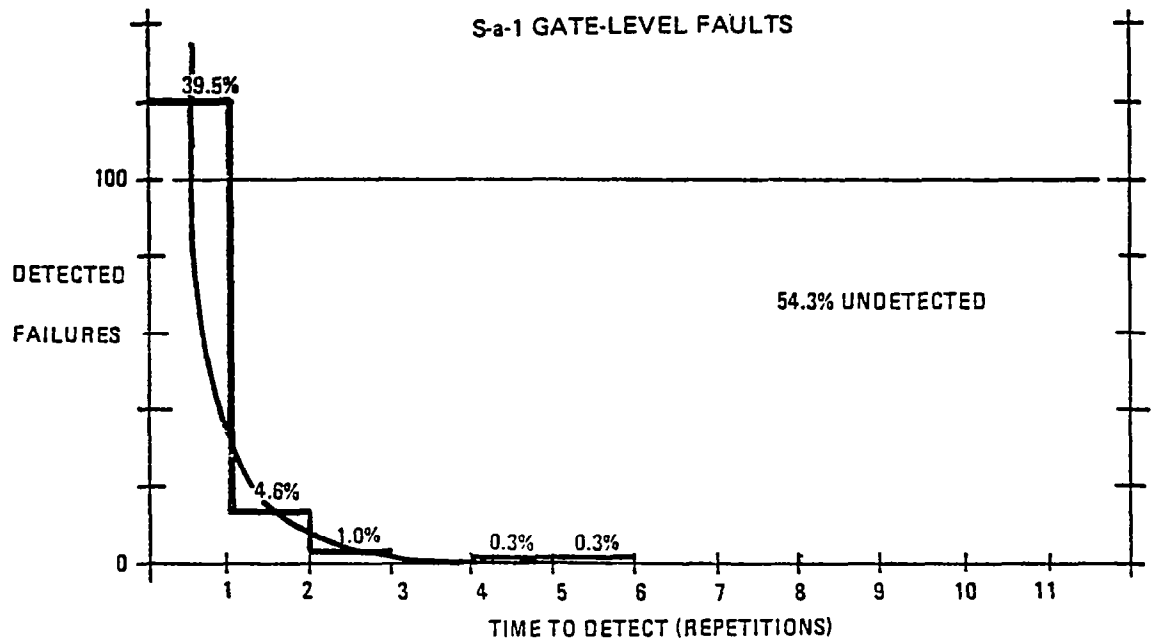

URN MODEL DISTRIBUTION
FIB
COMBINED GATE-LEVEL FAULTS
360
300
200
100
0
58.2% UNDETECTED
DETECTED FAILURES
35%
5.5%
0.7%
0.2%
0.2%
0.2%
0.2%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 10a

URN MODEL DISTRIBUTION

FIB



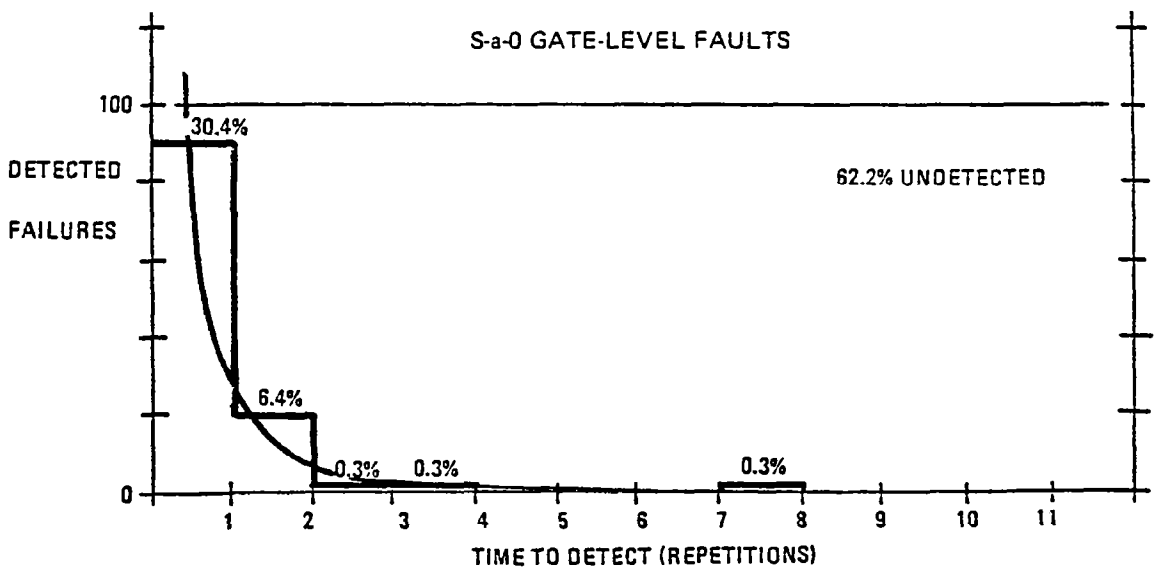


FIGURE 10b

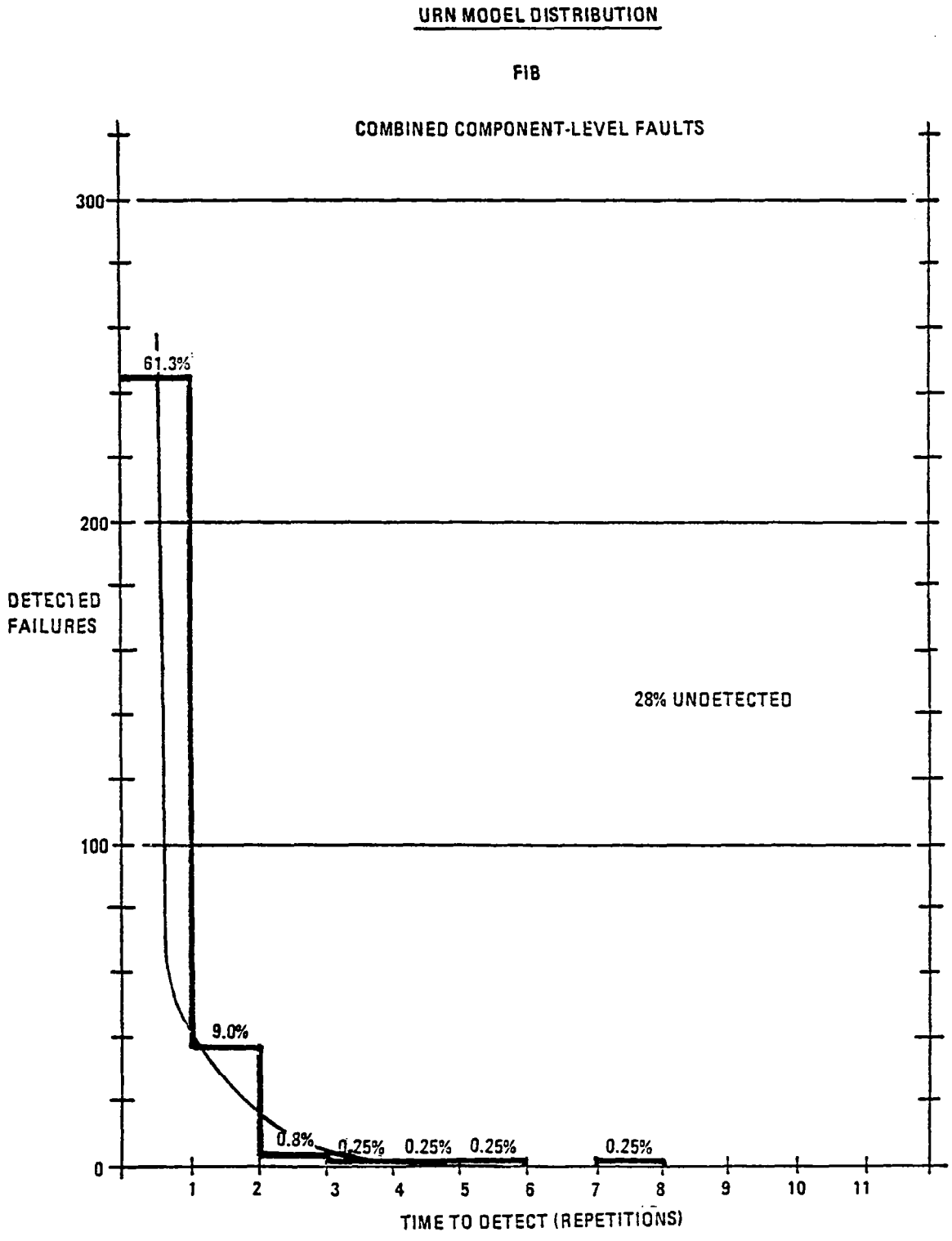
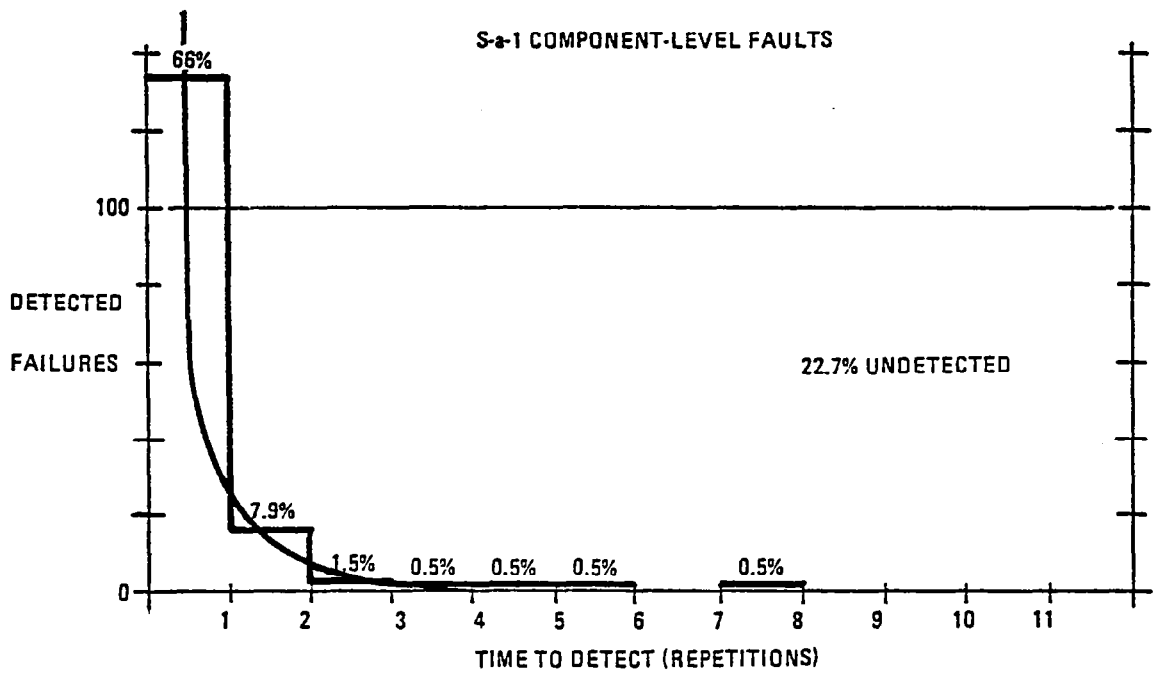

URN MODEL DISTRIBUTION
FIB
COMBINED COMPONENT-LEVEL FAULTS
300
200
100
0
DETECTED FAILURES
61.3%
9.0%
0.8%
0.25%
0.25%
0.25%
0.25%
28% UNDETECTED
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 10c

URN MODEL DISTRIBUTION

FIB



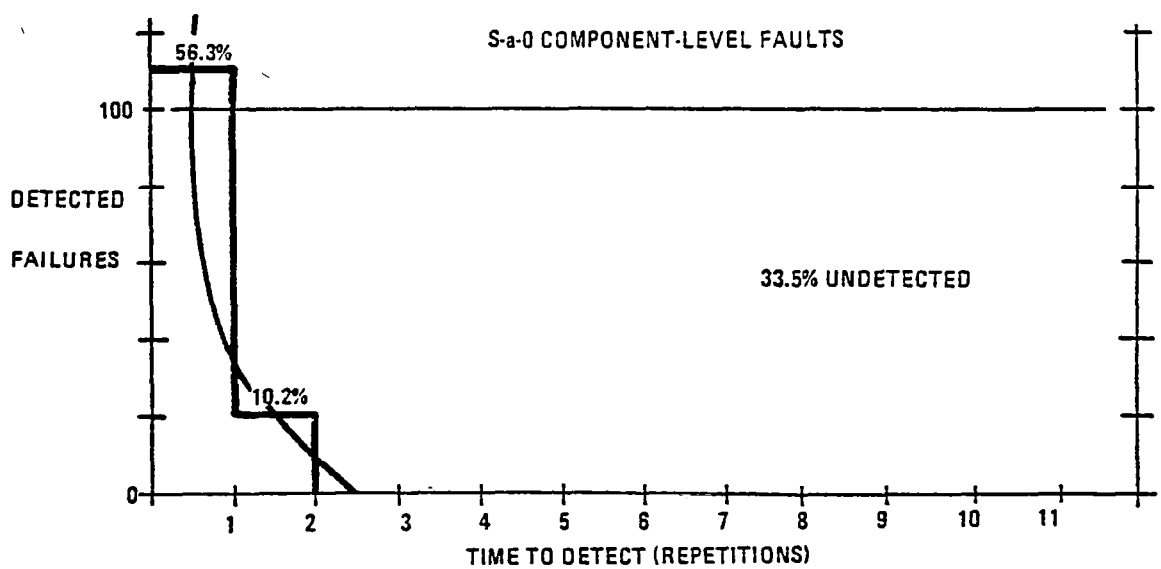


FIGURE 10d

TEST		$\sigma_{(PP)}$	$\sigma_{(aa)}$	$\sigma_{(PoPo)}$	$\sigma_{(Pa)}$	$\sigma_{(PPo)}$	$\sigma_{(aPo)}$
FETSO/GATE	COMBINED	.45874E-3	.18519E-2	.23886E-3	.71435E-4	-.44562E-5	-.35322E-4
	S-a-Ø	.10553E-2	.33012E-2	.46305E-3	.85297E-4	-.48770E-5	-.41971E-4
	S-a-1	.80100E-3	.42498E-2	.49385E-3	.25294E-3	-.18026E-4	-.12646E-3
FETSTO/COMP	COMBINED	.64313E-3	.29281E-2	.57644E-3	.73853E-4	-.62756E-5	-.60236E-4
	S-a-Ø	.16216E-2	.61628E-2	.12976E-2	.67009E-3	-.10948E-3	-.52802E-3
	S-a-1	.11130E-2	.57103E-2	.10547E-2	.13421E-4	-.62136E-6	-.11471E-4
FIB/GATE	COMBINED	.54472E-3	.36941E-2	.40560E-3	.36381E-5	-.76297E-7	-.18193E-5
	S-a-Ø	.14096E-2	.67498E-2	.79468E-3	.45248E-5	-.95559E-7	-.21307E-5
	S-a-1	.84973E-3	.80817E-2	.81649E-3	.11592E-4	-.24245E-6	-.61405E-5
FIB/COMP	COMBINED	.44114E-3	.34677E-2	.50403E-3	.20647E-5	-.63936E-7	-.17476E-5
	S-a-Ø						
	S-a-1	.80075E-3	.67568E-2	.86450E-3	.47317E-4	-.25023E-5	-.42921E-4
ADDSUB/GATE	COMBINED	.59953E-3	.36954E-2	.40378E-3	.84486E-4	-.38398E-5	-.41539E-4
	S-a-Ø	.16319E-2	.71298E-2	.79192E-3	.42279E-3	-.28793E-4	-.19350E-3
	S-a-1	.91413E-3	.77717E-2	.81514E-3	.46365E-4	-.13664E-5	-.24488E-4
ADDSUB/COMP	COMBINED	.49296E-3	.35619E-2	.54633E-3	.51528E-5	-.18170E-6	-.41501E-5
	S-a-Ø	.12593E-2	.63014E-2	.11841E-2	.67686E-5	-.25128E-6	-.52836E-5
	S-a-1	.76671E-3	.81346E-2	.98431E-3	.16280E-4	-.54722E-6	-.13542E-4

TABLE 16 ERROR COVARIANCE MATRIX ELEMENTS FOR URN MODEL ESTIMATES

TEST		$\sigma^{(PP)}$	$\sigma^{(aa)}$	$\sigma^{(PoPo)}$	$\sigma^{(Pa)}$	$\sigma^{(PPo)}$	$\sigma^{(aPo)}$
FETSTO/GATE	COMBINED	2193.2	544.74	4198.7	-84.059	28.486	78.984
	S-a-Ø	949.65	303.89	2162.2	-24.438	7.7870	27.287
	S-a-1	1272.7	241.53	2041.0	-74.940	27.266	59.115
FETSTO/COMP	COMBINED	1559.5	343.23	1738.6	-39.068	12.896	35.441
	S-a-Ø	646.59	175.28	799.62	-68.002	26.880	65.588
	S-a-1	898.52	175.13	948.18	- 2.1108	.50641	1.9035
FIB/GATE	COMBINED	1835.8	270.70	2465.5	- 1.8078	.33723	1.2139
	S-a-Ø	709.41	148.15	1258.4	- .47553	.084030	.39718
	S-a-1	1176.9	123.74	1224.8	- 1.6878	.33677	.93009
FIB/COMP	COMBINED	2266.8	288.38	1984.0	- 1.3496	.28287	.99971
	S-a-Ø	1012.7	0.0	884.27	0.0	0.0	0.0
	S-a-1	1249.4	148.11	1157.1	- 8.7288	3.1829	7.3280
ADDSUB/GATE	COMBINED	1673.4	271.79	2479.6	-38.124	11.992	27.598
	S-a-Ø	622.48	143.34	1271.5	-36.540	13.704	33.694
	S-a-1	1094.3	128.72	1226.9	-6.5232	1.6383	3.8560
ADDSUB/COMP	COMBINED	2028.6	280.75	1830.4	-2.9338	.65239	2.1317
	S-a-Ø	794.12	158.70	844.50	- .85287	.16471	.70792
	S-a-1	1304.3	122.94	1016.0	-2.6093	.68922	1.6900

TABLE 17 INVERSE ERROR COVARIANCE MATRIX ELEMENTS FOR URN MODEL ESTIMATES

		$m=\sum_{i=j}^{9} m_i$	m_j	$\sum_{i=j}^{8} m_i$	$\sum_{i=j}^{8} i m_i$	A	B	C	D
FETSO/GATE	COMBINED	1000	299	383	564	-407	491	-181	97
	S-a-0	503	136	183	277	-235	282	-94	47
	S-a-1	497	163	200	287	-172	209	-87	50
FETSO/COMP	COMBINED	400	205	258	366	-263	316	-108	55
	S-a-Ø	197	92	118	184	-116	142	-66	40
	S-a-1	203	113	140	182	-147	174	-42	15
FIB/GATE	COMBINED	600	210	251	311	-227	268	-60	19
	S-a-Ø	296	90	112	143	-123	145	-31	9
	S-a-1	304	120	139	168	-104	123	-29	10
FIB/COMP	COMBINED	400	245	288	349	-240	283	-61	18
	S-a-Ø	197	111	131	151	-120	140	-20	0
	S-a-1	203	134	157	198	-120	143	-41	18
ADDSUB/GATE	COMBINED	600	201	243	329	-208	250	-86	44
	S-a-Ø	296	84	106	157	-103	125	-51	29
	S-a-1	304	117	137	172	-105	125	-35	15
ADDSUB/COMP	COMBINED	400	228	271	336	-236	279	-65	22
	S-a-Ø	197	100	124	159	-133	157	-35	11
	S-a-1	203	128	147	177	-103	122	-30	11

TABLE 18 INTERMEDIATE URN MODEL PARAMETER ESTIMATES

6.0 SUMMARY OF EXPERIMENTS

6.1 Phase I Experiments

From the results of the previous section we observe that

- Most detected faults are detected in the first repetition. Subsequent repetitions do not appreciably increase the proportion of detected faults.
- S-a-1 faults are easier to detect than S-a-0 faults.
- The micromemory (i.e., Partition #5) contains a large proportion of indistinguishable faults.
- Faults in memory units (i.e., Partitions #5, #6) are difficult to detect.
- A large proportion of faults remain undetected after as many as 8 repetitions.
- Component-level faults are easier to detect than gate-level faults.
- The coverage estimates of the Phase I experiments are not corrected for indistinguishable fault content.

