NASA Contractor Report-189590

p- 164

Advanced Information Processing System: Fault Injection Study and Results

Laura F. Burkhardt
Thomas K. Masotto
Jaynarayan H. Lala

THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA 02139

Contract NAS1-18565
May 1992

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

NASA Contractor Report-189590

Advanced Information Processing System: Fault Injection Study and Results

Laura F. Burkhardt
Thomas K. Masotto
Jaynarayan H. Lala

THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA 02139

Contract NAS1-18565
May 1992

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS v

LIST OF TABLES vii

1.0 APPROACH TO FAULT INJECTION 1-1
- 1.1 Introduction 1-1
- 1.2 Test Case Specification 1-3
  - 1.2.1 All Possible Test Cases 1-3
  - 1.2.2 Subset of Actual Test Cases 1-8
- 1.3 Test Case Measurements 1-9
- 1.4 Test Case Execution 1-10
  - 1.4.1 Experimental Setup 1-10
  - 1.4.2 Test Case Measurements 1-11

2.0 THE FAULT INJECTION ENVIRONMENT 2-1
- 2.1 Overview 2-1
  - 2.1.1 Hardware Fault Injection 2-1
  - 2.1.2 Software Fault Injection 2-3
- 2.2 Fault Injector Hardware 2-4
- 2.3 Fault Injection Software 2-16
  - 2.3.1 The FIS Main Menu 2-16
    - 2.3.1.1 The FIS Edit Faults Menu 2-16
    - 2.3.1.2 The FIS Multiplexer Signal Selection Menu 2-18
    - 2.3.1.3 The FIS Boolean Function Selection Menu 2-18
    - 2.3.1.4 The FIS Fault Application Option 2-19
    - 2.3.1.5 The FIS Save Fault File Option 2-22
    - 2.3.1.6 The FIS Load Fault File Option 2-22
    - 2.3.1.7 The FIS Reset Option 2-23
    - 2.3.1.8 The FIS Quit Option 2-23
  - 2.3.2 Fault Injection Software - Multiple Fault Application 2-23

3.0 APPLYING FAULTS TO THE I/O NETWORK 3-1
- 3.1 Overview of AIPS I/O Network 3-1
- 3.2 Specification of I/O Network Faults 3-2
  - 3.2.1 Creating Node and Link Faults 3-2
  - 3.2.2 Specification of Test Cases 3-3
- 3.3 Test Results 3-9
  - 3.3.1 Maximum and Average Times 3-10
  - 3.3.2 Frequency Histograms 3-23

3.3.2.1 Variance of the Detection Times....3-24
3.3.2.2 Variance of the Reconfiguration Times....3-24
3.3.2.2.1 Reconfiguration Variance-Second Reconfiguration Attempts....3-25
3.3.2.2.2 Reconfiguration Variance-Presumed Reconnection....3-25
3.3.2.2.3 Reconfiguration Variance-Inconclusive Analysis....3-26
3.3.2.2.4 Reconfiguation Variance-Simple and Complex Error Symptoms....3-26
3.3.3 Probability and Cumulative Density Functions....3-74
3.4 I/O Network Fault Injection Conclusions....3-75

4.0 APPLYING FAULTS TO THE CORE FTP....4-1
4.1 Overview of AIPS FTP....4-1
4.2 Specification of Core FTP Faults....4-3
4.2.1 Specification of Test Cases for Software-Injected Memory Faults....4-3
4.2.1.1 Fast FDIR....4-4
4.2.1.2 Watchdog Timer Reset....4-5
4.2.1.3 Background Self Test....4-5
4.2.1.4 Hardware Exception Handler....4-6
4.2.2 Specification of Test Cases for Hardware-Injected Faults....4-6
4.2.2.1 Transient FDIR....4-6
4.2.2.2 Lost Soul Sync....4-8
4.2.2.3 System Restart....4-9
4.2.3 Reconfiguration....4-9
4.3 Test Results....4-9
4.3.1 The Software Fault Injection Plan....4-9
4.3.2 The Hardware Fault Injection Plan....4-18
4.4 Core FTP Fault Injection Conclusions....4-21
4.4.1 Software Fault Injection Test Results....4-21
4.4.1.1 Maximum and Average Times....4-21
4.4.1.2 Probability and Cumulative Density Functions....4-32
4.4.2 Hardware Fault Injection Test Resutls....4-36
4.4.3 Design Flaws Uncovered by the Fault Injection Tests....4-37
4.5 Core FTP Fault Injection: Conclusions....4-38

5.0 CONCLUSIONS....5-1

6.0 REFERENCES....6-1

LIST OF ILLUSTRATIONS

Figure	Title	Page
1-1	Hardware Configurations	1-5
1-2	Software Configurations	1-6
1-3	Fault Injection Configuration	1-7
1-4	Fault Injection Experimental Setup	1-11
2-1	Experimental Setup-Software Fault Injection	2-3
2-2	Fault Injector Logical Organization	2-6
2-3	Insertion of FETs between Socket and Device	2-7
2-4	Fault Injector Implant	2-8
2-5	Fault Injector Hardware	2-9
2-6	Fault Description Word	2-11
2-7	Mux A, B, C Selection Word	2-12
2-8	Boolean Function Generator Data Word	2-13
2-9	The FIS Main Menu	2-16
2-10	The FIS Edit Faults Menu	2-17
2-11	The Fault Direction Options	2-17
2-12	The Fault Type Options	2-18
2-13	The FIS Mux Signals Menu	2-19
2-14	The FIS Boolean Function Menu	2-19
2-15	The Boolean Function Options	2-20
3-1	15 Node I/O Network-No Fault Configuration	3-2
3-2	Test A.1.a	3-27
3-3	Test A.1.b	3-28
3-4	Test A.1.c	3-29
3-5	Test A.2.a	3-30
3-6	Test A.2.b	3-31
3-7	Test A.2.c	3-32
3-8	Test A.3.a	3-33
3-9	Test A.3.b	3-34
3-10	Test A.3.c	3-35
3-11	Test B.1.a	3-36
3-12	Test B.1.b	3-37
3-13	Test B.1.c	3-38
3-14	Test B.2.a	3-39
3-15	Test B.2.b	3-40
3-16	Test B.2.c	3-41
3-17	Test B.3.a	3-42
3-18	Test B.3.b	3-43
3-19	Test B.3.c	3-44
3-20	Test B.4.a	3-45
3-21	Test C.1.a	3-46
3-22	Test C.1.b	3-47
3-23	Test C.1.c	3-48
3-24	Test C.1.d	3-49
3-25	Test C.2.a	3-50
3-26	Test C.2.b	3-51
3-27	Test C.2.c	3-52
3-28	Test C.3.a	3-53
3-29	Test C.3.b	3-54
3-30	Test D.1.a	3-55
3-31	Test D.1.b	3-56

3-32 Test D.1.c....3-57
3-33 Test D.1.d....3-58
3-34 Test D.2.a....3-59
3-35 Test D.2.b....3-60
3-36 Test D.2.c....3-61
3-37 Test D.3.a....3-62
3-38 Test D.3.b....3-63
3-39 Test D.4.a....3-64
3-40 Test D.4.b....3-65
3-41 Test D.5.a....3-66
3-42 Test D.5.b....3-67
3-43 Test D.5.c....3-68
3-44 Test D.5.d....3-69
3-45 Test E.1.a....3-70
3-46 Test F.1.a....3-71
3-47 Test F.2.a....3-72
3-48 Test F.3.a....3-73
3-49 The Probability Density Function for the Detection Times....3-74
3-50 The Cumulative Density Function for the Detection Times....3-75
3-51 The Probability Density Function for the Reconfiguration Times....3-76
3-52 The Cumulative Density Function for the Reconfiguration Times....3-76
4-1 Fault Tolerant Processor: Functional View (One Channel)....4-2
4-2 Data Exchange Network....4-18
4-3 Fault Tolerant Clock Network....4-19
4-4 The Probability Density Function for the Detection Times....4-33
4-5 The Probability Density Function for the Detection Times: Expansion of the 0 to 10,000 ms. Region....4-34
4-6 The Cumulative Density Function for the Detection Times....4-34
4-7 The Probability Density Function for the Reconfiguration Times....4-35
4-8 The Probability Density Function for the Reconfiguration Times: Expansion of Range 0 to 2400 ms....4-35
4-9 The Cumulative Density Function for the Reconfiguration Times....4-36

LIST OF TABLES

Table	Title	Page
2-1	Fault Injector Address Space	2-10
2-2	Fault Type Selection	2-12
2-3	Mux A, B, C Source Selection	2-13
2-4	Boolean Functions of Two Variables	2-14
2-5	Fault Direction Control	2-15
3-1	Victim Node Components-Schematic Location, Pin Number, and Fault Logic Level	3-4
3-2	Victim Port Components-Schematic Location, Pin Number, and Fault Logic Level	3-5
4-1	Software Fault Injection Plan	4-17
4-2	Hardware Fault Injection Plan	4-20
4-3	Software Fault Injection Results	4-26
4-4	Hardware Fault Injection Results	4-37

ADVANCED INFORMATION PROCESSING SYSTEM: FAULT INJECTION STUDY AND RESULTS

1.0 APPROACH TO FAULT INJECTION

1.1 Introduction

The overall objective of the Advanced Information Processing System (AIPS) program is to achieve a validated fault tolerant distributed computer system architectures suitable for a broad range of applications, including those which have a failure probability requirement as low as 10^{-9} at 10 hours. As a part of this process, an AIPS knowledgebase has been developed. Various domains of the AIPS knowledgebase and a design-for-validation methodology that uses the knowledgebase to synthesize computer system architectures are described in [1]. The present report focuses on the fault injection study and its results, which are a component of the performability knowledgebase.

To configure AIPS building blocks to meet specific application requirements, it is necessary to characterize performability, i.e., performance and reliability, of building blocks and of ensembles of building blocks as a function of fundamental architectural parameters. The performability knowledgebase would eventually consist of all quantifiable knowledge about the architecture that affects its performability. It is organized as analytical and empirical relationships between three major domains: performance metrics, reliability metrics and architectural parameters. The metrics and the AIPS architectural parameters are described in Section 5 of [1], which also discusses the empirical relationships between these three domains using the results obtained on the AIPS engineering model.

The requirement of extremely low system failure rates for the AIPS applications (typically 10^{-6} to 10^{-10} per hour) precludes computer reliability validation exclusively by any single technique, tool, or approach. A balanced validation plan that uses analytical models, formal proofs, empirical test and evaluation, and architectural attributes that enhance the "validatability" of the system [1] can be cost effective and feasible in achieving validated fault tolerant computer system architectures. The performability knowledgebase is an important part of the balanced approach to validation. A set of analytical models has been developed to characterize reliability and availability of the AIPS hardware building blocks. To appreciate the role of empirical evaluation in constructing the performability knowledgebase, we quote from [2] as follows:

"Design-for-validation concept consists of ... 1. The system is designed in such a manner that a complete and accurate reliability model can be constructed. All parameters of the model which cannot be deduced from the logical design must be measured. All such parameters must be measurable within a feasible amount of time."

The design of the AIPS building blocks has adhered to this precept of the "design for validation" methodology. For example, by complying with all the known theoretical requirements for Byzantine resilience, the reliability of the AIPS Fault Tolerant Processor or the Inter-Computer communications network can be modeled analytically with just a few parameters. It is not necessary to exhaustively enumerate failure modes and show that each mode is covered with the requisite probability [6]. The analytical models are discussed in Section 4 of [1] in the context of the Advanced Launch System mission requirements. The models are, however, general enough so that by changing a few parameters one can predict the reliability and availability for other mission scenarios also. The reliability models use three types of parameters: component failure rates, fault response times (detection and reconfiguration times), and fault coverages (detection and reconfiguration coverages). The component failure rates are estimated using the MIL-HDBK-217E. The other parameters, however, must be deduced from the design or measured experimentally.

Since engineering models of the AIPS building blocks have been fabricated, it is feasible to measure system response to faults and measure some of the parameters experimentally. The analytical modeling and empirical characterization of the AIPS building blocks complement each other. Analytical models are abstractions of physical reality. Test and evaluation on the engineering model can help verify model assumptions, determine unknown parameters and increase overall confidence, and hence claims of validation, in the system.