Subsequent analysis of undetected faults indicates that the proportion of indistinguishable faults at the gate-level is 23.66% and 5.5% at the component-level. The combined, S-a-1 and S-a-0 coverage estimates should be corrected by dividing the raw coverage by $1-\gamma^*$ where

$1 - \gamma^* = .7633$ for gate-level coverage

$= .945$ for component-level coverage

As an example consider the raw coverage in the FETSTO experiment. The uncorrected data indicates 29.9% detection in the 1st repetition. The corrected coverage is, in fact, 39.17%. The 61.7% undetected is corrected to 49.84%.

The poor detection coverage of the six programs of Phase I is not surprising particularly if one considers that Self-Test, which exercises a much greater mix and quantity of instructions, achieve 86.5% detection (at the gate-level). Table 19 shows the instruction mix and quantity of instructions executed versus coverage for each of the six programs. By contrast, Self-Test exercises almost the entire instruction set of the CPU and executes approximately 2000 instructions in a single pass.

In the present study no attempt was made to evaluate the coverage capability of each instruction of the Phase I experiments. As a consequence, it is difficult to correlate instructions mix and coverage. However, the number of executed instructions was plotted versus coverage for each of the programs for gate-level and component-level faults. These results are given in Figures 11 and 12, respectively. The proportions of undetected faults in the QUAD, SERCOM and LINCON experiments were obtained by extrapolating to 8 repetitions, by the method described in Sections 5.2.4, 5.2.5 and 5.2.6.

Referring to the figures, the proportion of faults detected in the first repetition and the proportion of undetected faults after 8 repetitions are linear functions of the number of executed instruction, at least in the range of values considered. It is unlikely, however, that this trend will continue for very large numbers of executed instructions.

The relatively high coverage of S-a-1 faults can be rationalized as follows. The most significant bits of most arithmetic registers are normally zero. Thus, a S-a-1 fault in one of these bits will be detected whenever the contents of the register are used, whereas, a S-a-0 fault will only be detected when the faulted bit is exercised to its complement.

6.2 Phase II Experiments

From the results of the Phase II (i.e., Self-Test) experiments we observe

- There is a significant difference in coverage of gate-level versus component-level faults, e.g., after disqualifying indistinguishable faults gate-level fault coverage was 86.5% whereas component-level fault coverage was 97.9%.
- There was a large proportion of indistinguishable faults in the gate-level emulation, e.g., 23.7%. The worst offender was the micromemory which yielded 33 indistinguishable faults out of a total of 41 selected.
- Only 48% of all detected faults were detected by an explicit test, i.e., 95 out of 198. 103 faults were detected because the fault resulted in a wild branch, i.e., a jump out of the first test.
- Most of the 241 tests comprising Self-Test were redundant; only 46 tests resulted in a detection.
- Of the 95 faults detected by an explicit test 59 were detected by the first 23 tests.
- This particular Self-Test was designed to exercise an instruction set rather than explicit hardware. An noted in Section 7, this approach results in an inefficient Self-Test since, it turned out, most of the tests exercised the same hardware.

6.3 Urn Model Distributions

From previous studies and results of experiments we make the following observations regarding the Urn Model.

- Despite its simplicity the Urn Model results in good correlation with all of the empirical distributions of the study. This is not surprising considering that the model has 3 degrees-of-freedom available for a best fit, i.e., P, P_0 and a, and the empirical distributions are heavily weighted in the first, second and last latency cells. As indicated in Section 9, other distributions could be conjectured that would yield equally good correlation.

- The generalized Urn Model, defined in Section 9, was not evaluated in the study. However, because faults were identified by partition, it is possible to obtain a rough estimate of g (a) by assuming a constant probability of detection in each partition and estimating the corresponding Urn Model parameter, a, suitably weighted by the failure rate of the partition.

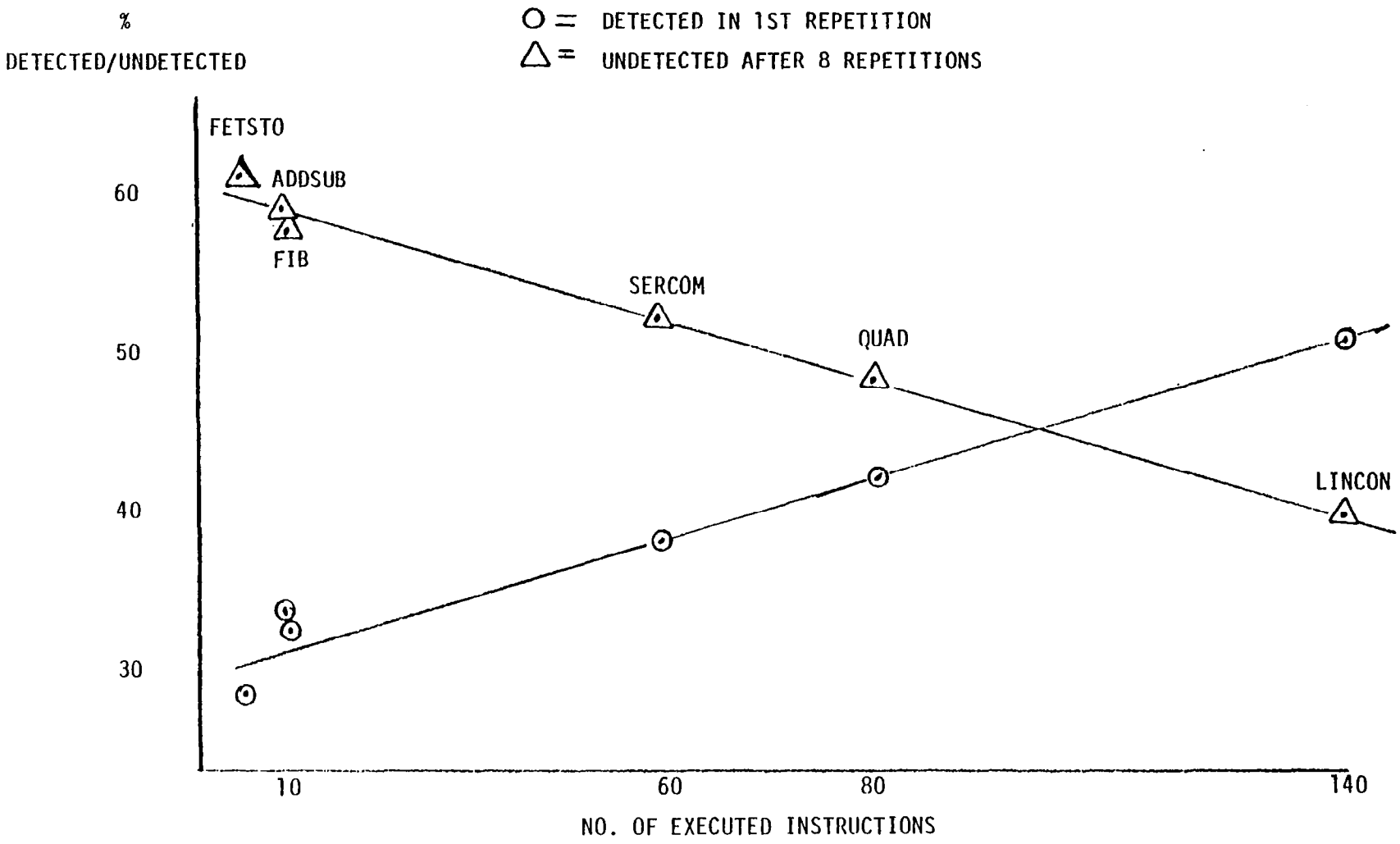


FIGURE 11 DETECTED COMBINED FAULTS VS. NO. OF EXECUTED INSTRUCTIONS
(GATE-LEVEL FAULTS)

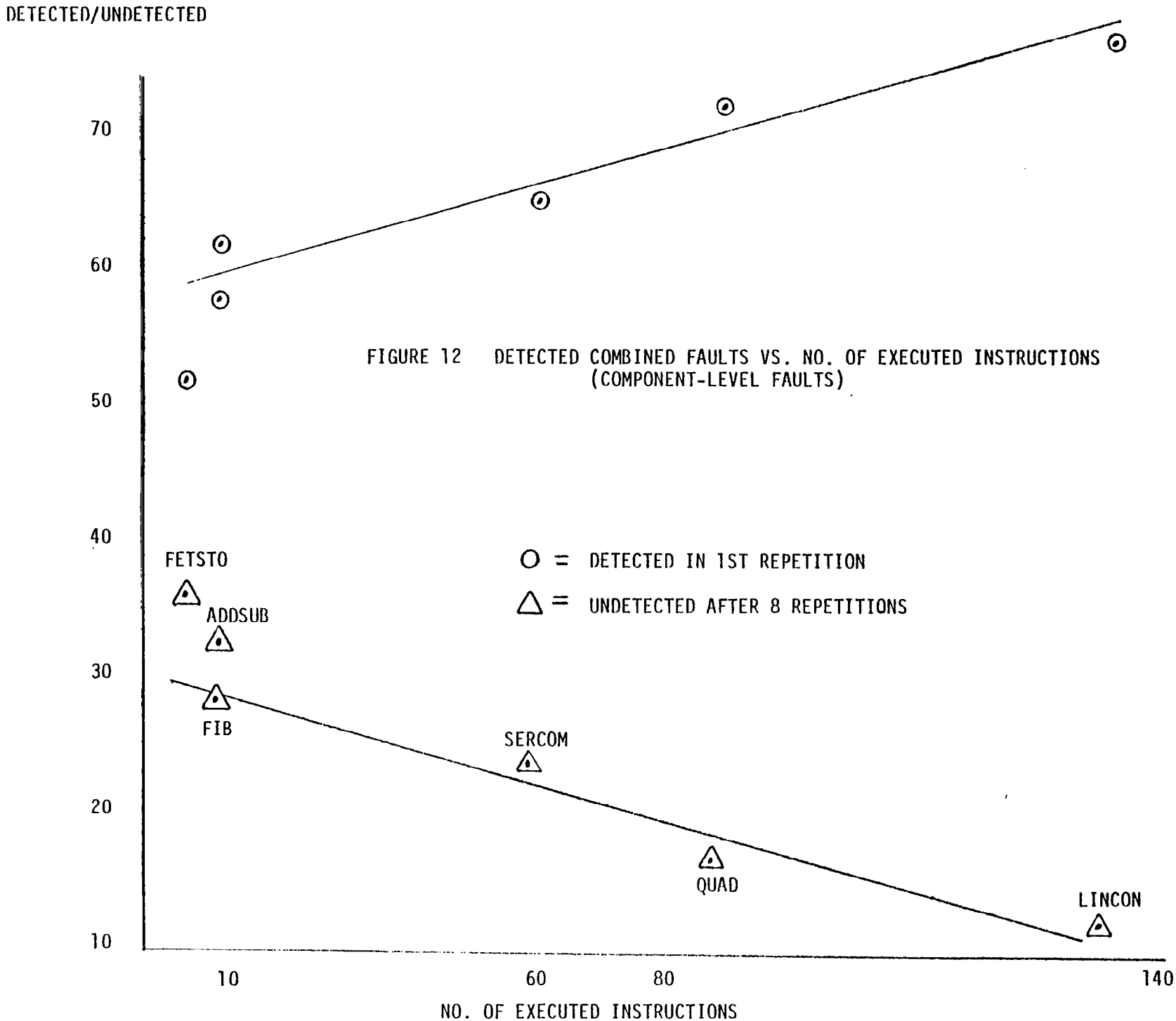


FIGURE 12 DETECTED COMBINED FAULTS VS. NO. OF EXECUTED INSTRUCTIONS (COMPONENT-LEVEL FAULTS)

PROGRAM	TOTAL # OF EXECUTED INSTRUCTIONS	TYPE OF INSTRUCTION					GATE-LEVEL		COMPONENT LEVEL	
		LOAD AND STORE	ADD AND SUBTRACT	BRANCH	TRANSFER	CLEAR	PERCENT DETECTED 1st REPETITION	PERCENT UNDETECTED	PERCENT DETECTED 1st REPETITION	PERCENT UNDETECTED
FETSTO	6	3	1	2	0	0	29.9	61.7	51.3	35.5
ADDSUB	11	4	3	2	2	0	33.5	59.5	57.0	32.3
FIB	11	3	3	4	0	1	35.0	58.2	61.3	28.0
QUAD	87	12	31	38	6	0	43.2	53.3	71.8	23.5
SERCOM	59	12	18	24	5	0	39.5	60.5	64.8	35.3
LINCON	147	76	20	39	11	1	51.7	48.3	76.5	23.5

Note: This table is based upon one pass through the main program.

INSTRUCTION MIX versus DETECTION in PHASE I EXPERIMENTS

Table 19

7.0 ANALYSIS OF UNDETECTED FAULTS

7.1 Phase I Experiments

Because of the large numbers of undetected faults in the Phase I experiments it was not practicable to determine why each undetected fault was not detected. However, from the breakdown of faults by partitions, from the nature of the Phase I programs and from the analysis of undetected faults in the Phase II experiments it is possible to assess, in general terms, why faults were not detected in the Phase I experiments.

7.1.1 Undetected Faults in Phase I

- From the Phase I latency distributions of Section 5 it can be seen that most undetected faults occurred in the micromemory (i.e., Partition #5). This was the result of the limited instruction sets used by the Phase I programs and the large proportion of industinguishable faults contained in the micromemory. As an example, the BDX-930 contains 79 types of macroinstructions but no Phase I program used more than 10. Moreover the Self-Test program, which used almost the entire instruction set, allowed 41 undetected faults of the micromemory of which 33 were indistinguishable. Thus, about 80% of all micromemory faults are indistinguishable.
- The 2901 chips were another prime source of undetected faults, as can be seen from the latency distribution for Partition #4. Excluding indistinguishable faults, these faults are associated with the 2901 RAM which contains the 16 arithmetic registers. The Phase I programs only exercised 2 or 3 of these registers so it is not surprising that faults in the unused registers were undetected.

7.2 Phase II Experiments

Every undetected fault in the Phase II gate-level experiment was analyzed to determine why it was not detected. This turned out to be an exceedingly difficult and tedious task that required an intimate knowledge of the computer and its operation. As an indication of the magnitude of the task, out of the 300 faults injected, 102 were undetected. Out of the 102 undetected faults 71 were identified as indistinguishable. This, however, did not lessen the task since indistinguishable faults were themselves, difficult to identify.

7.2.1 Undetected Faults in Phase II

- The gate-level undetected faults were distributed over the partitions as follows:

PARTITION	DISTINGUISHABLE	INDISTINGUISHABLE	TOTAL
1	4	2	6
2	7	9	16
3	0	16	16
4	9	6	15
5	8	33	41
6	3	5	8
	31	71	102

- The micromemory contained the largest number of indistinguishable faults. This was due to the following:
  - 3 bits of every 56-bit microword were unused.
  - 100 out of 512 microwords were spares.
  - Approximately 10 of 56 bits in a microword are used in executing an instruction; the remaining bits whether faulted or not, are effectively ignored.

  Indistinguishable faults in the micromemory were especially difficult to identify. The problem is that an unused but faulted bit violates the ground rules of the hardware design and the resultant effects are, therefore, unanticipated. To ascertain the effects of such a bit it is necessary to consider all possible scenarios in which the faulted microword is used and track the effects of the faulted bit in each instance. In the present study only the most obvious scenarios were analysed. Even so, it required an engineer with considerable expertise in the hardware design to perform the task.
- In Partition #1 the 4 distinguishable undetected faults were associated with the most significant bits of the memory address register. These were faults that would have prevented access to memory locations not accessed by the Self-Test program.
- In Partition #2 the 7 distinguishable, undetected faults affected the program counter (3), the multiplexer that selected either the program counter or the temporary register (2), and the generation of I/O strobes (2).

As with the memory address register, the faults affecting the program counter would have prevented the counter from addressing memory not accessed by the Self-Test program. The multiplexer faults caused the temporary register to be selected instead of the program counter. The contents of these registers are identical during the Self-Test program.

- In Partition #3 there were no undetected, distinguishable faults. However, all faults that affected the power-on sequence and were not detected, were designated as indistinguishable. Other indistinguishable faults affected the unused carry bits of registers.
- In Partition #4 the 9 distinguishable, undetected faults affected the upper arithmetic registers (7) and the register carry bits (2). The Self-Test program did not sufficiently exercise the bit patterns in the upper arithmetic registers.
- In Partition #5 the 8 distinguishable, undetected faults were the result of not sufficiently exercising the microinstruction set. Because of the very large number of variations a thorough test of the micromemory is extremely difficult to achieve. In the future, processors should incorporate a micromemory sum check for this purpose.
- In Partition #6 the 3 distinguishable, undetected faults affected the "jump relative" instructions which were not exercised by the Self-Test program. Again, it is extremely difficult to exercise these instructions in every possible variation.

7.3 Gate-Level versus Component-Level Faults

A detailed analysis of the undetected failures indicated that component-level faults (i.e., faults at the device pins) are relatively easy to detect. In fact, with a little extra care in the Self-Test design there would have been no undetected component-level faults. It should be noted, in this regard, that no effort was made to modify the Self-Test on the basis of trial runs; the initial Self-Test program remained unchanged throughout the study.

The ease of fault detection at the component-level is not surprising when one considers that a single pin is used in a variety of operations and consequently, will affect many diverse operations when faulted. Two examples are (1) the output bits of a microword and (2) the 2901 arithmetic register data outputs. A fault of a microword pin-out will affect that bit position in every microword. Such a fault will surely be detected whenever a microinstruction uses the complementary value of the faulted bit. In the case of the arithmetic registers, it is not possible to fail a bit of one of the 16 registers without, at the same time, failing this bit in all arithmetic registers. Thus, if the faulted bit is suitably exercised by at least one arithmetic register the fault will be detected.

It may be conjectured that, by injecting faults into the approximately 1200 pins of the BDX-930, an efficient self-test could be designed to achieve 100% detection of component-level faults.

The major obstacle to detection of gate-level faults is their data-dependence. A faulted gate node may not manifest itself at a pin-out unless it is exercised by an appropriate combination of input and internal state. In particular, the fault may not show-up during a test because the test did not create the correct conditions. We note, also that such faults are exceedingly difficult to analyze.

8.0 UNIPROCESSOR BIT

In the present study faults were limited to the central processor unit of the avionics processor. In practice, however, fault analysis must include the entire processor. In this section methods of detecting faults which occur elsewhere in the processor are examined.

An avionics processor customarily contains a built-in-test (Bit) procedure for fault detection and identification of faulted components. A typical Bit procedure consists of applying stimuli to the processor circuitry and determining whether the resultant responses at designated test points are correct. A Bit procedure which utilizes only the resources of a single processor is referred to as a "Uniprocessor Bit".

There are typically three types of uniprocessor Bit: Bit to detect faults 1) prior to take-off, 2) for maintenance purposes and 3) inflight. The differences are mainly in the coverage and isolation requirements, time available to complete the test and the inclusion or exclusion of other subsystems such as sensors and actuators. In the present discussion we will consider only inflight Bit although the methodology is general and applies more or less verbatim to all types of Bit.

Control System Scenario

In this scenario the purpose of inflight Bit is to eliminate a channel of redundancy. The configuration is triplex consisting of three identical processors with dedicated sensors.

The system is designed to maximize survivability subject to the constraint of three channels. This is achieved by taking full advantage of the inherent self-detection capability of each processor. When a processor detects a failure of itself it will either disengage itself from the affected axis but otherwise continue all other control functions and computations or, if it is a computational failure which requires data from the other channels for detection, will select the correct data and use it in all other control computations. This strategy localizes the effects of a failure and allows the processor to perform those remaining control functions and computations that are unaffected by the failure. If the processor cannot detect its own failure and take correct action then the other processors will cause the errant processor to be disengaged from all control axes via dedicated failure logic.