Apart from gathering data for reliability parameter estimation, fault injection plays another important role in the overall system validation. Fault injection can be used to obtain feedback for fault removal from the design implementation. Again, the role of fault injection in finding and fixing design errors should be kept in the proper perspective. One cannot rely solely on fault injection to uncover design, specification and implementation errors. Fault injection is not a substitute for the design-for-validation methodology. However, it *is* a component of the methodology just as specifications, design reviews, analytical models, and formal methods are.

If the fault injection process does not uncover a single flaw in the system under test, it does not imply that there are no flaws in the system, only that the system is correct with respect to the fault set to which it was subjected. But what if some design flaws *are* uncovered? Does that mean the exercise was useless? On the contrary, a utility of the fault injection technique is in uncovering shortcomings in the system. One gains a deeper understanding of the fault tolerance design, a more fundamental appreciation of the cascade of events triggered by a fault, including complex interactions between hardware and software elements and the timing relationships between various events.

With the above discussion in mind, the goals of this fault injection study, as stated in the statement of work, were as follows:

1. To test the system design specification for fault tolerance.

2. To obtain feedback for fault removal from the design implementation.

3. To obtain statistical data regarding fault detection, isolation, and reconfiguration responses.

4. To obtain data regarding the effects of faults on system performance.

The organization of this report is as follows. The remainder of this section describes the parameters that must be varied to create a comprehensive set of fault injection tests. The subset of test cases selected for this study, the test case measurements, and the test case execution are also described in Section 1. Both pin-level hardware faults using a hardware fault injector and software-injected memory mutations were used to test the system. Section 2 provides an overview of the hardware fault injector and the associated software used to carry out the experiments. Sections 3 and 4 give detailed specifications of faults and test results for the I/O Network and the AIPS Fault Tolerant Processor, respectively. Section 5 summarizes the results and gives conclusions of the study.

1.2 Test Case Specification

This section explains how the test cases used in the AIPS Fault Injection Study were derived. Section 1.2.1 describes the major parameters that could be varied in order to create a comprehensive set of tests. Section 1.2.2 describes how the parameters actually were varied in order to select from all the possible tests a limited subset that was executable within the time and financial constraints of the Fault Injection Study.

1.2.1 All Possible Test Cases

Any given fault that is injected occurs within a particular context. This context consists of the hardware environment, the software environment, and the fault injection environment. Thus there are four major parameters that can be varied when creating test cases:

- Hardware environment
- Software environment
- Fault injection environment
- The actual fault

Hardware Environment

The hardware environment consists of the hardware building blocks that make up the system during the particular test. These hardware building blocks may be arranged in varying configurations. The parameters that describe the possible configurations are:

- Redundancy level of the victim FTP

- I/O network configuration
- IC network configuration

A graphical representation of all combinations of these parameters is shown in Figure 1-1. For clarity, the variations of the parameters under the top level are shown only once.

Software Environment

Similarly, the software environment consists of the software building blocks that compose the system during the particular test. These software building blocks may be arranged in varying configurations. The parameters that describe the possible configurations are:

- Combinations of system functions and applications, and support functions
- Iteration rates of system functions
- Number of application tasks
- Computational loads of application tasks
- I/O requirements of application tasks
- IC requirements of application tasks

A graphical representation of all combinations of these parameters is shown in Figure 1-2. In this case each parameter could take on multiple values, but the figure arbitrarily shows only two. For clarity, the variations of the parameters under the top level are shown only once.

Fault Injection Environment

The fault injection environment consists of attributes of the faults to be injected. These attributes may be varied and arranged in different configurations. The attributes are:

- Number of FTPs monitoring fault injection
- Number of FCRs affected
- Number of simultaneous faults
- Duration of fault
- Random placement vs. selected placement
- Hardware injection vs. software injection
- Scope of fault (i.e., FCR, Board, Chip, Pin)

A graphical representation of all combinations of these parameters is shown in Figure 1-3. For clarity, the variations of the parameters under the top level are shown only once.

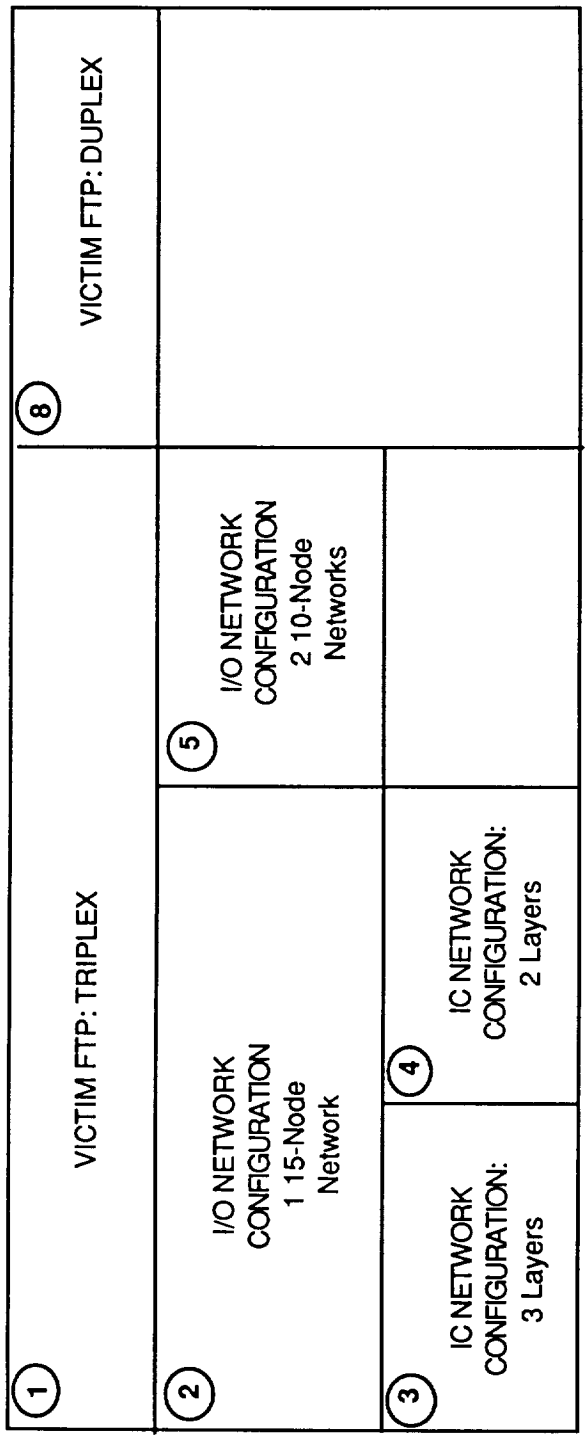

Figure 1-1. Hardware Configurations

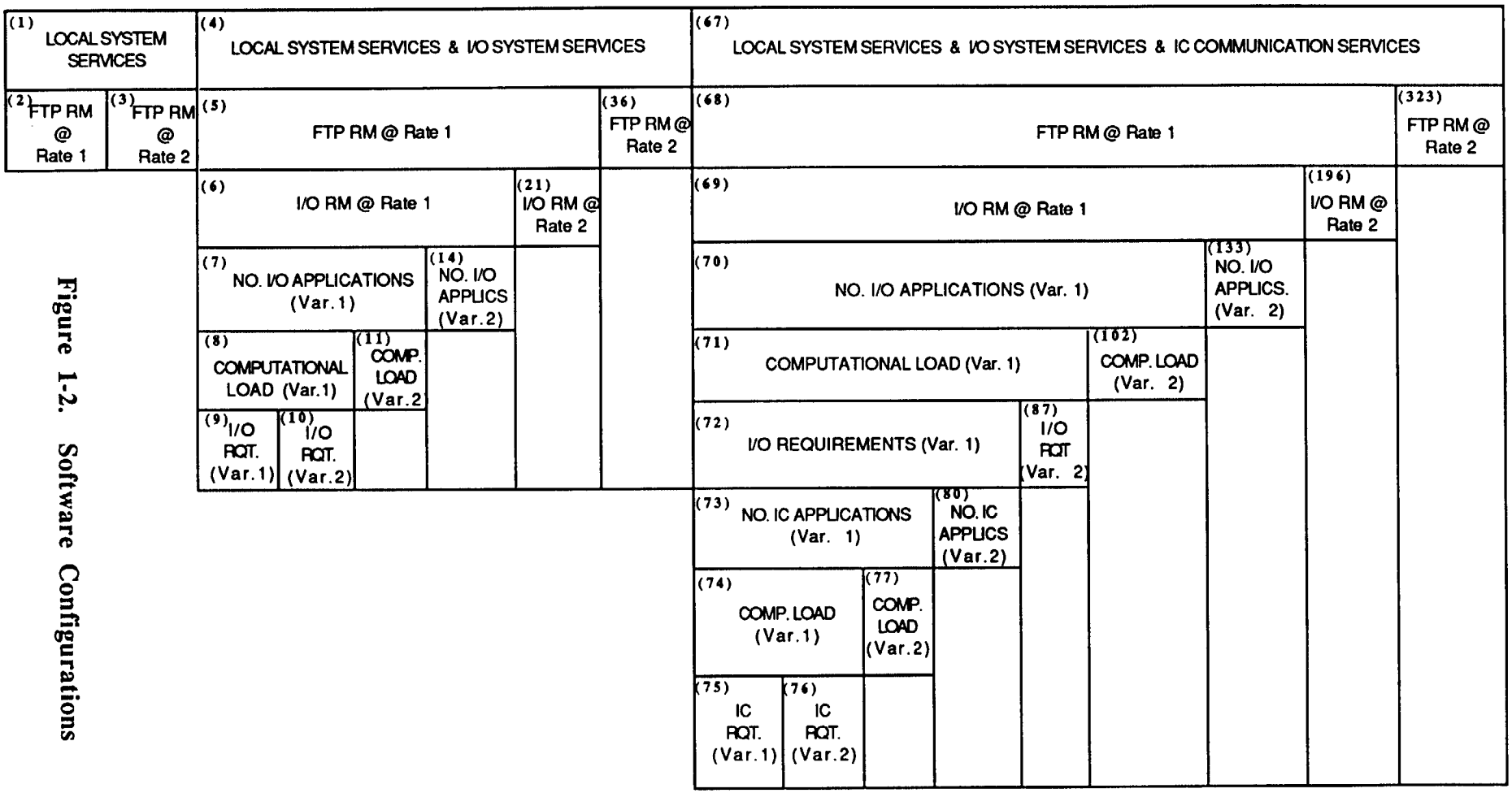

Figure 1-2. Software Configurations

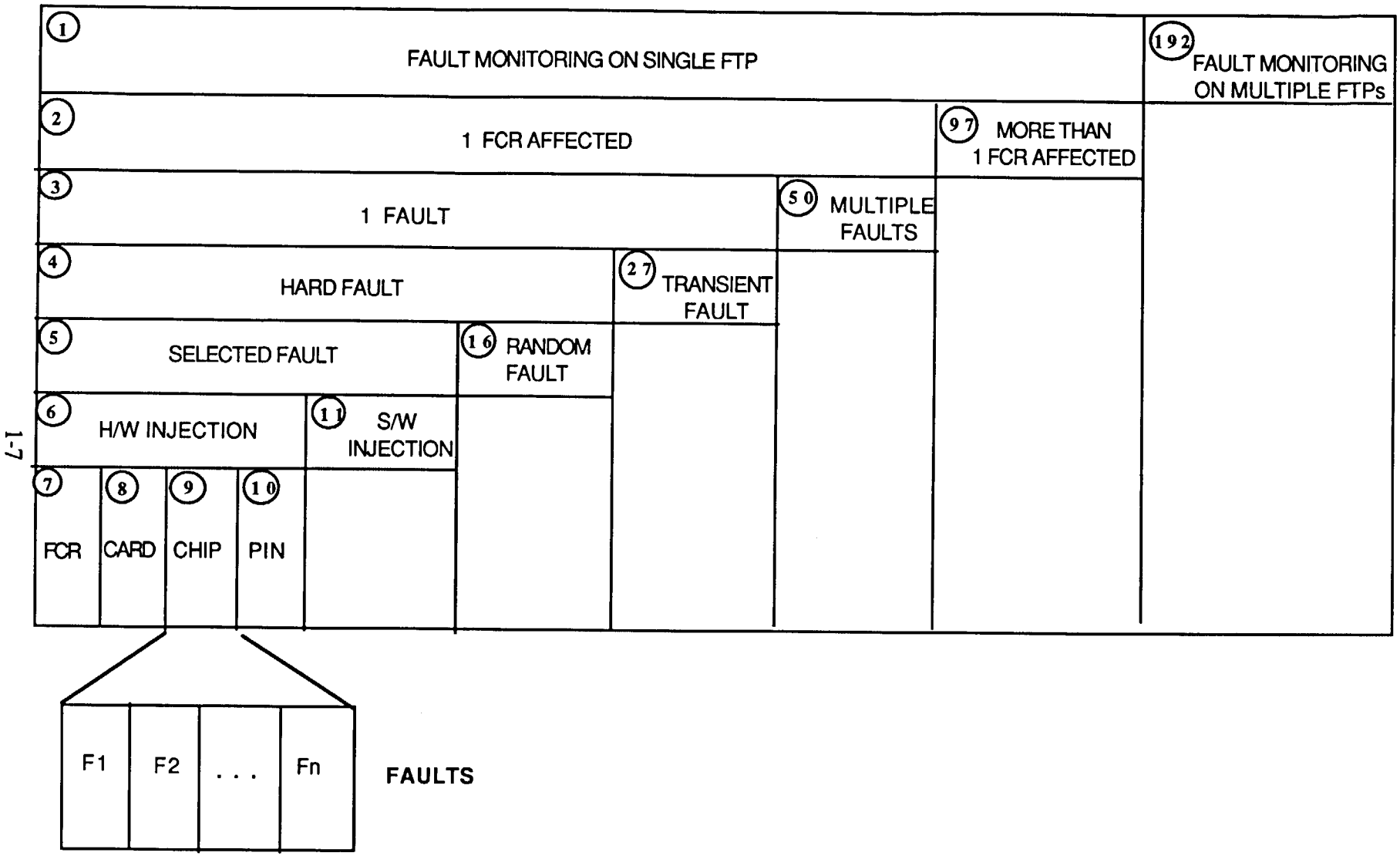

Figure 1-3. Fault Injection Configuration

The Actual Fault

Finally, each individual fault to be injected was determined by examining the possible places at which to inject a fault in each particular fault region (i.e., FCR, board, chip, pin). The individual faults are discussed in detail in Sections 3 and 4 of this report.

To reiterate, a test case consists of an actual **fault** within a **fault injection configuration** within a **software configuration** within a **hardware configuration.** The total number of possible test cases is the product of the number of hardware configurations, software configurations, fault injection configurations, and faults, i.e.,

$$N_{Tot} = N_{HW} * N_{SW} * N_{FI} * N_F$$

Each test case may be specified using the assigned numbers from Figures 1-1 through 1-3. An example test case would be:

H1/2/3 - S1/2 - FI1/2/3/4/5/6/10 - F1

The hardware configuration for this test case consists of a triplex FTP as the victim (H1), a 15-node I/O network (H2), and a 3-layer IC network (H3). The software configuration includes only Local System Services (S1) and has the FTP RM executing at Rate 1. The fault injection attributes include fault monitoring on a single FTP (FI1), only 1 FCR affected by the injection (FI2), only 1 fault injected at a time (FI3), that fault being a hard fault (FI4) selected for its ability to create a known effect (FI5). The fault is injected by means of the hardware fault injector (FI6) and is applied at the pin level (FI10). The actual fault injected is Fault #1 (F1).

1.2.2 Subset of Actual Test Cases

If all possible variations of the hardware environment, software environment and fault injection environment shown in Figures 1-1 through 1-3 were exercised with only one fault each, this alone would represent approximately 594,000 test cases (8 hardware configurations, 290 software configurations, 256 fault injection configurations). In order to limit the test cases to a number consistent with the time and financial constraints of this study, one hardware configuration, one software configuration, and two fault injection configurations were chosen. The number of test cases, then, came from the different faults that were injected within these two environments.