If maximum survivability is to be obtained it is essential not only to detect but isolate a second failure to the failed processor. The self-detection capability of each processor insures that a significant proportion of second failures may be detected and isolated to the offending processor without the need for comparison-monitoring. For those faults that require comparison-monitoring for their detection, isolation is achieved by executing an on-line

self-test program in each of the contending processors. This application of self-test is exactly the same as described in Section 4.6.

Survivability Benefits of Inflight Bit

Inflight Bit is comprised of tests which are 1) conducted as part of normal inflight redundancy management, 2) callable by the software program or 3) initiated by processor interrupt. Let us assume that the first failure resulted in complete loss of the affected processor. Actually, this is a conservative assumption since the fault might have been isolated to the affected axis by the faulty processor. Upon the occurrence of a second fault in one of the remaining processors, which cannot be isolated by the errant processor by normal redundancy management procedures or by a callable subroutine, then the non-faulted, and perhaps even the faulted processor, call for the initiation of an interrupt. In this event both processors are interrupted (subsequent interrupts are, thereafter, inhibited) to execute an on-line self-test which includes the cpu and portions of critical memory. During execution of this self-test all control law computations are suspended. If the faulty processor detects its own fault then it is disengaged from the system and the non-faulted processor assumes complete control. Inability of the faulty processor to detect its own fault is assumed to result in loss of control.

If λ = failure rate of a processor

T = duration of a flight

$1 - \alpha$ = second failure coverage

then

1) $3 (\lambda T)^2 \alpha$ = probability of loss of control with inflight bit,

2) $3 (\lambda T)^2$ = probability of loss of control without inflight bit

Comparing (1) and (2), it can be seen that inflight Bit improves survivability by the factor $1/\alpha$. In many control applications this is sufficient to justify the elimination of a redundant processor.

Scope of Inflight Bit

An examination of the inflight Bit scenario indicates that the procedures for isolating the second failure rely entirely on the resources of a single processor. The differences between inflight Bit and conventional preflight Bit are:

1) Inflight Bit is initiated by a miscomparison whereas conventional Bit is initiated by an external command.

2) Inflight Bit is severely constrained by time: it must detect the fault before it adversely affects the aircraft and in such a way that the processor's ability to control the aircraft is not significantly diminished. As a consequence, inflight Bit does not exercise sensors or actuators or other aircraft subsystems.

As an indication of the scope of inflight Bit the critical components of a typical avionics processor are identified in Table 20 along with their corresponding failure rates*. It is noted that only loss of critical components affects survivability. Thus, the failure rate of (1) and (2) refer to critical components, only. In the target processor of Table 20 critical components comprise about 82% of the total processor.

Referring to Table 20 it can be seen that the cpu comprises 10.76% of the critical components. If

$1 - \alpha_1$ = coverage of the cpu

and $1 - \alpha_2$ = coverage of all other critical components

then the total coverage of critical components is

3) $1 - \alpha = 0.1076\ (1 - \alpha_1) + 0.8924\ (1 - \alpha_2)$.

Fault Detection Procedures of Inflight Bit

A detailed discussion of the fault detection procedures of inflight Bit is beyond the scope of this study. The treatment will be limited to a general survey. Table 21 indicates the principal tests used to detect faults in selected components. These tests include:

1) CPU Self-Test

The reader is already familiar with this test.

2) Watchdog Timer

The watchdog timer is a frequency sensitive circuit which "times-out" unless it is updated by a toggle bit at a fixed frequency. The toggle bit alternates between logic 0 and logic 1 and is supplied by the software program. One of its uses is to detect variations of the real-time clock which exceed a specified threshold. The principal use, however, is to detect a jump out of the program caused either by a hardware fault or a software error, either of which prevents update of the toggle.

* From MIL-HDBK 217B, Notice 2.

3) Parity

The program and scratchpad memories contain an extra bit in each word for parity. The parity is checked, by hardware, after every memory read. A failure of parity results either in an interrupt or the setting of a flag.

4) Memory Sum

The "read-only" memories are subdivided into 1K blocks and the sum of each block is precomputed and stored. Their sums can be checked periodically or as required.

5) RAM Addressing

These procedures test column addressing, row addressing and block addressing for each 1K block of RAM. The tests completely check the row and column decoders within each block of RAM.

6) Wrap-Arounds (Analog Signals)

The analog outputs at the sample and holds and the valve drive amplifiers are fed-back as analog inputs and checked against the digital commands.

7) Bias Inputs to Multiplexers

Each analog input multiplexer contains a bias input which is settable by software. The bias inputs are located at selected pins in such a way that a faulty address will input a signal other than a bias or at least one multiplexer.

8) Reconfiguration

When a critical sensor miscompares and it cannot be isolated to a portion of the I/O circuitry both processors call for a reconfigured control law which does not use the affected sensor.

Inflight Bit Design Methodology and Validation

The methodology of the Bit design is as follows:

1) Identify potentially critical components based upon

- anticipated function of the device
- projected failure modes and effects

If the criticality of a device is doubtful, assume it is critical.

2) Identify the failure modes of each critical device at the device-level.

3) Identify the effects of each failure mode and assess its criticality.

4) Associate a probability of occurrence with each critical failure mode. Disqualify non-critical faults and reduce the failure rate of the device accordingly.

5) For each critical failure mode establish a failure detection procedure based upon the anticipated failure effects. This procedure may consist of a combination of software and additional hardware (e.g., the addition of wrap-arounds).

6) Estimate the level of coverage for each critical device and for the total critical system.

In following this procedure the designer should emphasize components with relatively high failure rates; otherwise a disproportionate effort could be placed on detecting faults with small probabilities of occurrence.

The validation of Bit coverage consists, essentially, in an independent assessment of steps (1) through (6). This assessment is made difficult by several factors:

- The number of possible responses of a digital circuit is large.
- The detection of most faults is dependent on the operating system software; the analyst must be familiar with the failure detection procedures which are implemented by that software.

These difficulties were overcome, at least for the cpu, by emulation using the actual self-test software. The difficulties remain, however, for the rest of the system which constitutes approximatly 90% of the total critical hardware.

TABLE 20

FAILURE RATES OF CRITICAL COMPONENTS

COMPONENT	FAILURE RATE ($x10^{-6}$)/HR.
CPU	42.94
Real Time Clock	2.81
Interrupt Logic	4.54
Program Memory	9.71
Scratchpad Memory	15.62
Memory Mapped Discretes	1.44
Memory Parity	7.82
I/O Controller (Sequencer & File Memory)	11.052
Intercomputer Data Links	29.592
AC/DC Inputs	20.697
Input Multiplexers & Ampl.	8.58
AD Converter	1.92
Input Discretes	9.11
DA Converter	5.08
Valve Drive Amplifiers	8.04
Sample & Hold Circuits	3.59
Output Discretes	15.53
Failure Logic	19.63
Power Supply	21.86
Misc.	0.22
CPU PC Board/Connector	34.4
Memory PC Board/Connector	33.15
Analog I/O PC Board/Connector	23.2
I/O Controller PC Board/Connector	19.81
Servo Ampl. PC Board/Connector	26.53
Harness Assy	22.15
Total Critical	399.021
Total Components	487.19

$$\frac{\text{Critical}}{\text{Total}} = 0.82$$

TABLE 21

INFLIGHT BIT TEST PROCEDURES

	COMPONENT	METHOD OF FAILURE DETECTION
1)	CPU	1) CPU Self-Test
2)	Real Time Clock	2) Watchdog Timer
3)	Program Memory	3) Parity & Memory Sum Test
4)	Scratchpad Memory	4) Parity Redundant Memory Addressing and Bit Pattern Tests
5)	DA, AD Converters Sample and Hold Circuits Valve Drive Amplifiers	5) Wrap-arounds
6)	Input Multiplexers	6) Bias Inputs, Patterns of Faults
7)	Discrete Inputs, Outputs	7) Wrap-arounds Patterns of Faults
8a)	I/O Controller Sequencer	8a) Patterns of Faults
8b)	I/O Controller File Memory	8b) Parity Tests Memory Sum Test
9)	Intercomputer Data Links	9) Not detectable
10)	AC, DC Inputs	10) Reasonableness and Reconfiguration
11)	PC Boards and Connectors	11) Detection dependent on affected devices
12)	Power Supply	12) Level Detectors

9.0 URN MODEL

9.1 Urn Model Description

Several models have been investigated in an attempt to characterize the dynamics of fault propagation in a digital computer. Although simplistic in their assumptions, these models may, nevertheless, provide insight into this undoubtedly complex process. It has been conjectured (ref. 1) that the distribution of latency can be modelled by analogy with balls in an urn. We prefer to employ a different analogy although the resultant distributions are the same.

We postulate that the computer can be subdivided into three sets of mutually exclusive components C_1, C_2, C_3 such that

C_1 = Set of components randomly exercised by the program

C_2 = Set of components continually exercised by the program

C_3 = Set of components never exercised by the program.

We make the further assumption that a fault is detected if and only if the faulted component is exercised. The scenario is that of an avionics computer executing two software programs one of which is executed full-time and the other, part-time. The components that are exercised by the full-time mode are denoted by C_2 and those exercised by the part-time mode by C_1. Neither the full-time or part-time modes exercise components, C_3.

We assume that the part-time mode is exercised randomly. If the unit of time is a repetition of the full-time program then we postulate that the excitation is poisson-distributed in time with a = probability that the part-time mode is exercised in a repetition of the full-time program.

Let λ_1 = Failure rate of C_1 (Failures/hour)

λ_2 = Failure rate of C_2 (Failures/hour)

λ_3 = Failure rate of C_3 (Failures/hour)

$\lambda = \lambda_1 = \lambda_2 + \lambda_3$ (Failures/hour)

We now derive the latency distribution given that a fault has just occurred. The distribution is defined in terms of three parameters, a, P and Q_0 where

P = Probability that the fault is detected in the first repetition given that it occurred in sets C_1 or C_2

Q_0 = Probability that the fault is never detected.

It is easy to derive the following relationships:

$$1)\quad P_0 = 1 - Q_0 = \frac{\lambda_1}{\lambda} + \frac{\lambda_2}{\lambda}\,,\quad Q_0 = \frac{\lambda_3}{\lambda}$$

$$2)\quad P = \frac{\frac{\lambda_2}{\lambda} + a\,\frac{\lambda_1}{\lambda}}{\frac{\lambda_2}{\lambda} + \frac{\lambda_1}{\lambda}} = \frac{\frac{\lambda_2}{\lambda} + a\,\frac{\lambda_1}{\lambda}}{P_0}\,.$$

If

p_k = probability that the fault is detected in the k-th repetition and not detected in a previous repetition, k = 1, 2, 3, ..., n,

q_{n+1} = Probability that the fault is not detected in the previous n repetitions,

then

$$\begin{aligned} p_1 &= P_0\,P = \frac{\lambda_2}{\lambda} + a\,\frac{\lambda_1}{\lambda} \\ p_2 &= (1 - P)\,a\,p_0 = a\,(1 - a)\,\frac{\lambda_1}{\lambda} \\ &\vdots \\ p_n &= (1 - P)\,(1 - a)^{n-2}\,a\,P_0 = a\,(1 - a)^{n-1}\,\frac{\lambda_1}{\lambda}\,,\quad n = 2,3,\ldots \end{aligned} \qquad 3)$$

$$q_{n+1} = Q_0 + \sum_{k=n+1}^{\infty} p_k = Q_0 + (1 - P)\,P_0\,(1 - a)^{n-1}$$

$$= \frac{\lambda_3}{\lambda} + (1 - a)^n\,\frac{\lambda_1}{\lambda}\,,\quad n = 1,2,3,\ldots$$

Observe that

$$q_{n+1} + \sum_{k=1}^{n} p_k = 1, \text{ as expected.}$$

In estimating the above distribution the number of repetitions will be limited to eight. Then, the study will estimate the quantities

$$p_1,\ p_2,\ \ldots.,\ p_8,\ q_9$$

for S-a-1, S-a-0 and combined faults.

9.2 Generalized Urn Model

One of the deficiencies of the Urn Model is that it assumes that each fault in set C_1 is exercised with the same probability, a. If this is not the case then the distribution cannot be represented by the Urn Model. This can be demonstrated as follows.

Subdivide C_1 into mutually exclusive sets C_{11}, C_{12} with failure rates λ_{11}, λ_{12}, respectively, and such that

a_i = probability that the components of C_{1i} are exercised in a repetition of the full-time program, for i = 1,2.

Naturally $\lambda_{11} + \lambda_{12} = \lambda$.

In this case we easily derive

$$p_1 = \frac{\lambda_2}{\lambda} + a_1 \frac{\lambda_{11}}{\lambda} + a_2 \frac{\lambda_{12}}{\lambda}$$

$$p_2 = a_1 (1 - a_1) \frac{\lambda_{11}}{\lambda} + a_2 (1 - a_2) \frac{\lambda_{12}}{\lambda}$$

4)

$$p_3 = a_1 (1 - a_1)^2 \frac{\lambda_{11}}{\lambda} + a_2 (1 - a_2)^2 \frac{\lambda_{12}}{\lambda}$$

$$\vdots$$

$$p_k = a_1 (1 - a_1)^{k-1} \frac{\lambda_{11}}{\lambda} + a_2 (1 - a_2)^{k-1} \frac{\lambda_{12}}{\lambda} , \quad k = 2,3,4,...$$

$$\vdots$$

But, according to the Urn Model, the distribution should be representable in the form

$$p_k = a (1 - a)^{k-1} \frac{\lambda_1}{\lambda} , \quad k = 2,3,...$$

However, there does not exist a value of "a" such that

$$a (1 - a)^{k-1} \frac{\lambda_1}{\lambda} = a_1 (1 - a_1)^{k-1} \frac{\lambda_{11}}{\lambda} + a_2 (1 - a_2)^{k-1} \frac{\lambda_{12}}{\lambda}$$

for all values of k = 2,3,4,..., unless $a_1 = 0$ or $a_2 = 0$ or $a_1 = a_2$.

In a real processor it must be presumed that faults in set C_1 cannot be characterized by a single probability of detection. It is more likely that the fault set produces a range of values from zero to one. If this is the case then it would appear that the Urn Model is severely restricted in its applicability. This is not to say that the model cannot provide a reasonably good description of fault latency in a real processor. In fact, the results of Section 5 show surprisingly good correlation between the model and the empirical distributions. This correlation is more than a coincidence, as we will now demonstrate by a comparison with a more elaborate and, hopefully, more realistic model.

9.2.1 An Alternate Model

The following derivation is informal and heuristic. No attempt will be made to evaluate the resultant model in the present study. Our intention is to exhibit the characteristics of a more realistic model for comparison with the Urn Model and as a baseline for future studies.

We associate with each fault in set C_1 a probability of detection, "a", where "a" can have any value on the interval $0 \leq a \leq 1$. we postulate the existence of a function, g (a), such that

$$\int_{\alpha}^{\beta} g(a)\, da = \gamma$$

where

γ = probability of occurrence of all faults which yield a value of "a" on the interval $\alpha \leq a \leq \beta$.

We observe that

$$\int_{0}^{1} g(a)\, da = \lambda_1 .$$

In the interests of simplicity we have idealized the processor in that we assume a continuum of faults and an integrable function, g(a). In a real processor the number of faults is finite and g(a) is actually a discrete function.

Now let

$$0 = a_0 \leqq a_1 \leqq a_2 \leqq \ldots \leqq a_n = 1$$

be a partition of the interval $0 \leqq a \leqq 1$

and define

$$da_i = a_i - a_{i-1}$$

$$d\lambda_i = g(a_i)\, da_i,\ i = 1,2,\ldots,n.$$

We note that

$d\lambda_i$ = probability of occurrence of all faults which yield a value of "a" on the interval $a_i \leqq a \leqq a_i + da_i$.

If da_i is sufficiently small we can assume that the faults corresponding to the interval $(a_i, a_i + da_i)$ can be represented by a single probability of occurrence, a_i. As a consequence, the Urn Model describes the latency distribution of faults on these intervals. Thus, the latency distribution over the entire interval is

$$p_1 = \frac{\lambda_2}{\lambda} + \frac{1}{\lambda} \sum a_i d\lambda_i$$

$$p_2 = \frac{1}{\lambda} \sum a_i (1 - a_i)\, d\lambda_i$$

$$\vdots$$

$$p_n = \frac{1}{\lambda} \sum a_i (1 - a_i)^{n-1}\, d\lambda_i \qquad 5)$$

$$q_{n+1} = \frac{\lambda_3}{\lambda} + \sum_{k=n+1}^{\infty} p_k \ .$$

If we replace $d\lambda_i$ by $g(a_i)\,da_i$ and pass to the limit we obtain

$$p_1 = \frac{\lambda_2}{\lambda} + \frac{\lambda_1}{\lambda} \int_0^1 \frac{1}{\lambda_1} a\, g(a)\, da$$

$$p_2 = \frac{\lambda_1}{\lambda} \int_0^1 \frac{1}{\lambda_1} a\,(1 - a)\, g(a)\, da$$

6) $\vdots$

$$p_n = \frac{\lambda_1}{\lambda} \int_0^1 \frac{1}{\lambda_1} a\,(1 - a)^{n-1}\, g(a)\, da,\ n = 2,3,4,\ldots$$

$$q_{n+1} = \frac{\lambda_3}{\lambda} + \sum_{k=n+1}^{\infty} p_k\ .$$

Again, we note that

$$q_{n+1} + \sum_{k=1}^{n} p_k = 1 \text{ and } \int_0^1 \frac{1}{\lambda_1} g(a)\, da = 1.$$

Let us represent (6) using the quantities P and P_0.

As before

7) $P_0 = \frac{\lambda_1}{\lambda} + \frac{\lambda_2}{\lambda}$.

If we define $\bar{a}$ as the average value of "a" defined by

$$\bar{a} = \int_0^1 \frac{1}{\lambda_1} a\, g(a)\, da$$

then

8) $P\,P_0 = \frac{\lambda_2}{\lambda} + \frac{\lambda_1}{\lambda}\,\bar{a}$

Note the similarity with (2). Solving (7) and (8) for $\frac{\lambda_1}{\lambda}$ gives

$$\frac{\lambda_1}{\lambda} = \frac{(1 - P)\, P_0}{1 - \bar{a}} .$$

Substituting this into (6) gives

$$p_1 = P\, P_0$$

$$9) \quad p_2 = \frac{(1 - P)\, P_0}{1 - \bar{a}} \int_0^1 \frac{1}{\lambda_1} a\, (1 - a)\, g(a)\, da$$

$$\vdots$$

$$p_n = \frac{(1 - P)\, P_0}{1 - \bar{a}} \int_0^1 \frac{1}{\lambda_1} a\, (1 - a)^{n-1}\, g(a)\, da.$$

In the Urn Model we had

$$\int_0^1 \frac{1}{\lambda_1} a\, (1 - a)^{n-1}\, g(a)\, da = \bar{a}\, (1 - \bar{a})^{n-1}$$

since g(a) was a delta function at $a = \bar{a}$. Substituting this into (9) and we obtain (3).

Equations (6) along with g(a) define the latency distribution. Regarding this distribution we observe that it is monotonic non-increasing, i.e.,

$$p_n \geqq p_{n+1}, \quad n = 2,3,4,...$$

This follows because

$$a\, (1 - a)^{n-2}\, g(a) \geqq a\, (1 - a)^{n-1}\, g(a).$$

9.2.2 Examples

We illustrate the model with several examples.

Example #1

If g(a) is a delta function of magnitude λ_1, i.e.,

$$g(a) = \lambda_1 \, \delta \, (a - \bar{a})$$

then equations (6) reduce to those of the Urn Model.

Thus,

$$p_1 = \frac{\lambda_2}{\lambda} + \bar{a} \quad \frac{\lambda_1}{\lambda}$$

$$\vdots$$

10) $$p_n = \bar{a} \, (1 - \bar{a})^{n-1} \quad \frac{\lambda_1}{\lambda} \, , \; n = 2,3,4,\ldots$$

$$q_{n+1} = \frac{\lambda_3}{\lambda} + (1 - \bar{a})^n \quad \frac{\lambda_1}{\lambda} \, , \; n = 1,2,3,\ldots$$

Example #2

Let $g(a) = \lambda_1$ for $0 \leqq a \leqq 1$.