Using the notation given above, the two environments may be specified as

(1) H1/2/3 - S1/67/68/69/70/71/72/73/74/75 - FI1/2/3/4/5/6/10

and

(2) H1/2/3 - S1/67/68/69/70/71/72/73/74/75 - FI1/2/3/4/5/11/14

The hardware configuration in both cases consisted of a triplex FTP as the victim (H1), a 15-node I/O network (H2) and a 3-layer IC network (H3). The software configuration in both cases included Local System Services, I/O System Services, and IC Communication Services (S67), FTP RM at 40 Hz (S68), I/O RM every 2 seconds (S69), no I/O applications (S70, S71, S72), and two IC applications (S73) doing minimal computation (S74) and moderate IC communication (S75). The fault injection configurations included fault monitoring on a single FTP (FI1), only 1 FCR affected by the injection (FI2), only 1 fault injected at a time (FI3), that fault being a hard fault (FI4) selected for its ability to create a known effect (FI5). Faults were injected either at the pin level by the hardware fault injector (FI6/FI10)) or at the chip level by software fault injection (FI11/FI14).

The actual faults that were injected are discussed in detail in Sections 3 and 4.

It should be noted that as an ongoing part of AIPS testing and debugging efforts, faults have previously been injected at the FCR and card level. In the case of the core FTP, a channel failure could be simulated by resetting either one processor or the entire channel. Data exchange faults were simulated by use of specially constructed switches that grounded power to pertinent parts of the data exchange hardware. In the case of the I/O network, injected faults included resetting a node, unplugging a node card and unplugging links between two nodes. These faults were inserted manually and the collection of fault detection and reconfiguration times was not automated, so that collecting statistics about a large volume of faults would have been very tedious and time-consuming.

1.3 Test Case Measurements

To achieve the goals stated in Section 1.1, certain information must be collected for each test case.

Goal 1: to test the system design specification for fault tolerance. This means it must be determined that an injected fault was actually tolerated, i.e., that the system continued to perform correctly in the presence of the fault. This requires a record of system performance before the fault is injected, which can then be compared to the record of system performance during and after fault detection. To obtain this record of system performance, all of the tasks executing in the system for an appropriate time period around the fault injection must be tracked. In addition to the clock time at which each task suspends and resumes, this record must give some indication of what the task was doing so as to confirm that it is executing normally and is not in some erroneous state. In the absence of tools that provide such a record, externally visible manifestations must be relied on, such as correct functioning of the CRT and MAC displays, continued correct configuration of I/O nodes, and continued communication between IC applications.

Goal 2: to obtain feedback for fault removal from the design implementation. To do this the following questions must be answered for each test case.

- Was the fault detected?
- Was the fault isolated?
- Was it correctly isolated?
- Did reconfiguration occur?
- Was the reconfiguration correct?

Goal 3: to obtain statistical data regarding fault detection, isolation, and reconfiguration responses. This may be achieved by recording the times when each of the three events (detection, isolation, reconfiguration) occurs.

Goal 4: to obtain data regarding the effects of faults on system performance. The information required to answer this question is similar to that required for Goal 1. A record of system performance before and after fault detection is required to determine effects of a fault on performance. Short-term effects, i.e., the effect of extra fault detection and identification overhead on system throughput must be measured, as well as long-term effects, e.g., extra redundancy management time required for 2-layer IC messages rather than 3-layer messages.

1.4 Test Case Execution

1.4.1 Experimental Setup

The experimental setup for injecting faults into the AIPS Distributed Engineering Model and measuring their effects is shown in Figure 1-4. The Engineering Model is shown on the right and consists of four Fault Tolerant Processors (FTPs), a fault-tolerant Inter-Computer (IC) Communication Network, and a fault-tolerant Input/Output (I/O) Network. On the left is the Fault Injection Software (FIS) that resides on a MicroVAX 3900 computer. This software controls the number and type of faults and the time of their insertion. It also collects time information recorded by the FTP so that it can compute performance measurements such as fault detection and reconfiguration times.

To create hardware-injected faults, the FIS communicates with a Fault Injector device by means of a VAX Qbus interface. The Fault Injector interfaces with the Engineering Model in such a way that a signal from the VAX will cause a fault to occur at any desired pin in the Engineering Model. To create software-injected faults, the FIS communicates with the FTP through a shared memory referred to as the Testport Interface. The only type of fault injected by software is an emulated transient memory fault. The FIS also uses the Testport Interface to retrieve the timing data recorded by the FTP.

The Fault Injector and the Fault Injection Software are discussed in detail in Section 2.

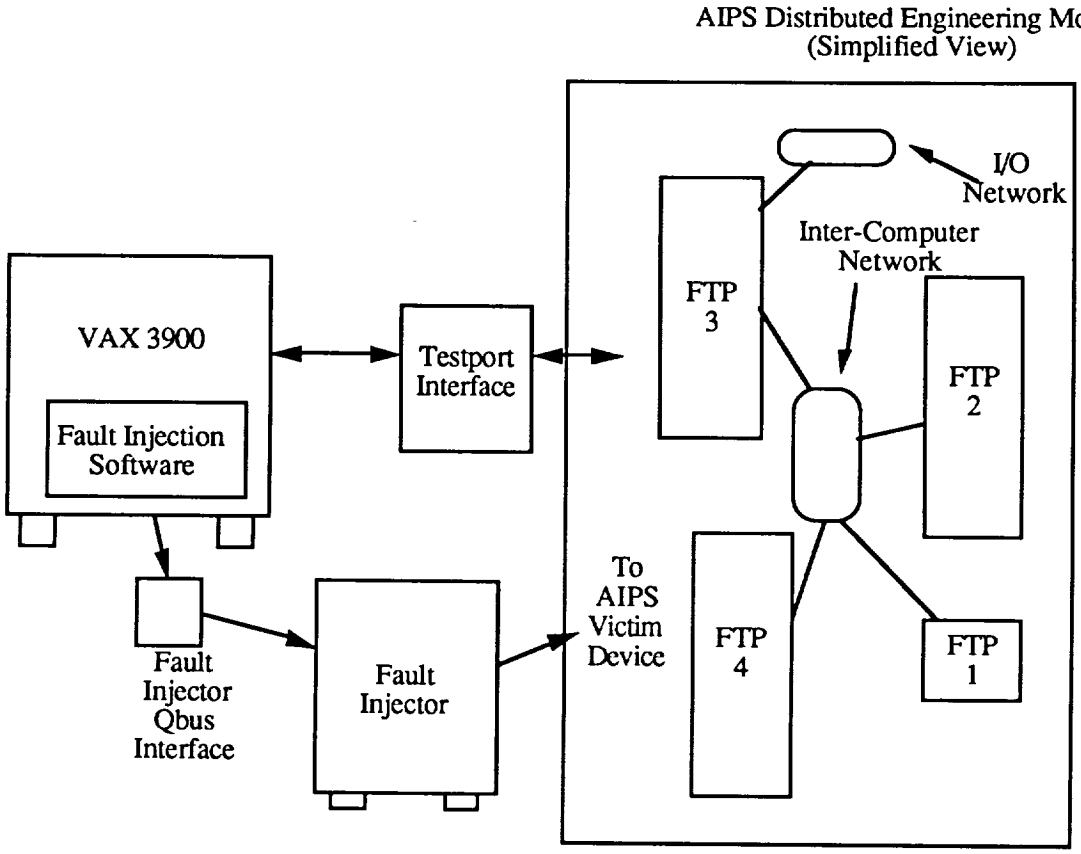

Figure 1-4. Fault Injection Experimental Setup

1.4.2 Obtaining Test Case Measurements

The test case measurements described in Section 1.3 were gathered by the Fault Injection Software and by visual inspection. This section explains how each type of measurement was obtained.

One step in **testing the system design specification for fault tolerance** (Goal 1) was to capture the record of system performance. This process was not automated and incorporated within the Fault Injection Software for this study. Instead we relied solely on visual observation of the CRT and MAC displays and configuration of the I/O network to determine that the system functioned correctly during and after the fault. Since the display tasks execute at the lowest priority, this ensured that no higher priority task was monopolizing the system as a result of the fault.