Then, from (6),

$$p_1 = \frac{\lambda_2}{\lambda} + \frac{1}{1 \cdot 2} \quad \frac{\lambda_1}{\lambda}$$

11) $$\vdots$$

$$p_n = \frac{1}{n \, (n + 1)} \quad \frac{\lambda_1}{\lambda} \, , \; n = 2,3,4,\ldots$$

$$q_{n+1} = \frac{\lambda_3}{\lambda} + \frac{1}{n + 1} \quad \frac{\lambda_1}{\lambda}, \; n = 1,2,3,\ldots$$

Example #3

Let $g(a) = 2\lambda_1 a$ for $0 \leqq a \leqq 1$.

Then, from (6),

$$p_1 = \frac{\lambda_2}{\lambda} + \frac{4}{1 \cdot 2 \cdot 3} \frac{\lambda_1}{\lambda}$$

$$\vdots$$

12)
$$p_n = \frac{4}{n(n+1)(n+2)} \frac{\lambda_1}{\lambda}, \quad n = 2,3,4,\ldots$$

$$q_{n+1} = \frac{\lambda_3}{\lambda} + \frac{2}{(n+1)(n+2)} \frac{\lambda_1}{\lambda}, \quad n = 1,2,3,\ldots$$

Example #4

Let $g(a) = 2\lambda_1 (1 - a)$, $0 \leqq a \leqq 1$.

Then, from (6),

$$p_1 = \frac{\lambda_2}{\lambda} + \frac{2}{2 \cdot 3} \frac{\lambda_1}{\lambda}$$

$$\vdots$$

13)
$$p_n = \frac{2}{(n+1)(n+2)} \frac{\lambda_1}{\lambda}, \quad n = 2,3,4,\ldots$$

$$q_{n+1} = \frac{\lambda_3}{\lambda} + \frac{2}{n+2} \frac{\lambda_1}{\lambda}, \quad n = 1,2,3,\ldots$$

9.2.3 Comparison of Models

It was stated previously that the correlation between the Urn Model and the empirical distributions was more than a coincidence. We illustrate by assuming that Example #4 depicts the actual processor with

$$\frac{\lambda_1}{\lambda} = .3, \quad \frac{\lambda_2}{\lambda} = .3, \quad \frac{\lambda_3}{\lambda} = .4.$$

In this case

$$p_1 = .4$$
$$p_2 = .05$$
$$p_3 = .03$$
$$p_4 = .02$$
14)
$$p_5 = .0143$$
$$p_6 = .0107$$
$$p_7 = .0083$$
$$p_8 = .0066$$
$$q_9 = .46\ .$$

Now let us fit the Urn Model to this distribution. We choose $\frac{\lambda_1}{\lambda}$, $\frac{\lambda_2}{\lambda}$ and "a" such that the Urn Model agrees with p_1, and p_2 and q_9. The result is easily shown to be

$$\frac{\lambda_1}{\lambda} = .2238,\ \frac{\lambda_2}{\lambda} = .3246,\ \frac{\lambda_3}{\lambda} = .4516,\ a = .337,\ P_0 = .5484,\ P = .73.$$

If these values are substituted into (3) we obtain

$$p_1 = .4$$
$$p_2 = .05$$
$$p_3 = .033$$
$$p_4 = .022$$
15)
$$p_5 = .0146$$
$$p_6 = .0097$$
$$p_7 = .0064$$
$$p_8 = .0042$$
$$q_9 = .46\ .$$

Comparing (14) and (15) shows that the differences between the true and Urn Model distributions are small. Clearly, it would require a very accurate statistical analysis to distinguish between the two distributions. We note, incidentally, that if the Urn Model had been used to estimate λ_1 the error would have been 25.4%.

10.0 STATISTICAL ANALYSES

10.1 Introduction

As indicated previously, the principal objective of the study is to obtain esitmates of fault coverage and fault latency in a typical avionics miniprocessor. Although the statistical experiments were carefully designed to yield high accuracy and confidence for the least cost the estimates should not be taken too literally. The reader is advised to exercise engineering judgement in interpreting the results especially when inferring conclusions that depend upon small differences in the estimates. The reason for caution is the uncertainty in the assumptions underlying the study - assumptions which may, if incorrect or inaccurate, contribute a far greater uncertainty to the results than the statistical analysis would imply.

For the record, the critical assumptions of the study are:

- From the standpoint of failure modes and effects every device can be represented by the manufacturer-supplied gate-level, equivalent circuit.
- Every fault can be represented as either a S-a-0 or S-a-1 at a gate node.
- The failure rate of each device is equally distributed over the gates of the gate-level equivalent circuit.
- The failure rate of each gate is equally distributed over the nodes of the gate.
- Memory failures are exclusively faults of single bits.

The assumption that S-a-0 and S-a-1 faults are equally likely is not critical since the experiments were conducted in such a way the the results can easily be modified to reflect a change in this assumption.

Until additional data becomes available the effects of these assumptions on the estimates cannot be properly assessed. In the interim it must be said that the results only pertain to a conjectured realization of the processor.

The statistical experiments (i.e., the number and distribution of faults) are designed to extract as much information as practicable from each experiment for a given set of faults. Thus, in addition to the principal fault coverage and fault latency estimates, it was considered desirable to obtain these estimates for each of the six partitions of the processor. These would provide a basis for more detailed analysis of the fault detection process, and would identify components according to their failure detection coverage. Such data, even if unused in the present study, would be available for future studies.

10.2 Estimators for Self-Test Coverage

The estimators for x, y and z are

1) $x^* = \frac{m_d}{m}$

2) $y^* = \frac{n_d}{n}$

3) $z^* = \frac{m_d + n_d}{m + n}$

where

x, y, z = probability that a S-a-0, S-a-1, combined fault is detected;

m_d, n_d = number of S-a-0, S-a-1 faults detected;

m,n = number of S-a-0, S-a-1 faults injected.

A more accurate estimate of z can be obtained if stratified sampling is employed. For example, let

a_x = proportion of S-a-0 faults in the fault set of the processor

a_y = proportion of S-a-1 faults in the fault set of the processor

where $a_x + a_y = 1$.

If m and n are selected such that

$m = a_x n$, $n = a_y n$

where

n = total number of faults injected,

then

$z^* = a_x x^* + a_y y^*$

is more accurate than (3) if $x \neq y$. Although stratified sampling was not intentionally employed in the study the actual selection resulted in an almost equal number of S-a-o and S-a-1 faults.(*)

* In the selection process $a_x = a_y = 0.5$, i.e., S-a-0 and S-a-1 faults were equally likely.

10.3 Estimators for Latency

The estimators for x_k, y_k and z_k are

$$x_k^* = \frac{m_k}{m}$$

4) $$y_k^* = \frac{n_k}{n}$$

$$z_k^* = \frac{m_k + n_k}{m + n}, \quad k = 1,2,3,\ldots, 8,$$

where

x_k, y_k, z_k = probability that a S-a-0, S-a-1, combined fault is detected in the k-th repetition;

m_k, n_k = number of S-a-0, S-a-1 faults detected in the k-th repetition.

With some abuse of terminology we define

x_9, y_9, z_9 = probability that a S-a-0, S-a-1, combined fault is <u>not</u> detected in the previous 8 repetitions.

We note that x_9 corresponds to q_9 of Section 9. The estimators for x_9, y_9 and z_9 are

$$x_9^* = \frac{m - m_1 - m_2 - \ldots - m_8}{m} = 1 - x_1^* - x_2^* - \ldots - x_8^*$$

5) $$y_9^* = \frac{n - n_1 - n_2 - \ldots - n_8}{n} = 1 - y_1^* - y_2^* - \ldots - y_8^*$$

$$z_9^* = \frac{m\, x_9^* + n\, y_9^*}{m + n} = 1 - z_1^* - z_2^* - \ldots - z_8^*.$$

10.3.1 Corrections for Indistinguishable Faults

The occupancy probabilities x_k, y_k, z_k assume that indistinguishable faults have been disqualified. As indicated previously, the fault set set will contain a certain proportion of indistinguishable faults, γ. When such faults are present the occupancy probabilities are x'_k, y'_k, z'_k where

$$x'_k = x_k\,(1-\gamma)$$

$$y'_k = y_k\,(1-\gamma)$$

$$z'_k = z_k\,(1-\gamma),\ k = 1,2,\ldots,8$$

6)

$$x'_9 = x_9\,(1-\gamma) + \gamma$$

$$y'_9 = y_9\,(1-\gamma) + \gamma$$

$$z'_9 = z_9\,(1-\gamma) + \gamma,$$

assuming that indistinguishable faults are uniformly distributed over S-a-0 and S-a-1 faults. Since indistinguishable faults were not disqualified in the Phase I experiments the estimates actually obtained are those of x'_k, y'_k and z'_k.

10.4 Estimators for Urn Model Parameters

The method of estimation will be described for S-a-0 latency distributions. With an obvious change in parameters, e.g., m_k, the estimates can be applied to S-a-1 and combined latency distributions, as well.

The method is based on the principal of maximum likelihood. We note that m_k S-a-0 faults are detected in the k-th repetition. Accordingly, we seek Urn Model parameters a, P and P_0 that maximize the likelihood function

$$L = p_1^{m_1}\ p_2^{m_2}\ \cdots\ p_8^{m_8}\ q_9^{m_9}$$

where

$$p_1 = P_0 P$$

$$p_2 = (1 - P)\, a\, P_0$$

$$p_3 = (1 - P)\, a\, P_0 (1 - a)$$

7)

$$\vdots$$

$$p_8 = (1 - P)\, a\, P_0 (1 - a)^6$$

$$q_9 = Q_0 + (1 - P)\, P_0 (1 - a)^7$$

and $m_9 = m - m_1 - m_2 - \ldots - m_8$

(See Section 9.1 for a definition of the Urn Model).

The maximum likelihood estimators for a, P and P_0 are obtained as the solution of

$$\frac{\partial L}{\partial a} = 0, \frac{\partial L}{\partial P} = 0, \frac{\partial L}{\partial P_0} = 0 .$$

It can be shown that the solution to the $\partial L/\partial P_0 = 0$ equation is:

8)
$$P_0^* = \frac{\sum_{i=1}^{8} m_i - m_1 (1 - a^*)^7}{\sum_{i=1}^{9} m_i \left[1 - (1 - a^*)^7\right]} .$$

The solution to the $\partial L/\partial P = 0$ equation is:

9)
$$P^* = \frac{m_i \left[1 - (1 - a^*)^7\right]}{\sum_{i=1}^{8} m_i - m_1 (1 - a^*)^7} .$$

Solving the $\partial L/\partial a = 0$ equation for the quantity $(1 - a^*)$ yields the following equation:

$$A (1 - a^*)^8 + B (1 - a^*)^7 + C (1 - a^*) + D = 0$$

where

$$A = -8 \sum_{i=1}^{8} m_i + \sum_{i=1}^{8} i m_i + 7 m_i$$

$$B = 9 \sum_{i=1}^{8} m_i - \sum_{i=1}^{8} i m_i - 8 m_i$$

$$C = \sum_{i=1}^{8} m_i - \sum_{i=1}^{8} i m_i$$

$$D = -2 \sum_{i=1}^{8} m_i + \sum_{i=1}^{8} i m_i + m_1 .$$

The roots of this equation are determined from a root solving routine, and substituted into (8) and (9) to obtain P_0^* and P^*.

10.5 Accuracy and Confidence of Coverage Estimates

It can be shown (ref. 2) that

10) $E (x^*) = x$, $E (y^*) = y$, $E (z^*) = z$

and

$$E \left((x - x^*)^2 \right) = \frac{x (1 - x)}{m}$$

11) $$E \left((y - y^*)^2 \right) = \frac{y (1 - y)}{n}$$

$$E ((z - z^*)^2) = \frac{z (1 - z)}{N}$$

where

$E (\cdot)$ = expected value of $(\cdot)$.

For m, n and N sufficiently large the estimators x*, y* and z* are approximately Gaussian with means and variances given by (10) and (11), respectively.

The following derivation of accuracy and confidence is general and applies to any quantity, x, estimated by the method of Section 10.2. As before,

x* = estimate of x

m = sample size.

It is well-known (See (ref. 3), for example) that the probability

that x lies between the limits

$$\frac{m}{m + \lambda^2}\left(x^* + \frac{\lambda^2}{2m} \pm \lambda \sqrt{\frac{x^*(1 - x^*)}{m} + \frac{\lambda^2}{4m^2}} \right)$$

or, equivalently, that x* lies between the limits

$$12)\quad x \pm \lambda \sqrt{\frac{x(1 - x)}{m}}$$

is equal to γ, where γ is the area of the standard Gaussian distribution between $-\lambda$ and λ. From (11) we may say that the error in the estimate, x*, is

$$13)\quad \varepsilon = \lambda \sqrt{\frac{x(1 - x)}{m}}$$

with a confidence level of γ.

Equation (13) is an ellipse in x. Table 22 gives a tabulation of $\varepsilon\sqrt{m}$ versus x for a confidence level of γ = .95.

It is often convenient to obtain error estimates that are independent of x. From (13) it can be seen that the maximum error occurs when x = 1/2. Table 23 gives a tabulation of this maximum error versus sample size and confidence level. It is noted that the maximum error can be extremely conservative.

10.6 Accuracy and Confidence of Latency Estimates

In this section we derive accuracy and confidence levels for S-a-0 latency estimates. Again, the results are easily extrapolated to S-a-1 and combined estimates.

It is shown [(ref. 3), pg 214] that

14) $E(x_k^*) = x_k$

15) $E\left((x_k - x_k^*)^2\right) = \frac{x_k(1 - x_k)}{m}$

16) $E\left((x_i - x_i^*)(x_j - x_j^*)\right) = -\frac{x_i x_j}{m}$, $i \neq j$

for $i, j, k = 1, 2, 3, \ldots, 9$.

From (14) and (15) it can be seen that the accuracy and confidence level of a single estimate, x^*_k, is identical to that obtained for the coverage estimate x^* in the previous section.

To obtain a measure of "goodness of fit" for the entire distribution we observe that, for m sufficiently large, the variable

17) $\chi^2 = \sum_{k=1}^{9} \frac{m}{x_k}(x_k - x_k^*)^2$

is distributed in a chi-square distribution with 8 degrees-of-freedom (see Ref (3), page 419).

If $\chi^2_{1-\gamma}$ denotes the $1 - \gamma$ level of χ^2, i.e.,

$$P\left[\chi^2 \leq \chi^2_{1-\gamma}\right] = \gamma$$

then the probability that a point, $(x_1^*, x_2^*, \ldots, x_8^*)$ lies inside the ellipsoid

18) $$\chi^2_{1-\gamma} = \sum_{k=1}^{9} \frac{m}{x_k} (x_k - x_k^*)^2$$

is equal to γ.

In principle, we can obtain error bounds for the estimates, x_k^*, from (18). There are two reasons why we do not do so:

1) A chi-square fit generally requires that $m\, x_k \geq 10$ for all x_k, a condition that is not satisfied for the latency cells k=4,5,6,7,8.

2) The Phase I experiments indicate that the latency distributions are concentrated at the 1st and 9th cells, the other cells contributing less than 10% to the total. Thus, the occupancy probabilities of the 1st and 9th cells are the most significant.

If we group the intermediate cells into a single cell and denote the occupancy probability by p and its estimate by p* then, for m sufficiently large, the variable

19) $$\chi^2_{1-\gamma} = \frac{m}{x_1} (x_1 - x_1^*)^2 + \frac{m}{x_9} (x_9 - x_9^*)^2 + \frac{m}{p} (p - p^*)^2$$

is chi-square distributed with 2 degrees-of-freedom. We note that

$$p = 1 - x_1 - x_9$$

$$p^* = 1 - x_1^* - x_9^* \quad .$$

Equation (11) represents a skewed ellipse in the plane of x_1^*, x_9^*.

We can simplify the error estimates by observing that the ellipse of (19) lies inside of the ellipse

20) $$\chi^2_{1-\gamma} = \frac{m}{x_1} (x_1 - x_1^*)^2 + \frac{m}{x_9} (x_9 - x_9^*)^2 \quad .$$

From (20) we conclude that the errors $(x_1 - x_1^*)$, $(x_9 - x_9^*)$ simultaneously lie on the intervals

21) $-\varepsilon_k \leq (x_k - x_k^*) \leq \varepsilon_k$

where

22) $\varepsilon_k = \sqrt{\frac{\chi^2_{1-\gamma}\, x_k}{m}}$, $k = 1,9$

with a probability not less than γ.

It is interesting to compare these errors if the x_k were tested independently. In this case the errors are

23) $\varepsilon_k = \lambda \sqrt{\frac{x_k\,(1 - x_k)}{m}}$, $k = 1,9$.

If we select $\gamma = .95$ (95% level) then

$\lambda = 1.96$, $\chi^2_{.05} = 5.99$ and (22), (23) reduce to

24)
$$\varepsilon_k = 2.45 \sqrt{x_k}$$
$$\varepsilon_K = 1.96 \sqrt{x_k\,(1 - x_k)} \text{ , respectively.}$$

10.7 Accuracy and Confidence of URN Model Parameter Estimates

Let p_i denote the probability that a fault is detected in the i-th repetition, $i = 1, 2, \ldots, 8$ and q_9 the probability that a fault is not detected in the previous 8 repetitions. Then the multinomial sampling distribution is

25) $f = p_i^{\mu_i}\ p_2^{\mu_2} \ldots p_8^{\mu_3}\ q_9^{\mu_9}$

where the p_i and q_9 are defined as in Section 10.4 and the points $\mu = (\mu_1, \mu_2, \ldots, \mu_9)$ are taken from the set

$(1,0,\ldots,0)$

$(0,1,0,\ldots,0)$

$\vdots$

$(0,0,\ldots,0,1)$.

We note that

$$\sum_{i=1}^{8} p_i + q_9 = 1.$$

In order to obtain error bounds for the Urn Model parameter estimates we invoke a theorem from (Ref. 2), page 212:

Theorem

The maximum likelihood estimators θ_1^*, θ_2^*, θ_3^* for the sampling distribution $f(\mu, \theta_1, \theta_2, \theta_3)$ from samples of size n are, for large samples, approximately distributed by the multivariate Gaussian distribution with means θ_1, θ_2, θ_3 and variances and covariances, σ_{ij}, where $||\sigma_{ij}||$ is the inverse of the matrix whose elements are

$$26) \quad \sigma^{ij} = - n E \left[\frac{\partial^2 \log f}{\partial\theta_i \, \partial\theta_j} \right], \; i, j = 1,2,3.$$

When applied to the Urn Model

$\theta_1 = P$

$\theta_2 = P_0$

$\theta_3 = a$

and θ_1^*, θ_2^*, θ_3^*, are the estimated values of θ_1, θ_2 and θ_3, respectively.

The elements of the covariance matrix $||\sigma_{ij}||$ were tabulated for the Phase I experiments and the results are given in Table 16.

A much simplified estimate of the errors can be obtained by employing an approximation that was suggested in (ref. 1). There, it was assumed that

$$q_9 = 1 - P_0 = Q_0.$$

In other words, detectable faults are always detected in the first 8 repetitions. From (7) this is equivalent to the approximation

27) $$(1 - P)\, P_0\, (1 - a)^7 = 0.$$

If this substitution is made in the likelihood function, L, then the resultant estimates are, for S-a-0 faults,

$$P_0^* = \frac{1}{m} \sum_{i=1}^{8} m_i$$

28)
$$p^* = \frac{m_i}{\sum_{i=1}^{8} m_i}$$

$$a^* = \frac{\sum_{i=1}^{8} m_i - m_1}{\sum_{i=1}^{8} i m_i - \sum_{i=1}^{8} m_i}.$$

The experimental results confirm the accuracy of those approximations* (see Table 15). More interesting, however, are the resultant error covariances. When the approximation of (27) is made we obtain

* At least for the distributions obtained in the study.

$$E((P - P^*)^2) = \frac{P\,(1 - P)}{m\,P_0}$$

29) $$E((P_0 - P_0^*)^2) = \frac{P_0\,(1 - P_0)}{m}$$

$$E((a - a^*)^2) = \frac{a^2\,(1 - a)}{m\,P_0\,(1 - P)}$$

and the cross-covariances vanish. Thus the estimates are independent and, at a confidence level of γ, the errors are, for P, P_0, a, respectively,

$$\varepsilon_P = \lambda\sqrt{\frac{P\,(1 - P)}{m\,P_0}}$$

$$\varepsilon_{P_0} = \lambda\sqrt{\frac{P_0\,(1 - P_0)}{m}}$$

$$\varepsilon_a = \lambda\sqrt{\frac{a^2\,(1 - a)}{m\,P_0\,(1 - P)}}$$

where λ is as defined in Section 10.5. As an indication of the error magnitudes, for a typical fit (See FETSTO results, Table 15),

$P \simeq .781$

$P_0 \simeq .383$

$a \simeq .464$

which result in the following errors at a 95% confidence level:

ε_p = .041 (5.2%)

$\varepsilon_{\bar{p}_0}$ = .030 (7.8%)

ε_a = .073 (15.7%).

The reader is reminded that these estimates are only valid if the Urn Model correctly represents the distribution. The example at the end of Section 9 illustrates the uncertainty in estimated variables when an incorrect model is used.

TABLE 22

Error Ellipse for a Confidence Level of $\gamma = .95$

$$\varepsilon\sqrt{m} = \sqrt{x\,(1 - x)}$$

$\varepsilon\sqrt{m}$	x
0.0	0
.427	.05
.588	.1
.70	.15
.784	.2
.849	.25
.898	.3
.935	.35
.960	.4
.975	.45
.98	.5
.975	.55
.96	.6
.935	.65
.898	.7
.849	.75
.784	.8
.7	.85
.588	.9
.427	.95
0.0	1.0

TABLE 23

MAXIMUM ERROR VERSUS SAMPLE SIZE AND CONFIDENCE LEVEL

CONFIDENCE LEVEL \ SAMPLE SIZE	200	300	400	600	1000
.6	.03	.025	.021	.017	.013
.7	.037	.03	.026	.021	.017
.8	.046	.038	.033	.027	.021
.9	.058	.048	.041	.034	.026
.95	.069	.056	.049	.04	.031

11.0 EMULATION DESCRIPTION

11.1 BDX-930 Architecture

The BDX-930 Digital Processor is a microprogrammed, pipelined machine designed around the AMD2901A four bit microprocessor slice. The machine contains sixteen general purpose registers of which four registers may be loaded directly from memory and two registers may be used as base registers. One register is used as a stack pointer.

The program counter and memory address register are contained in the 9407, a chip designed to perform memory address arithmetic. Along with a temporary register contained on the same chip, the BDX-930 is able to perform four basic addressing modes involving three registers and various instruction fields.

The machine contains three memory interface data registers which are used to input and output memory data. There are also a number of one bit status flag registers that can be manipulated under program control. This includes the F1 and F2 registers, which are hardware flags, and the interrupt enable, overflow status registers. There also exist the indirect and link registers used by the microcode for branching.

The microcode is contained in seven proms and a pipeline register is included for simultaneous microcode fetch and decoding. Various internal and external conditions can affect microcode branching as selected by the microcode itself and a microcode control prom. In addition to a rich instruction set which includes 16 and 32 bit fixed point operations, there is a test set interface in the microcode. A selectable saturate mode is available which limits the results of arithmetic operations when overflow or underflow occur.

For simulation purposes, the computer has been divided into six partitions, consisting of the following principal devices:

Partition 1 - Address Processor

- 4 - 9407 Memory Address Processor Equivalent Circuit
- Selector Chips to Multiplex Memory Address Source
  - 4 - 54LS352 4:1
  - 2 - 54LS158 2:1

Partition 2 - Data and Status Registers

- 2 - 54LS374 Memory Input Buffer Register
- 2 - 54LS374 Memory Output Buffer Register
- 2 - 54LS374 Next Instruction Register
- 3 - 54LS113 Single Bit Registers for
  - overflow
  - indirect addressing
  - link (bit carry for divide)
  - interrupt mode
  - F1 and F2
- 2 - 54LS153 Select Overflow, Link, and Indirect Bit Sources.
- 2 - 54LS245 octal bus transceivers

Partition 3 - Microcontroller

- Pipeline Register
  - 4 - 54LS273 octal latch
  - 4 - 54LS175 quad latch
  - 1 - 54LS374 octal latch with tri-state
- 1 - 54LS273 External Signal Synchronizer
- 3 - 54LS151 Selectors 8:1 for Branch Conditions
- 1 - 54LS169 Counter for Shift and Multiply Instructions
- 1 - 54LS169 Counter for Multiple Register Load-Store Instructions
- 1 - 54LS377 Instruction Register
- 1 - 54LS253 Microcode Branch Selector

Partition 4 - Execute

- 4 - AMD2901A 4 Bit Slice ALU
- 1 - AMD2902 Lookahead Carry
- 2 - 54LS153 Selector 4:1 Register Selectors
- 1 - 54LS253 Selector 4:1 Shift Bit Selector

Partition 5 - Microcode

- 7 - 54S472 Proms with 56 Bit Wide Microcode

Partition 6 - Control Proms

- 1 - 54S472 Prom Microcode Start Address for Macroinstructions
- 1 - 54S288 Prom Control for Microcode Branch

Instruction execution is accomplished by a pipelined architecture; various stages of execution occur simultaneously for a sequence of instructions. Consider, for instance, four instructions, A,B,C,D, to be executed in sequence. During the same clock cycle it is possible for the program counter to be incremented to point to instruction D, while instruction C is being fetched, instruction B is being decoded and instruction A is being executed.

With this level of parellelism, it will be noted that when the execution phase of an instruction is one clock cycle, the average time to perform the entire instruction will be one clock cycle. This relation can better be understood by referring to Table 24.

It should also be noted that the partitioning of the BDX-930 is roughly broken up into the stages of the pipe: - address, fetch, decode, and execute. These stages of the pipe are joined by various buses throughout the CPU. These buses are formed from tri-state logic and some are bidirectional. An enumeration of the major buses includes

- Y - Connects the output of the ALU (AMD2901A) to the address processor and the output register. In addition, it connects the output of the next instruction buffer to the start address register and instruction register.
- D - Connects the memory data register and the program counter to the input of the ALU.
- DAT - Bidirectional bus connecting memory and I/O to the memory data register and output register.
- M - Bidirectional memory data bus
- MAR - Memory Address Bus
- U - Microcode Bus
- IR - Instruction Register

A list of the devices used in the BDX-930 and their failure rates is given in Table 25. The data was obtained from MIL-HDBK127B, Notice 2.

11.2 Description of the Emulator

The emulation includes the components of the CPU (Central Processor Unit), scratchpad memory and those portions of the program memory containing the six target programs and the target self-test program. The emulation is derived from the circuit schematics. Each device is represented by a gate-level equivalent circuit supplied by the chip manufacturer. It was found that six types of gates were sufficient to represent any device, e.g., NAND, AND, OR, NOT, NOR, EXCLUSIVE OR. Table 26 gives the number of equivalent gates in each device of the CPU. In all, 5,100 gates were required. In the interests of reducing execution time, it was not expedient to emulate all components at the gate-level. The following elements are represented at the functional-level:

program memory
scratchpad memory
microprogram and control memories
16 general purpose arithmetic registers.

The emulation did not include the direct memory access unit (DMA) or any of the devices of the I/O. The emulated devices of the CPU are shown in Figure 13.

Faults were injected into all devices except the program and scratchpad memories. Because the program memory is "read-only", no processor, faulted or not, is permitted to write into this memory. However, even though the scratchpad memory is never faulted, a faulty processor can write into it. As a consequence, in the parallel mode of operation where 36 processors are simultaneously emulated, the corresponding 36 scratchpad memories are also emulated.

No delay has been simulated between logic gates. It is assumed that all combinational logic is stable at the output the instant an input pattern is applied to it. This means that each time the input is changed, the network need only be evaluated once to supply the correct output pattern. Operating in this manner is very time efficient, but puts stringent requirements on the order of evaluation of the gates. To be able to meet these requirements, the logic is levelized, i.e., placed in groups or levels that represent the proper order of evaluation.

The emulator utilizes the parallel method of logic simulation (see, for instance, Seshu and Freeman (ref. 5), or Hardie and Suhocki (ref. 6)). The data word of a PDP-10 contains 36 bits; each bit position is used to represent a different machine. The simplest gate operations are represented by a single Boolean instruction; when the two inputs occupy the same bit positions in their respective words, the output also occupies this bit position. The advantage of this technique is execution time savings. Typically, the amount of code necessary to simulate 36 machines is of the same order as the amount of code necessary to simulate only one machine. The BDX-930 description is contained in compiled code, rather than in tables, which was also done for speed.

Certain portions of the machine, notably the memory elements, were represented at a functional level rather than a gate level. For microprogram memory, two words of PDP-10 storage contain 56 bits of microstore; at micro memory fetch time, these bits are retrieved from the proper address for each of the simulated machines and combined to form suitable words to interface the gate portion of the emulation. The ROM portion of main memory is handled in the same manner. Writable store contains a routine to translate the gate inputs into consecutive PDP-10 storage words so that there is one copy of writable storage for each machine being emulated. On reading this storage, the process is reversed.

In a typical run of the emulator, 36 different machines are exercised; 35 faulted machines and one good machine. Each faulted machine is assumed to have a single solid fault at one node, either stuck-at-one (S-a-1) or stuck-at-zero (S-a-0). The faults are injected by defining extra gates at each node, an AND gate for stuck at zero and an OR gate for stuck at one. A typical AND gate using this technique is shown in Figure 14.

To demonstrate the use of this technique for injecting and emulating the propagation of gate-level faults, refer to Figure 15. In the figure, a typical gate-level operation is shown involving four gates. Logic is levelized in terms of two levels. Let us assume that the input has the value '10' on it, and we would like to simulate 6 faults (S-a-0 at leads 3,4, and 6 and S-a-1 at leads 3,4, and 5) in the circuit. The first step would be to define two PDP-10 computer words to represent the inputs at each lead. Bit position 0 would represent the unfaulted machine while positions 1-6 would represent the faulted ones. Next, fault words are defined for S-a-1 and S-a-0 faults at each node and each node is assigned a word to contain the results of its operation. First, the input faults are applied to the input words yielding words (1) and (2). Then, the two buffer operations are performed. Buffer output faults are applied yielding words (3) and (4). Note that (3) and (4) occupy the same physical storage as (1) and (2) yielding a memory efficient algorithm. The second level is evaluated in much the same manner, yielding the results (5) and (6). Table 27 shows the value of the nodes for Figure 15.

An additional reduction in run-time can be achieved by observing that not all gate faults are distinguishable at the gate output. For example, a S-a-0 fault on the input node of an AND gate is indistinguishable from a S-a-1 fault on the output node. As a consequence, if two or more indistinguishable faults of the same gate are selected, only one fault will be emulated.

It will be noted that only one partition of the BDX-930 runs with faults injected in each simulated run. The remaining partitions run 'true value', that is, logic without fault injection capabilities. This results in a time saving in program execution. When the entire emulator is run true-value, the execution ratio between PDP-10 time and simulated time is 21,000:1, with faults injected in one partition, this number is approximately 25,000:1. In order to achieve these ratios, a number of problems had to be solved.

Stabilization

The propagation of logic signals through a combinational logic network involves many concurrent paths of travel; the value at the output of any particular gate is only stable after a certain interval. The inherently sequential execution of a computer program presents a potential problem as to the order of evaluation of gates. One approach to parallel operation in a sequential emulation is, during each BDX-930 clock time, to evaluate the gates repeatedly until re-evaluation produces no further change in state. It is desired to minimize program execution time; therefore the number of times each logic gate is evaluated should be minimized. If a particular sub-circuit is free of memory elements and feedback paths, it need be evaluated only once. The order of analysis here is critical, but not necessarily unique. Feedback elements represent a special problem. For a simple R-S type flip flop, the proper output states can be ascertained by evaluating each element, at most twice.

D-Latches

The edge-triggered D latches represent a much harder circuit to model. The circuit diagram of such a latch is shown in Figure 16.

Operation of these circuits is dependent upon receiving both the low and high levels of the clock signal to trigger the latch. All of the combinational logic in the BDX-930 requires only one evaluation per clock cycle. In the interest of reducing execution time, the D latch in Figure16awas replaced by the latch in Figure16bwhich is functionally equivalent, but requires only one evaluation per clock cycle.

Tri-State Buses

In order to evaluate tri-state buses, it is necessary to replace them with a gate equivalent circuit. Such an equivalent circuit is easy to synthesize in the case of a wired-OR type circuit, but tri-state logic also may fail to a high impedance state. In this case, any change on the line is particularly sluggish. The last failure-free output on a tri-state line will exponentially approach the high state as a function of wiring capacitance and other circuit parameters at the time of the failure. This failure mode was not simulated: tri-state failure modes are considered the same as wired-OR failure modes. The justification is to avoid a failure mode that is random, and dependent on the past history of the gate. The equivalent circuit is shown in Figure 17.

Serial-to-Parallel/Parallel-to-Serial

In a gate-level emulation each node is represented by a single word of the host computer whereas,in a register-level emulation, a collection of nodes is so represented. For example, a set of 16 nodes might represent the address bits of memory data which, at the register-level, would be represented by a single, 16 bit word in the host computer. In an emulation which contains both gate-level and register-level components, it is necessary, in passing from one level to another, to convert the host computer words to compatible formats.

In the present emulation the following memory devices are represented at the register-level:

microprogram	(7, 54S472, 512 x 8 proms)
microprogram control	(54S288, 32 x 8 proms)
macroinstruction start address	(54S472, 512 x 8 proms)
main memory	

The first 3 components are read-only proms which require a conversion of the address nodes to register-level and data to gate-level. The main memory is both ROM and RAM and this requires a conversion of both the address and data nodes to register-level in the write cycle and address nodes to register-level and data to gate-level in the read cycle.

For each of the above elements, two conversions are required for each pass. Because of the quantity of such elements and the frequency of passage it is essential to implement a real-time efficient conversion algorithm.

We assume a parallel fault emulation with each node represented by a 36-bit word of the host computer.

A collection of m nodes is represented by the set of words

$$A_1 = (a_{11}, a_{12}, \ldots, a_{1,36})$$

$$A_2 = (a_{21}, a_{22}, \ldots, a_{2,36})$$

$$A_m = (a_{m1}, a_{m2}, \ldots, a_{m,36})$$

Assume that the corresponding register-level data are represented by the words

$$B_1 = (a_{11}, a_{21}, \ldots, a_{m1}, x, x, \ldots)$$

$$B_2 = (a_{12}, a_{22}, \ldots, a_{m2}, x, x, \ldots)$$

$$B_{36} = (a_{1,36}, a_{2,36} \ldots, a_{m,36}, x, x, \ldots)$$

where the x's represent unused bits, with $m \leq 36$.

The resultant conversion algorithms are shown in Figures 18a,18b.

11.3 Preprocessor and Postprocessor

The host processor for this emulation is a Digital Equipment Corporation model PDP10. This is a 36 bit machine built around 1970 taking about 1.5 to 3 microseconds for an integer add instruction, depending on addressing mode. The processor is located at Carnegie-Mellon University in Pittsburgh, Pennsylvania and was accessed via telephone lines. This machine supports time sharing for university and research projects in which the university participates.

The emulation was written in the BLISS language. BLISS was developed by DEC for its own system programming efforts, and is well suited for our applications. It allows bit manipulation in a very efficient manner while maintaining many of the structures of a higher order language. The macro facility proved invaluable in developing an efficient simulation.

The emulation process consists of running three separate programs, a preprocesor, the emulation proper, and a postprocessor. The function of the preprocessor is to select a random set of faults for emulation while the postprocessor interprets and prints the results.

The preprocessor reads in a list of components in the BDX-930 and their failure rates. The program then queries for the number of faults to be run, and faults are selected in the manner described in Section 3. Faults are broken up into sets for each partition, and further into groups of 35 or less for each emulator run. All pertinent information is written out onto disk for processing by the emulation phase.

The emulation phase runs the BDX-930 emulator program repetitively until all groups of 35 faults have been processed. After each simulated run, the results are written out to disk for later processing.

The postprocessor takes the results of the emulator runs and prints a table. It is first decided whether or not the fault was detected, and if so, during which iteration or step of the test program detection occurred. The exact location of the fault is determined, and all this information is displayed for the information of the analyst. Cumulative statistics are also computed.

The true-value emulator was subsequently verified by single stepping through a self-test program consisting of 2000 executable instructions. The self-test was designed to exercise every instruction type in the instruction repertoire of the BDX-930. This is essentially the same procedure used to validate the hardware version of the processor.

11.4 Typical Circuit Representations

Some typical representations of components are shown in Figures 19 thru 22. Each of these diagrams represents a single integrated circuit chip, which is coded in BLISS as a subroutine. In turn each partition consists of subroutine calls that simulate a particular function of the CPU.

11.5 Summary of Emulation Characteristics

A summary of emulation characteristics is given in Table 28.