To determine the **correctness and completeness of the fault detection and identification** (Goal 2) for each test case, two methods were used. One was visual inspection of the error logs that are maintained by the core FTP RM and the I/O network

RM processes. These logs indicated whether a particular fault was isolated and, if so, whether it was isolated correctly. The other method involved setting breakpoints in the FTP code at the fault isolation and reconfiguration points and having the FIS verify that these breakpoints were reached. In addition, the FIS verified that a fault was recovered from before injecting the next fault.

Statistical data about the fault detection and identification (Goal 3) was obtained by having the FIS note the time at which the isolation and reconfiguration breakpoints were reached. In addition, the FTP code logged the time the fault was detected in the Testport Interface.

The effects of faults on system performance (Goal 4) include both (1) the additional time required by the particular FDIR process when it is dealing with a fault and the subsequent scheduling delays incurred by other tasks, and (2) the effects on users of the faulty component, e.g., an application task that receives erroneous input or cannot transmit output because an I/O fault has not yet been detected and/or reconfigured around. The additional time required by the FDIR processes has been measured at other times during the life of the AIPS project and documented in a previous report [1]. Additional measurements were also obtained in the I/O network portion of this Fault Injection Study but not in the core FTP portion.

The scheduling delays incurred by non-FDIR tasks and the effects of a fault on users of the faulty component require the same instrumentation as for Goal 1. This instrumentation was not in place at the time of this study, and this data was not collected.

2.0 FAULT INJECTION ENVIRONMENT

2.1 Fault Injection Experiment Setup

2.1.1 Hardware Fault Injection

To inject hardware faults into the AIPS Distributed Engineering Model, a device called the Fault Injector (FI) was designed and built at the Draper Laboratory [3]. The fault injector interfaces with the Engineering Model on one end and with the VAX 3900 Qbus on the other end. The number and type of faults and the time of their insertion are controlled by the Fault Injection Software (FIS) that is resident on the VAX computer. The VAX and the Engineering Model are linked together by a shared memory interface that was designed by Draper Laboratory (referred to as the Testport Interface). This interface is used by the Fault Injection Software to communicate with the AIPS system software executing on the Model. As a result, the experimental setup is a closed loop system in which the executor, the FIS, and the victim device on the Distributed Engineering Model are in constant touch with each other. This setup, as shall be seen later, makes it possible to automate the fault insertion process and to collect data that otherwise would not be possible to acquire.

Figure 1-4 is a block diagram of the experimental setup. The victim system is shown on the right and is composed of four Fault Tolerant Processors (FTPs), a fault-tolerant Inter-Computer (IC) Communication Network, and a fault-tolerant Input/Output (I/O) Network. Each of these building blocks was built with wire-wrapped circuit boards and open paneled chassis. Consequently, their electronic circuitry can be accessed by the Fault Injector's implants, described in the following paragraph, in a relatively easy manner.

Faults are normally injected one pin at a time. To insert faults, controllable DIP extenders or implants (part of the fault injector) are plugged into the DIP socket. Each implant accepts the DIP pins it replaced and contains circuitry which can interrupt (or reconnect) each DIP pin and each incidental signal line from their socket. Six implants, each of which handles 8 DIP pins, are provided. Thus, up to 48 pins on one DIP or on a combination of DIPs may be set up for fault injection at a given time.

The 48 implant pins of the fault injector are individually addressable by the VAX 3900. Each pin appears as a Qbus address to the Fault Injection Software (FIS). The type of fault to be produced at any pin is controlled by writing appropriate data to the Qbus address corresponding to the pin. Once a fault or set a of faults has been defined, they can be "enabled", that is, inserted into the victim by writing to another Qbus address. The fault injector hardware listens to this address space, decodes the data, and produces the fault that is requested. It also enables or clears the fault when appropriate data is written to the enable/clear address.

It is possible to produce signals other than simply the stuck-at class of faults. Faults that are boolean functions of signals on other pins can be generated. This can be used to

simulate faults which are rather unlikely but which have been known to happen. For example, it is possible to turn a NAND gate into a NOR gate. Nonetheless, the main utility of the Fault Injector lies in its ability to inject faults into tristate signals. For instance, the data pins of a random access memory (RAM) have signals that are either inputs to or output from the memory depending on whether memory is being written to or read from. To inject a fault into such a device pin, the direction of the fault signal should be correct in order to avoid any possible damage to the device. Such a signal can be produced by generating the fault as a function of other signals on the device that determine the direction of the data such as read/write and chip enable signals on the RAM DIP.

The Fault Injection Software has been written to facilitate automatic fault injection by providing commands that are used to define the victim device, map its pins into implant pins, specify the type of fault for each pin, and enable and clear faults. The FIS can execute a series of such commands, making it possible to go through a number of faults automatically once the victim device has been physically moved to the implants. Another condition necessary for automatic fault injection is some form of communication between the FIS and the AIPS system software to indicate whether the Engineering Model is ready to accept a new fault. Such messages between the FIS and the Model are exchanged using the Testport Interface. A modified version of the AIPS system software is responsible for communicating the timing metrics to the Testport's memory. The same data path is used in reverse to send messages from the FIS to the Distributed Model.

The FIS in conjunction with the AIPS software is able to record two times: the time of error detection and the time of system reconfiguration. The important metrics, however, are the time necessary to detect the fault and the time required to reconfigure around it. Since the time of fault injection is known to the FIS, the difference between the fault injection and error detection times constitutes the time required to detect the fault. Strictly speaking, this should be called the error detection time. However, we hypothesize that the error is caused by a fault. Therefore, the detection of an error is also an indirect indication of the occurrence of a fault. Since this is the earliest indication of a fault, we will call this the fault detection time. Also, the difference between the fault detection and the system reconfiguration times equals the time necessary to reconfigure around the fault.

To communicate the fault detection time to the FIS, the AIPS system software reads the AIPS real-time clock after the fault was detected and stores the time in the Testport Interface. To indicate the reconfiguration time, the AIPS system software suspends execution when it reaches a break-point at the end of the reconfiguration process. A program, AIPSDEBUG, stores the real-time clock value in the testport interface. When the FIS notices that the FTP is at the breakpoint, it reads the fault detection time and the reconfiguration time from the Testport Interface. Subsequently, the FIS calculates the times required for fault detection and reconfiguration. After these times have been determined and recorded for later analysis, execution of the AIPS system software is resumed to permit it to return to a state such that a new fault can be applied.