TABLE 24 PARALLEL OPERATION OF THE BDX-930 PROCESSOR

BDX - 930 INSTRUCTION FLOW DIAGRAM

MICRO CYCLE				1	2	3	4	5	6	7	8
CALCULATE INSTRUCTION ADDRESS	A	B	C	D			E	F			G
FETCH	Z	A	B	C			D	E			F
DECODE	Y	Z	A	B			C	D			E
EXECUTE	X	Y	Z	A	B	B	B	C	D	D	D
EFFECTIVE EXECUTION TIME				A	B			C	D		

XYZ - PREVIOUS INSTRUCTIONS
E, F, G = SUBSEQUENT INSTRUCTIONS

❺ CONCURRENT CALCULATE INSTRUCTION ADDRESS, FETCH, DECODE AND EXECUTE OF INSTRUCTIONS

❽ 8 MICRO CYLES - TOTAL EFFECTIVE EXECUTION TIME

TABLE 25 COMPONENTS OF THE BDX-930 CPU

DEVICE	FAILURE RATE/PER UNIT (PPMH)
9407	1.3431
2901A	2.1656
2902	0.3898
5440	0.0653
54125	0.0855
54S00	0.0855
54S04	0.1003
54S10	0.0764
54S20	0.0654
54S32	0.2138
54S288 (32x8 prom)	0.1787
54S472 (512x8 proms)	1.009
54LS00	0.084
54LS02	0.084
54LS04	0.0983
54LS08	0.0752
54LS11	0.084
54LS86	0.084
54LS113	0.1447
54LS151	0.1483
54LS153	0.1447
54LS158	0.1410
54LS169	0.6603
54LS175	0.1703
54LS245	0.3792
54LS253	0.1447
54LS273	0.1636
54LS352	0.6882
54LS367	0.2681
54LS374	0.3117
54LS377	0.1100
	0.7234
	0.7148

TABLE 26

MICROCIRCUITS AND EQUIVALENT GATE COUNT

DEVICE	EQUIVALENT GATES
2901A	798
2902	19
54113	8
54151	17
54153	16
54158	15
54169	58
54175	22
54245	18
54253	16
54273	34
54352	16
54374	26
54377	35
9407	143

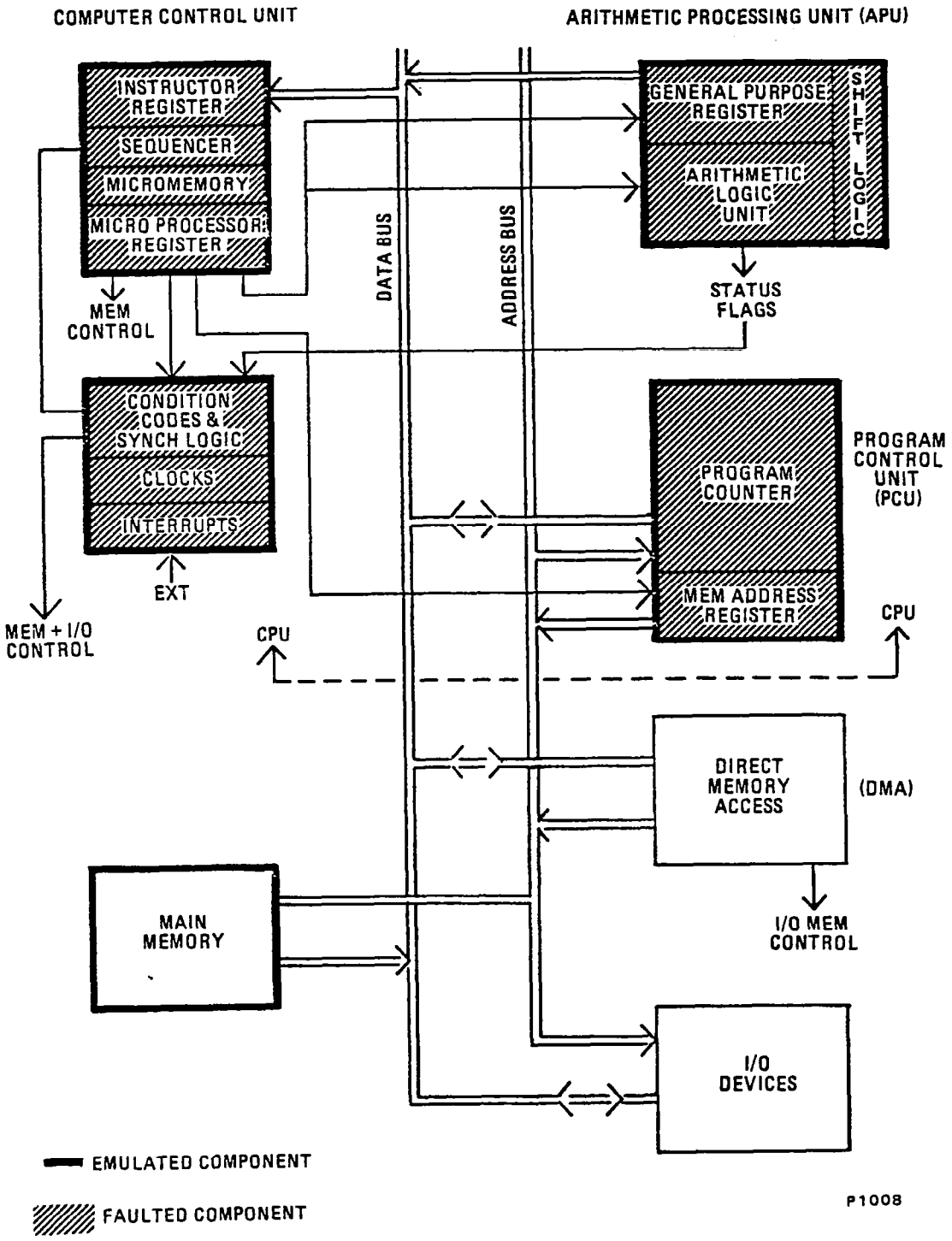


FIGURE 13. PROCESSOR ARCHITECTURE

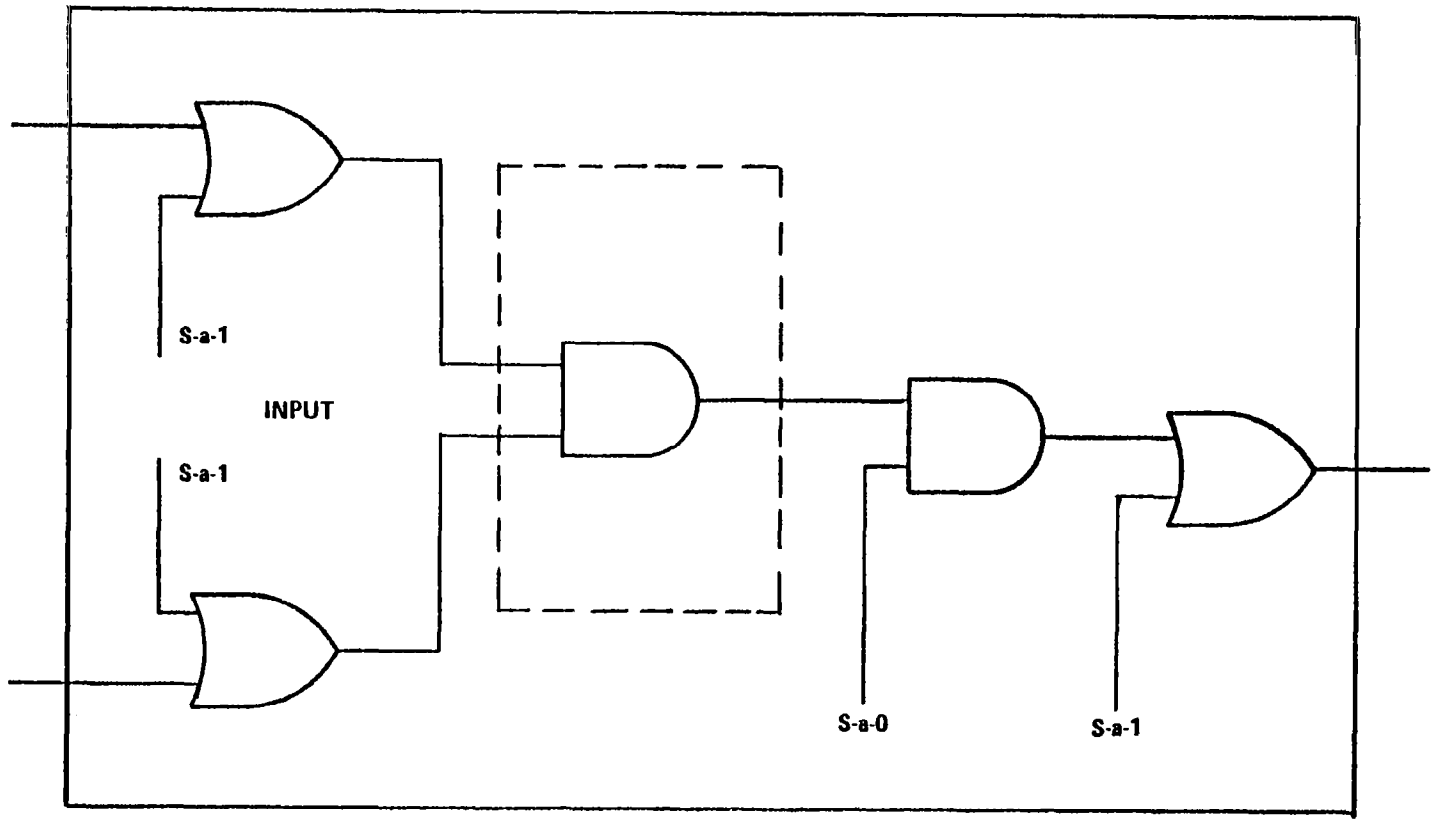


FIGURE 14 BASIC TWO INPUT AND GATE FAULT MODEL

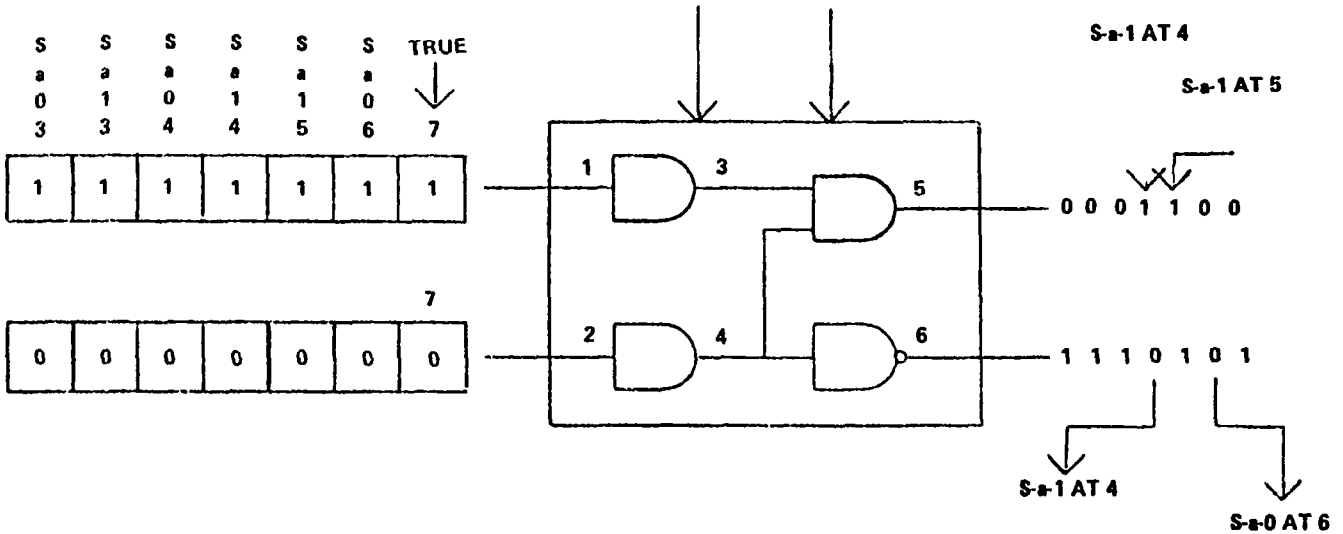
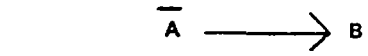


FIGURE 15 TYPICAL GATE-LEVEL COMPUTATION

TABLE 27 VALUE OF NODES



	S a 0 3	S a 1 3	S a 0 4	S a 1 4	S a 1 5	S a 0 6	T R U E
(1)	1	1	1	1	1	1	1
(2)	0	0	0	0	0	0	0
(3)	0	1	1	1	1	1	1
(4)	0	0	0	1	0	0	0
(5)	0	0	0	1	1	0	0
(6)	1	1	1	0	1	0	1

3=1 AND 1
4 = 2 AND 2
5 = 3 AND 4
6 = NOT 4

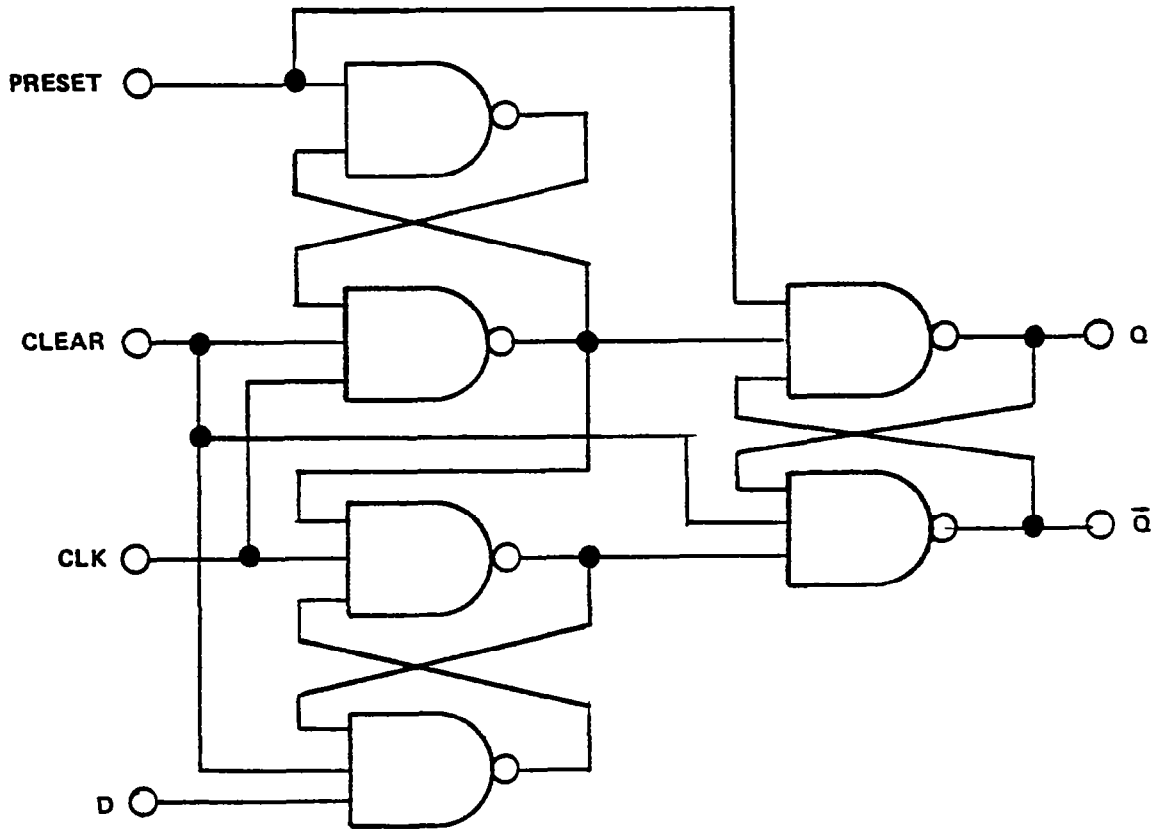


FIGURE 16a GATE-LEVEL D-LATCH MODEL

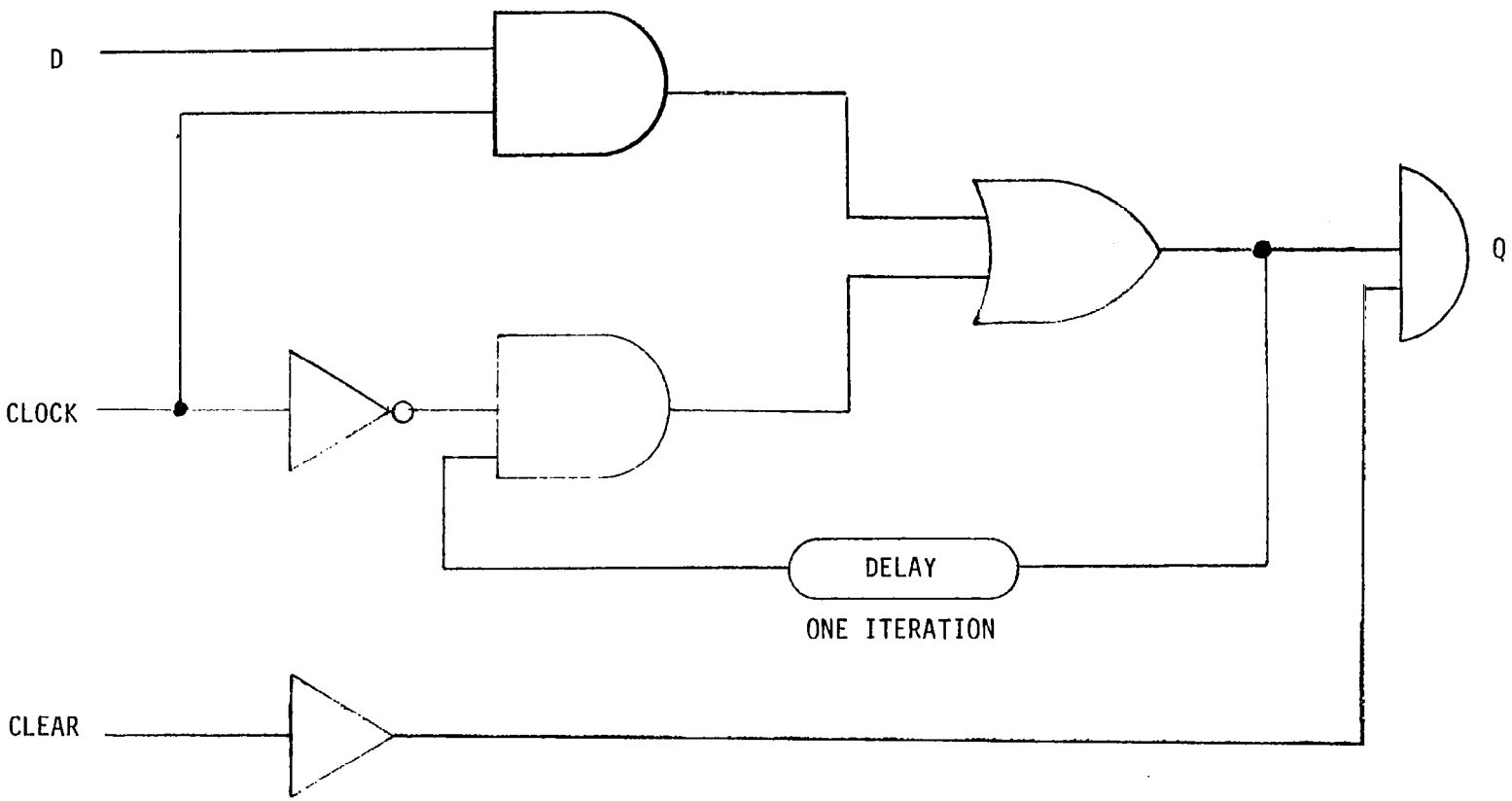


FIGURE 16b TYPICAL FUNCTIONAL D LATCH (THIS CIRCUITRY IS NOT FAULTED)

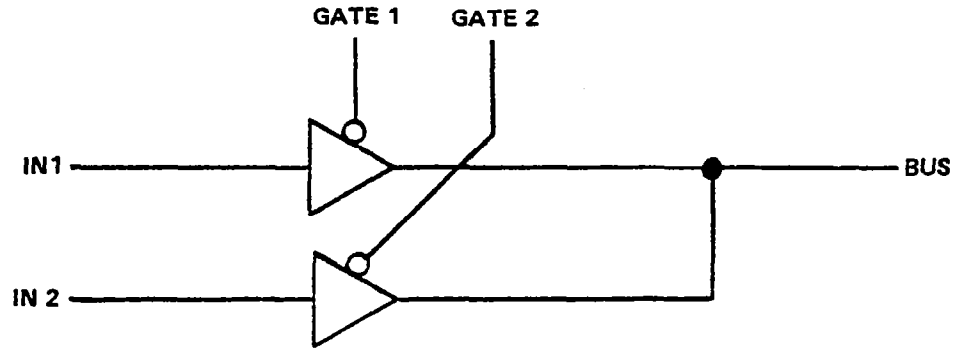


FIGURE 17a TRI-STATE BUS

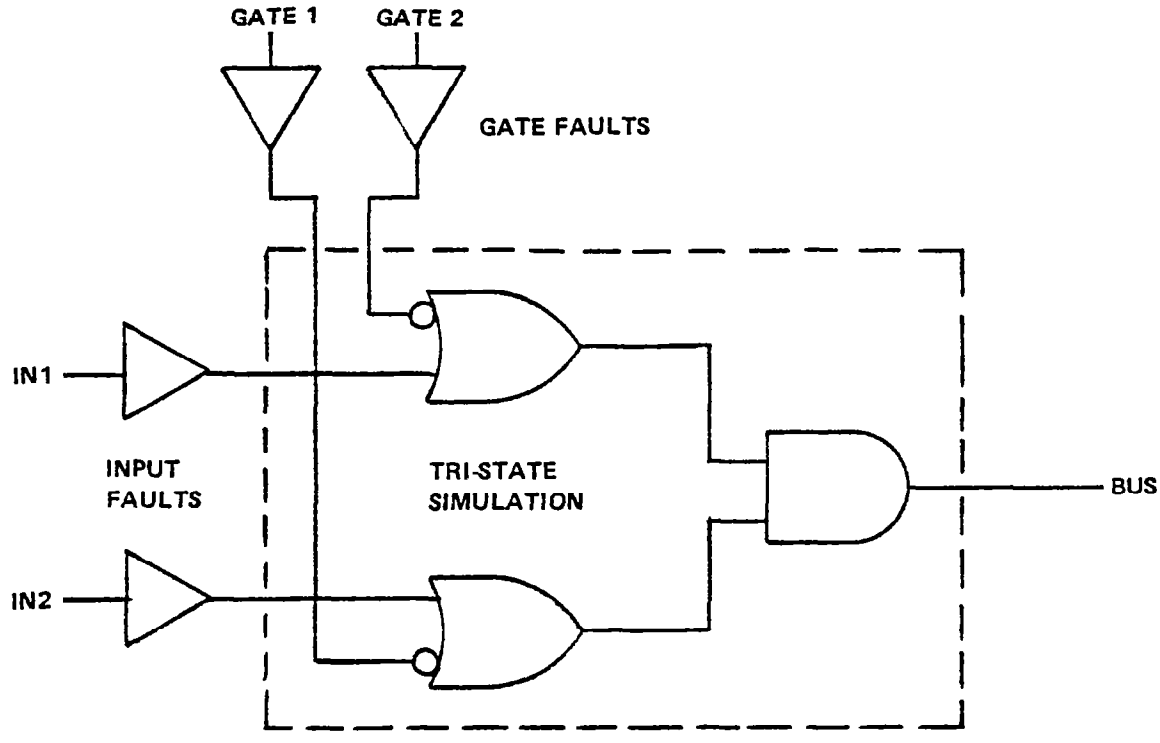


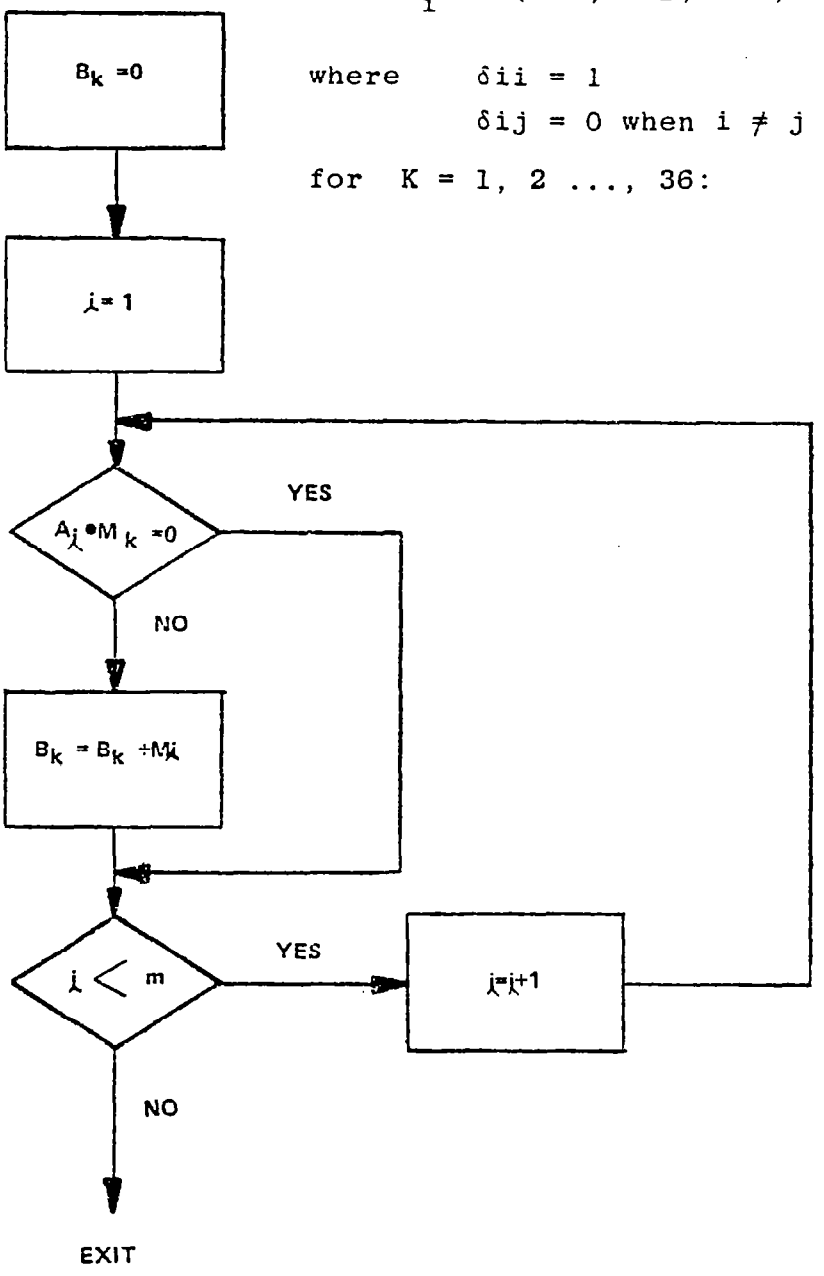
FIGURE 17b SIMULATION OF TRI-STATE BUS

Set M_i = ($\delta i1$, $\delta i2$, ..., $\delta i,36$) , i=1,2, ..., m

where $\delta ii = 1$

$\delta ij = 0$ when $i \neq j$

for K = 1, 2 ..., 36:



NOTE: "." = logical "AND", "+" = logical "OR"

FIGURE 18a GATE-LEVEL TO REGISTER-LEVEL CONVERSION ALGORITHM

For i = 1,2, ..., m:

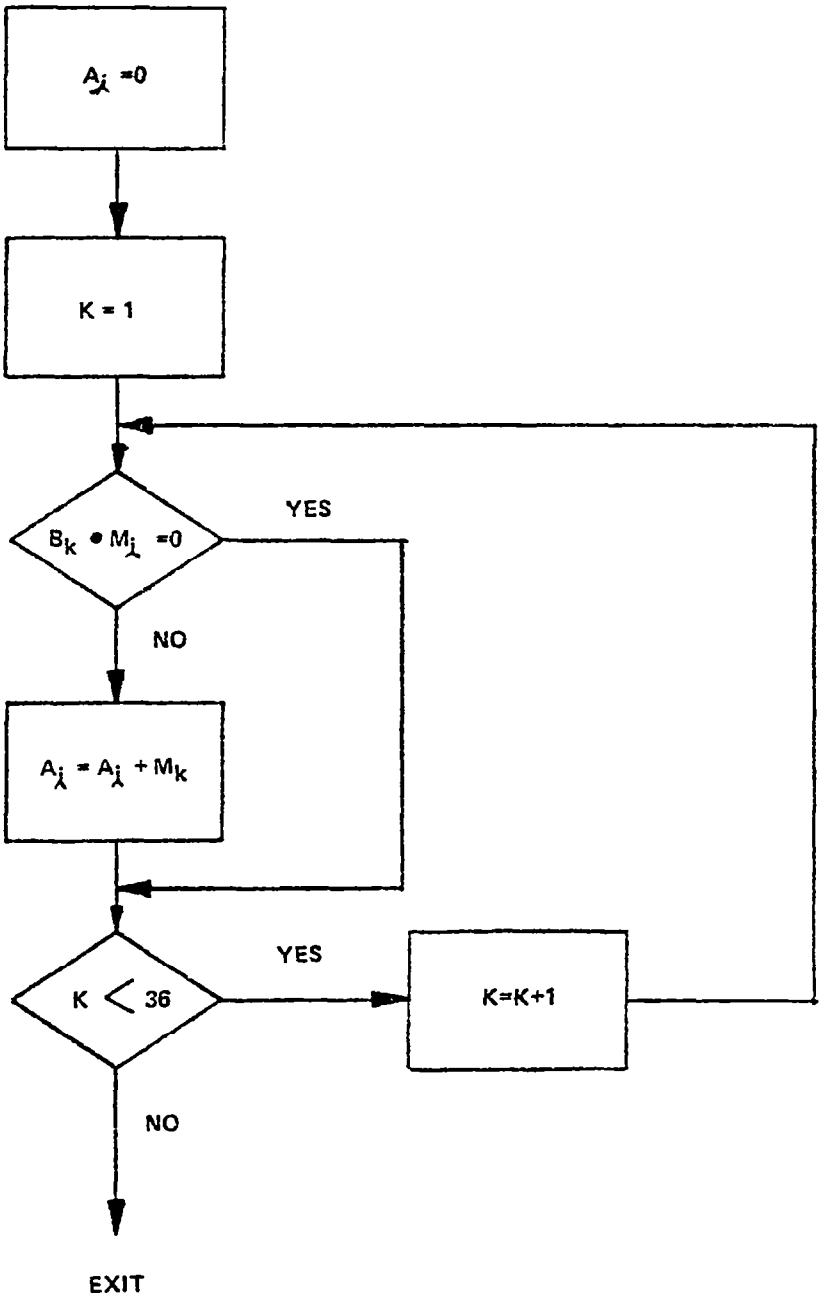


FIGURE 18b REGISTER-LEVEL TO GATE-LEVEL CONVERSION ALGORITHM

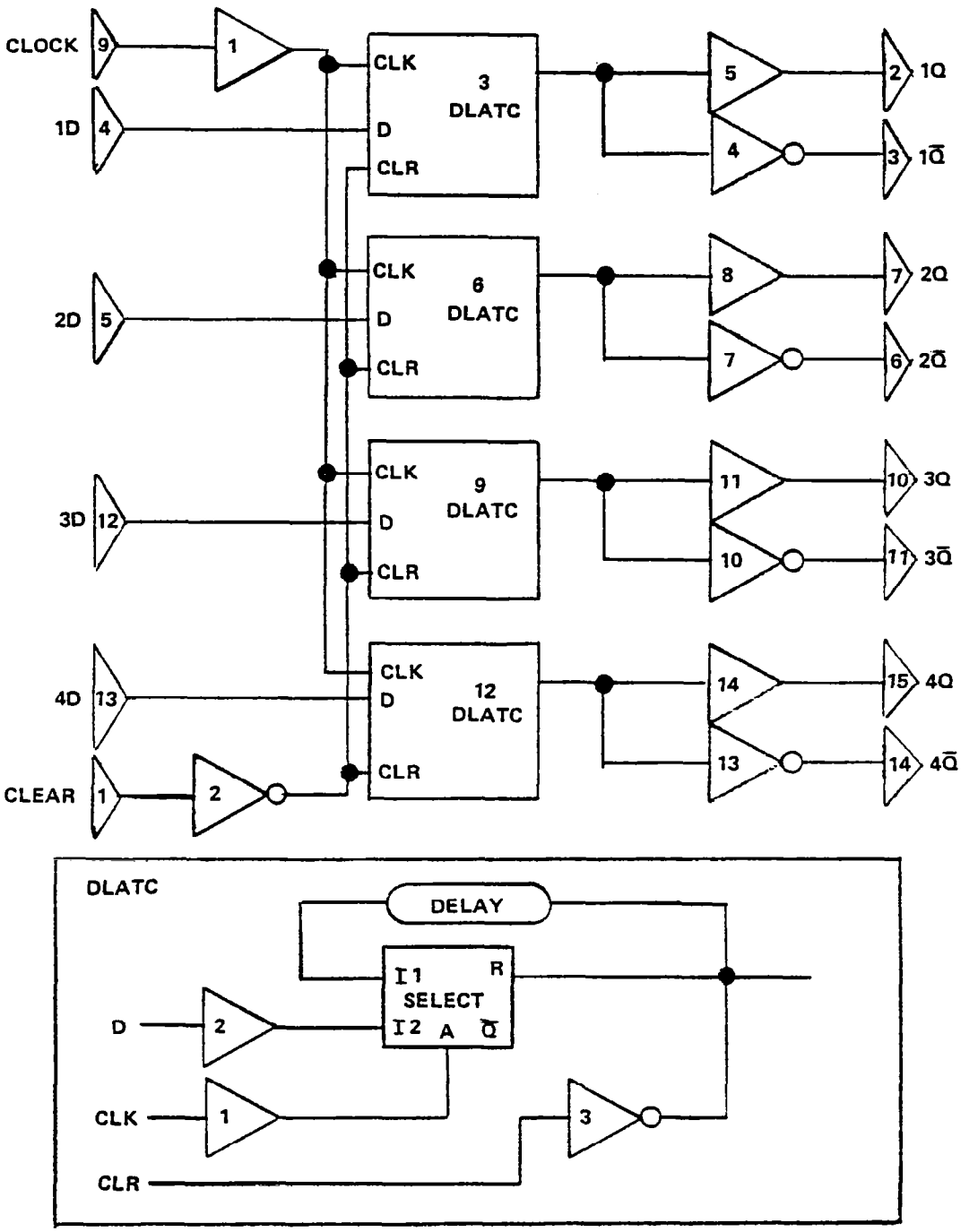


FIGURE 19 IC 175

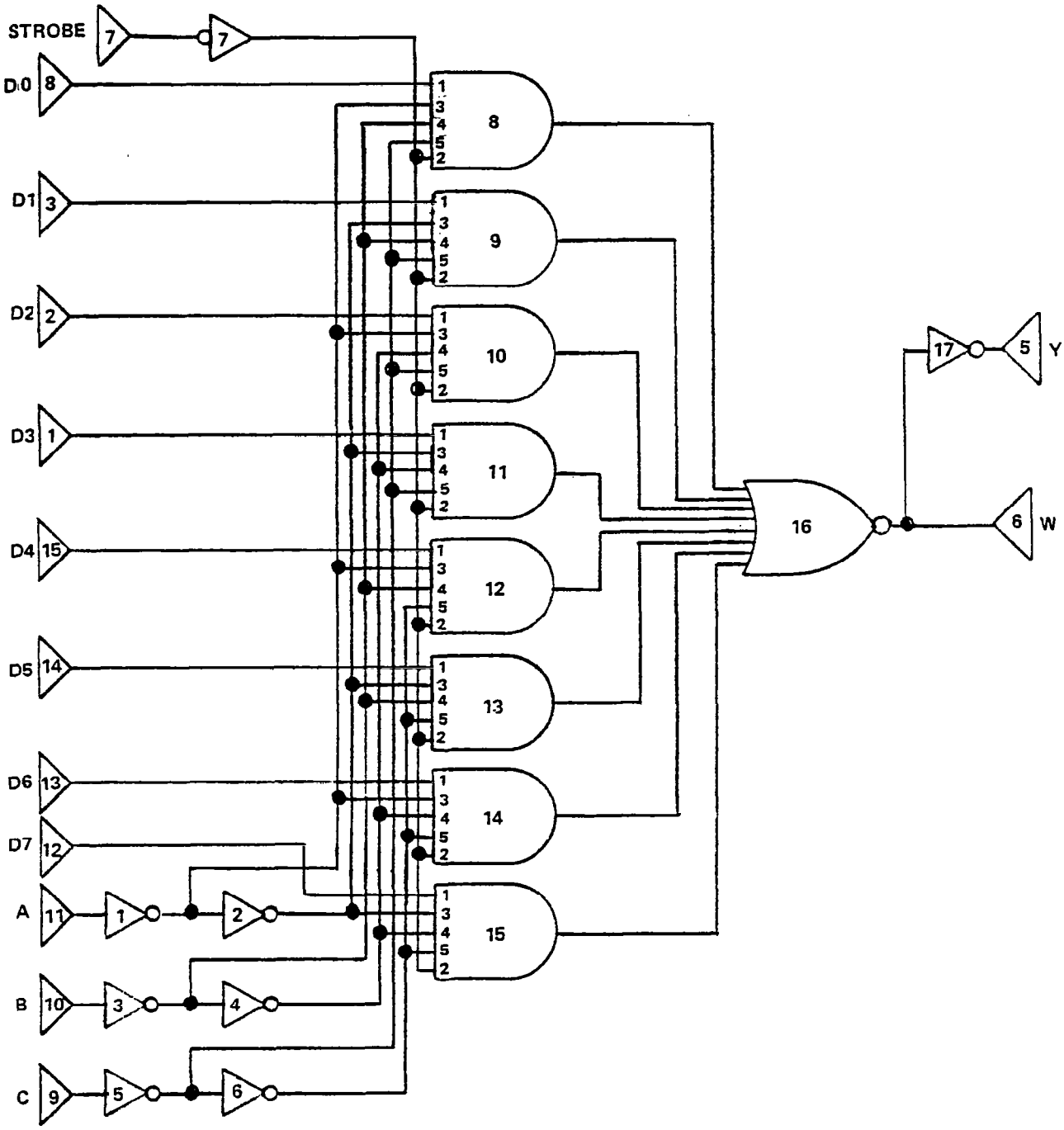


FIGURE 20 IC 151

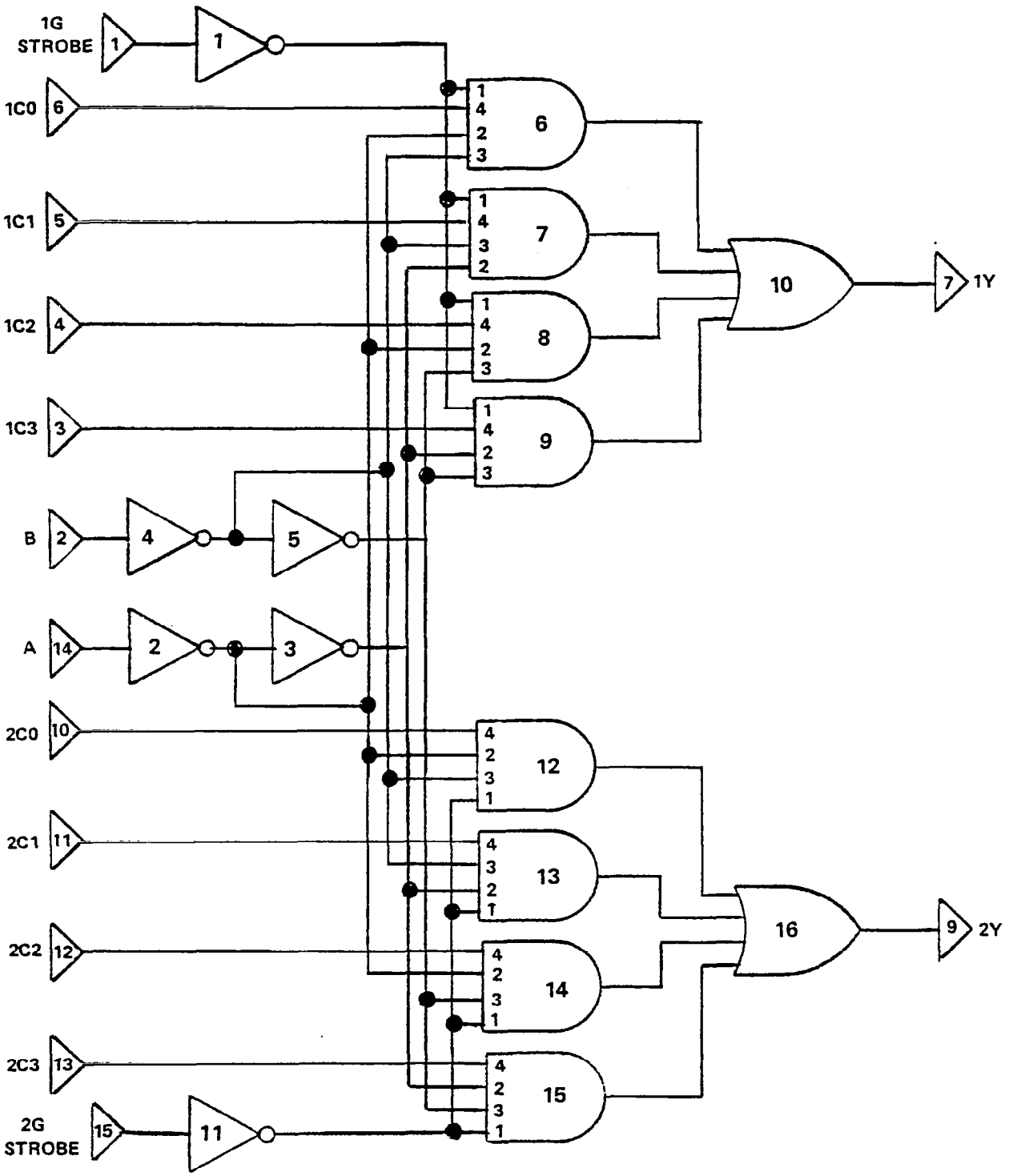


FIGURE 21 IC 153

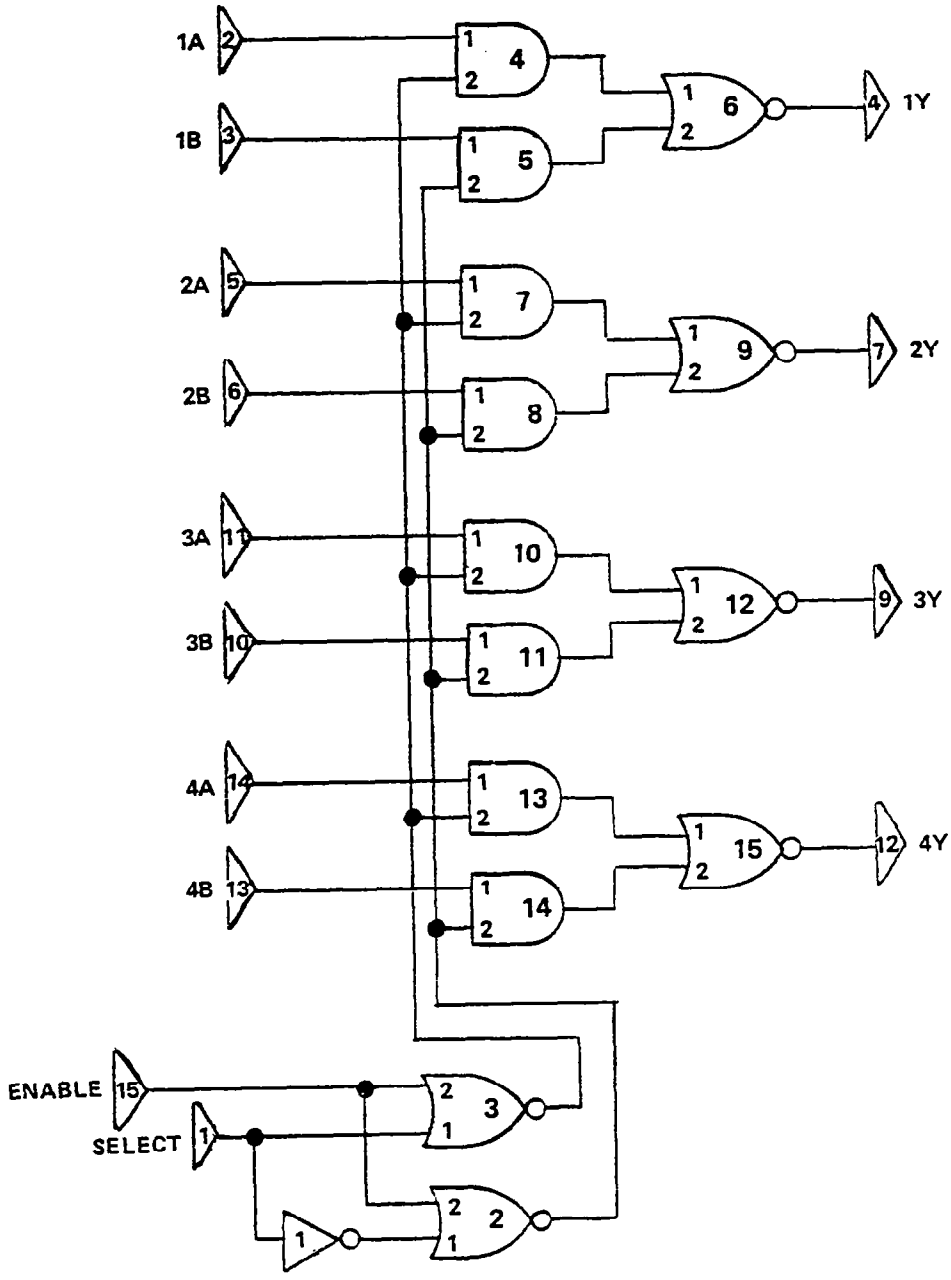


FIGURE 22 IC 158

TABLE 28 EMULATOR CHARACTERISTICS

GATE-LEVEL EMULATOR

- CODED IN BLISS
- BDX-930 CPU @ GATE LEVEL
  (A FORTIORI, AT COMPONENT-LEVEL)
- MICROPROGRAM MEMORY PROMS (7, 512x8)
- MICROPORGRAM CONTROL PROM (1, 32x8)
- MACROINSTRUCTION START ADDRESS PROM (1, 512x8)
- MAIN PROGRAM MEMORY (ROM) @ REGISTER-LEVEL
- SCRATCHPAD MEMORY (RAM) @ REGISTER-LEVEL
- 36 CPU'S EMULATED IN PARALLEL
- 36 SCRATCHPAD MEMORIES EMULATED IN PARALLEL
- 1 MAIN MEMORY EMULATED AND SHARED BY 36 COPIES
- 5100 GATES @ 4 NODES/GATE, AVERAGE
- 33,024 BITS OF PROM @ 1 NODE/BIT
- S-A-0, S-A-1 FAULTS @ EACH GATE NODE OR BIT
- NO FAULTS IN SCRATCHPAD MEMORIES
- NO FAULTS IN MAIN MEMORY
- 67,256 FAULTS POSSIBLE
- TRUE-VALUE REAL-TIME RATIO = 21,000:1
- REAL-TIME RATIO WITH FAULTS = 25,000:1
  (PDP-10, CMUD COMPUTER)

12.0 EXTENSION OF EMULATOR TO MULTIPROCESSOR SYSTEMS

One of the objectives of the present program was to study and make recommendations on how the emulation could be utilized to perform fault injection experiments on the SIFT (Software Implemented Fault Tolerance) computer system which was developed by SRI International with Bendix, Flight Systems Division, as a major subcontractor.

12.1 Description of SIFT (See (ref. 4))

SIFT is an ultra-reliable computer system that is designed for flight critical aircraft control and avionics applications. It is based on a multiprocessor architecture that achieves fault tolerance by replicating computing tasks among processing units. Error detection and system reconfiguration are performed by software. The SIFT system is shown in Figure 23 in a 7-processor configuration. A single processor is shown in Figure 24.

Initially, each SIFT processor is assigned a set of software tasks. If a task is critical it will be redundantly executed by either three or five processors, depending upon the criticality of the task. Each processor executing a critical task inputs sensor data over a dedicated 1553A bus. This data is stored in memory and transmitted to the other processors over a high speed, serial, intercomputer data link which operates in a broadcast mode. By means of a selection algorithm each processor of the redundant set selects the same inputs, computes its assigned task and transmits the results to the other processors over the intercomputer data link.

The results of each computation are compared in the local processors and any discrepancies are noted. When a faulted processor has been identified the processor is, thereafter, "ignored" by the other processors. The critical tasks are then redistributed amoung the remaining processors.

Fault isolation and reconfiguration are the functions of the Global Executive task which, because of its criticality, is also redundantly computed. The Global Executive reads the error reports from the local processors and attempts to identify the faulted processor. When the processor is identified the Global Executive computes a new distribution of tasks and informs the remaining processors of their new assignments.

12.2 SIFT Emulator

The above description of SIFT was given to orient the reader in the SIFT philosophy and to note that the fault isolation and reconfiguration tasks may be performed by processors other than those computing the application tasks. As we shall see presently, implementing this feature will result in an increase in run-time of the emulator.

In order to emulate the SIFT system it is necessary to extend the present emulation to include the I/O interface hardware, shown in Figure 24, e.g.,

- Transaction and data files
- 1553A controller
- Broadcast sequencer
- Receiver sequencer
- Broadcast bus

The parallel mode of operation of the present emulator automatically accommodates a multiprocessor system. Instead of using 36 bits to represent 36 versions of the same processor the bits are subdivided into sets of 7, 7, 7, 7, 7, 1 with each 7-bit set representing a single version of the SIFT system. The last bit is extraneous. Faults would then be injected into one of the 7 bits of each segment and the emulation would be run, exactly as in the present study. If faults are limited to a single processor and its program memory then it is only required to emulate the 6 memories of the non- faulted processors and the 5 memories of the faulted processors.

The Preprocessor and Post processor programs would have to be modified to reflect the new rules of fault injection and fault identification.

In the SIFT emulation only 5 faults can be emulated in a single run as compared with 35 in the present study. This reduction was the result of emulating the processor in groups of 7. An apparently attractive alternative approach would be to emulate the 6, non-faulted processors and 30, faulted processors in a single run. The problem here is that the action of both the local and global processors may be different for different faults. As a consequence, it is necessary to emulate the entire SIFT system for each fault. The effect of emulating 5 instead of 35 faults in a single run is a 7-fold increase in run-time of the emulator.

12.3 SIFT Fault Injection Experiments

Having established the essential features of the SIFT emulator we now consider some possible applications.

Experiment #1

Inject a fault and determine the time required to detect and isolate the fault and reconfigure the system. The data will consist of 3 latency distributions for time of detection, isolation and reconfiguration.

Experiment #2

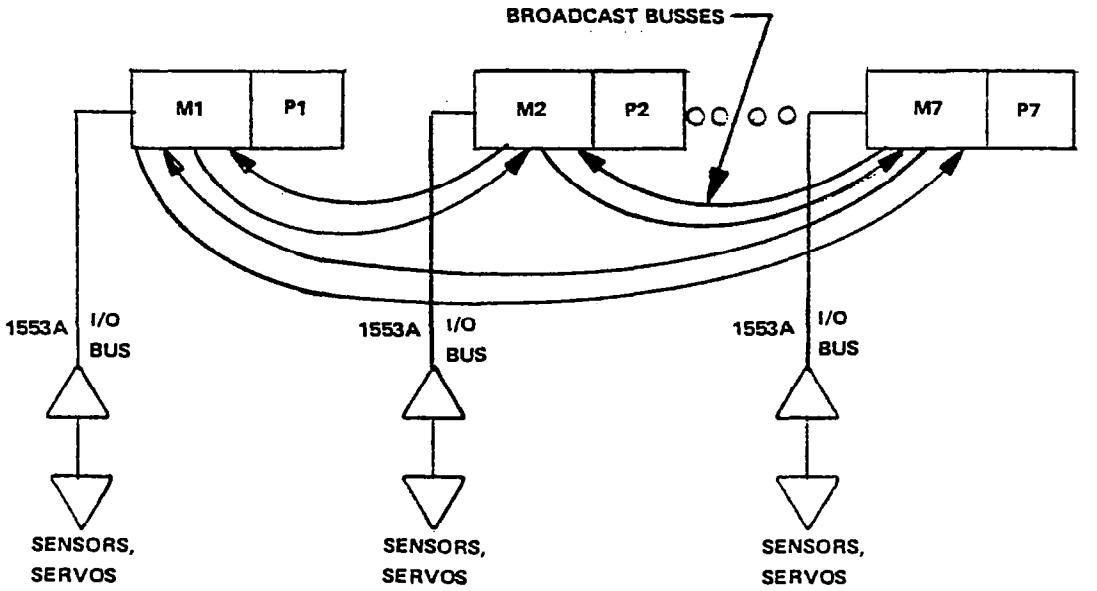
From Experiment #1 identify those faults which remain latent after several repetitions of the application task. These faults are likely to remain latent for long periods of time, making the detection or isolation of subsequent faults more difficult.

The experiment consists of injecting a latent fault in one processor and a random fault in another processor of the same redundant set and observing the time to detect, isolate and reconfigure.

Experiment #3

The results of Experiment #1 established the time frame for detection, isolation and reconfiguration. In this experiment the effect of a second fault in this time frame will be observed. The first fault is injected as in Experiment #1. A second fault is injected in a different processor of the redundant set at a randomly selected point in time within the time frame for reconfiguration. The subsequent detection, isolation and reconfiguration will be observed.

Each of these experiments would require a small modification to the Preprocessor program since they differ in the way faults are injected. In all experiments the emulator remains unchanged.



M1 . . . . M7 MEMORY
P1 . . . . P7 PROCESSOR

FIGURE 23 SIFT SYSTEM

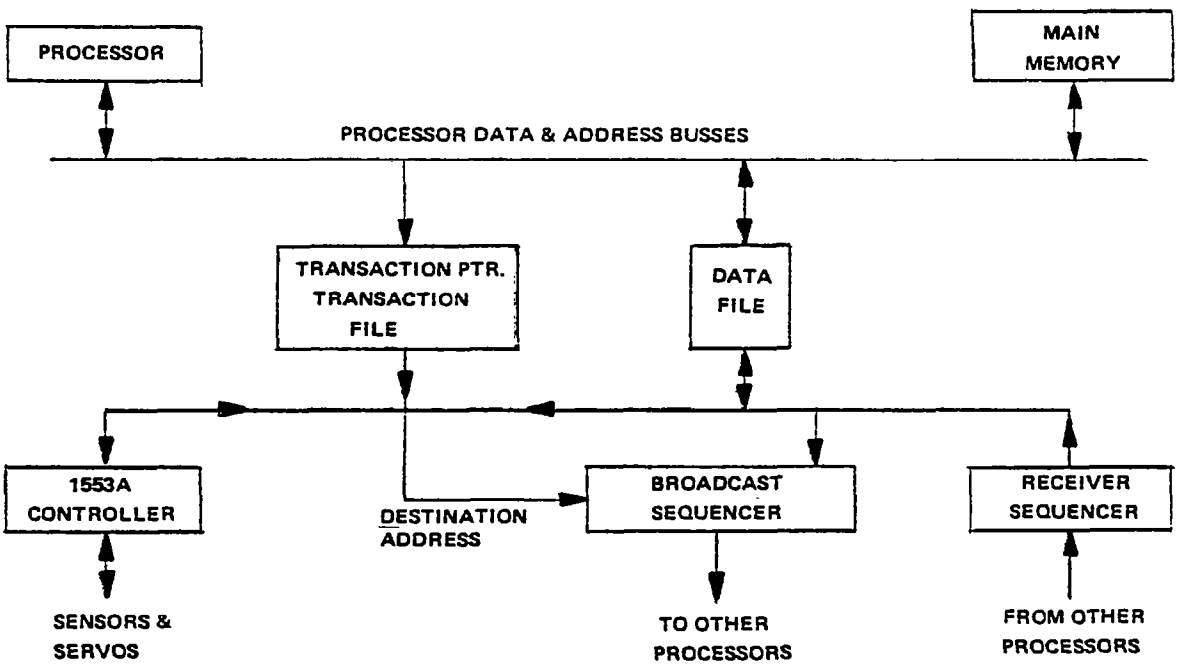


FIGURE 24 SIFT COMPUTER

13.0 CONCLUSIONS

On the basis of the study we conclude:

- Emulation is a practicable approach to failure modes and effects analysis of a digital processor.
- The run time of the emulated processor on a PDP-10 host computer is only 20,000 to 25,000 times slower than the actual processor. As a consequence large numbers of faults can be studied at relatively little cost and in a timely manner.
- The fault model, although somewhat arbitrary, can be updated as more data becomes available.
- Gate-level equivalent circuits are available for digital devices including the 2901.
- Gate-level faults are more difficult to detect than component-level faults.
- A computer self-test program of the order of 2000 executable instructions can detect 98% and possibly 99 or 100% of component-level faults. The feasibility of detecting the same proportions of gate-level faults remains to be determined.
- Emulation can be an important tool in the design of an efficient self-test.
- In a comparison-monitored system the accumulation of latent faults can be significant. In the study the proportion of undetected faults after 8 repetitions ranged from 40 to 62%.
- For the range of values considered the proportion of undetected faults after 8 repetitions is a linear function of the number of executable instructions.
- With a suitable choice of parameters the Urn Model can be used to describe fault latency in a comparision-monitored system. However, the proposed alternate model should be investigated.
- Faults in the micromemory are difficult to detect.
- In a comparison-monitored system most detected faults are detected in the first repetition of the program. Subsequent repetitions do not appreciably increase the proportion of detected faults.
- A gate-level emulation of a real processor may contain a large proportion of indistinguishable faults. Identifying such faults is difficult.
- Only 48% of all detected faults were detected by an explicit subtest of Self-Test; 52% were detected because the fault resulted in a wild branch.

- The results of the present study with regard to latent faults are in remarkably close agreement with the results of (ref. 1). The similarity is even more surprising when one considers that (ref. 1) employed a very simple, idealized processor with only 13 instructions and equally distributed faults. A comparison of the latency distributions for FETSTO, FIB and ADDSUB are given in Table 29. Based on this similarity it may be conjectured that the results of the present study can be extrapolated to other processors of comparable complexity.

TABLE 29

COMPARISON OF LATENCY ESTIMATES

	FETSTO DETECTED		FIB DETECTED		ADDSUB DETECTED	
REPETITION	REF (1)	PHASE I	REF (1)	PHASE I	REF (1)	PHASE I
1	.187	.3	.261	.35	.313	.335
2	.051	.048	.057	.055	.096	.027
3	.017	.021	.047	.007	.067	.03
4	.017	0	.009	.002	.024	.003
5	.017	0	.028	.002	.014	.005
6	.042	.006	.033	.002	.014	.003
7	0	.002	.019	0	.019	.002
8	.025	.007	.009	.002	.004	0
9 (undetected)	.644	.617	.537	.582	.449	.595

14.0 RECOMMENDATIONS FOR FUTURE STUDIES

- The Phase I experiments should be repeated using flight critical, flight control computations. The instruction set should not be limited as it was in the present study. Additional tasks would include
  - Determination of the proportion of faults that affect the control surfaces.
  - Determination of the proportion of faults that prevent failure detection in the faulted processor.
- Investigate other methods of fault detection such as the use of redundant computations in a non-redundant processor in a flight critical, flight flight control application.
- Investigate the feasibility of extending the emulation to I/O interface devices such as AD and DA converters, I/O controllers, etc.
- Generate more realistic fault models. Perhaps manufacturers could be prevailed upon to supply equivalent circuits that are more closely correlated with failure modes as well as with performance.
- Develop a more realistic Urn Model. The resultant model could be an important tool in reliability modelling of a redundant system.

15.0 REFERENCES

1. Nagel, P., "Modeling of a Latent Fault Detector in a Digital System". NASA CR-145371, 1978.

2. Mc Farlane Mood, A., Introduction to the Theory of Statistics, McGraw-Hill; New York, 1950.

3. Cramer, H., Mathematical Methods of Statistics, Princeton University Press; Princeton, 1958.

4. Moses, K., Forman, P., "SIFT: Multiprocessor Architecture for Software Implemented Fault Tolerance, Flight Control and Avionics Computers", Proceedings of the 3rd Digital Avionics Systems Conference, Fort Worth, Texas, November, 1979.

5. Seshu, S. and Freeman, D.N., "The Diagnosis of Asynchronous Sequential Switching Systems", IRE Transactions on Electronic Computers, Vol. EC-11 No. 4 August, 1962, pp. 459-465.

6. Hardie, F. H., and Suhocki, R. J., "Design and Use of Fault Simulation for Saturn Computer Design", IEEE Transactions on Electronics Computers, Vol. EC-16, No. 4, August, 1967, pp. 412-429.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-3462		

4. Title and Subtitle	5. Report Date
MEASUREMENT OF FAULT LATENCY IN A DIGITAL AVIONIC MINI PROCESSOR	October 1981
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
John G. McGough and Fred Swern	
9. Performing Organization Name and Address	10. Work Unit No.
Flight Systems Division Bendix Corporation Teterboro, N.J. 07608	11. Contract or Grant No. NAS1-15946
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered Contractor Report
National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

Langley NASA Project Engineer: Salvatore J. Bavuso
Progress Report

16. Abstract

This report describes the results of fault injection experiments utilizing a gate-level emulation of the central processor unit of the Bendix BDX-930 digital computer. The primary objective of the study was to ascertain the failure detection coverage of comparison-monitoring and a typical avionics CPU self-test program.

The specific tasks and experiments included:

1. Inject randomly selected gate-level and pin-level faults and emulate six software programs using comparison-monitoring to detect the faults.
2. Based upon the derived empirical data develop and validate a model of fault latency that will forecast a software program's detecting ability.
3. Given a typical avionics self-test program, inject randomly selected faults at both the gate-level and pin-level and determine the proportion of faults detected.
4. Determine why faults were undetected.
5. Recommend how the emulation can be extended to multiprocessor systems such as SIFT.
6. Determine the proportion of faults detected by a uniprocessor BIT (built-in-test) irrespective of self-test.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Emulation Gate-Level Fault Detection Fault Latency Self-Test Comparison-Monitoring	Unclassified - Unlimited Subject Category 59

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	209	A10

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

NASA-Langley, 1981