2.1.2 Software Fault Injection

To inject software faults into the AIPS Distributed Engineering Model, the Fault Injection Software and Draper Laboratory's Testport Interface are utilized. The FIS inserts faults by corrupting a channel's program and/or data memory. This is accomplished by sending commands through the Interface. As with the hardware fault injection, the number and type of faults and the time of their insertion are controlled by the FIS. After the designated number of faults have been injected, the FIS employs the Testport Interface to retrieve the fault detection and reconfiguration metrics.

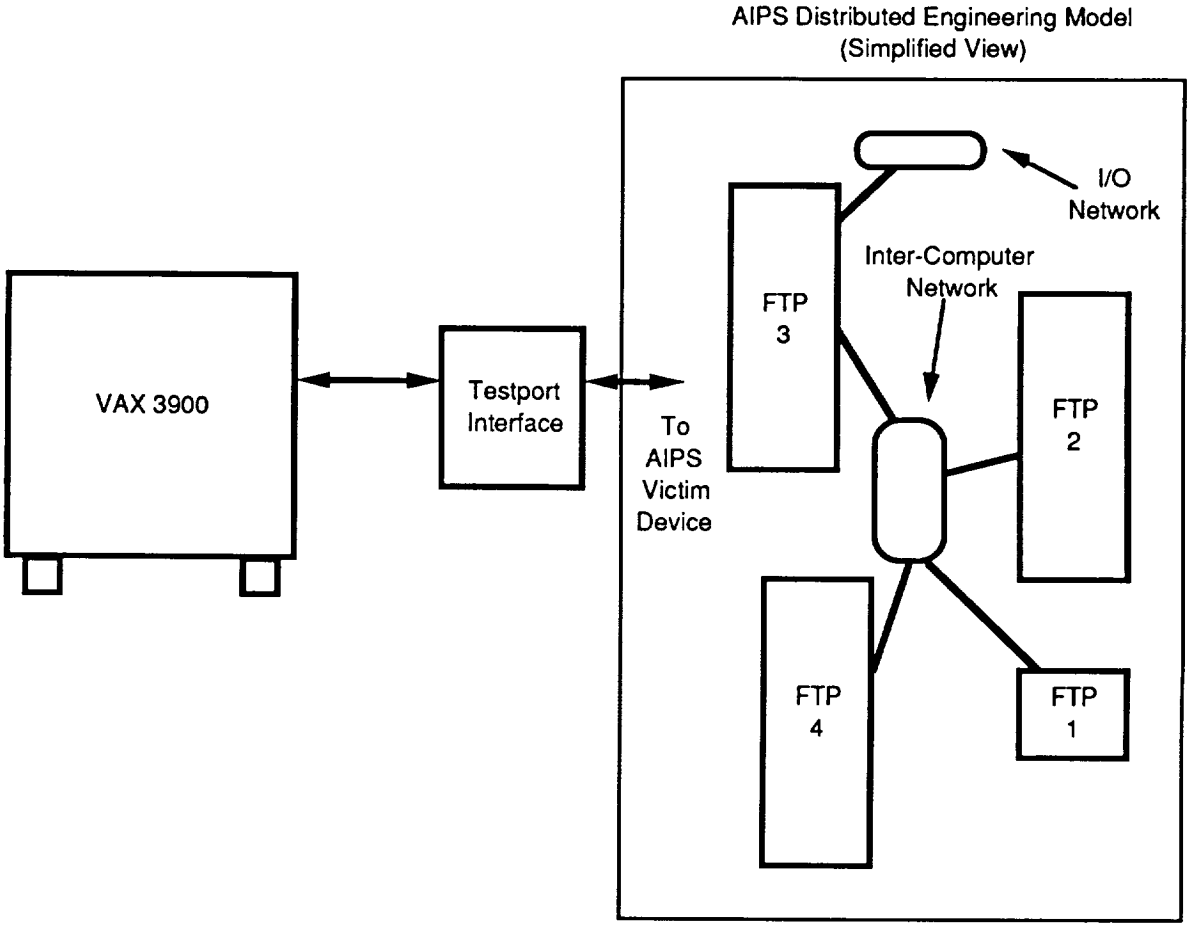

Figure 2-1. Experimental Setup - Software Fault Injection

A block diagram of the software fault injection setup is shown in Figure 2-1. As can be seen, this is a subset of system that is used for hardware fault injection.

The Fault Injection Software has been written to facilitate automatic injection of software faults by providing commands that allow the user to define the memory region to be

corrupted, the value of the fault, and the FTP channel and processor in which to insert the fault. As with hardware fault injection, the FIS can also execute a series of such commands, making it possible to go through a number of faults automatically.

The Fault Injector Hardware and Software are described in more detail in Sections 2.2 and 2.3 respectively.

2.2 Fault Injector Hardware

The fault injector hardware used for this study is described in detail in Reference 3. For the sake of completeness, a brief summary of the fault injector follows. A functional block diagram of the fault injector is shown in Figure 2-2. The heart of the fault injector is a pair of FETs that are interposed between the device pin and the socket pin. By turning on the FETs, a direct connection is established between the device and the socket. This is the normal situation when no fault is being injected on the pin. The device-socket connection can be severed by turning off the FETs. Now any desired signal may be applied to the device or the socket pin, whichever pin has the input signal (see Figure 2-3). A choice of eight signals is provided, one of which may be selected by multiplexer M1 for the device pin and by multiplexer M2 for the socket pin, as shown in Figure 2-2. A set of 48 FET and mux pairs are provided, one pair for each victim pin. This allows one to extend up to 48 pins on one DIP or a combination of DIPs. The choice of faults for each circuit pin is as follows.

1. Socket/Device Signal: This provides the original signal to the victim pin. That is, no fault is injected.

2. Mux A: This signal is the output of multiplexer A as shown in Figure 2-3. The inputs to the multiplexer are the 48 signals from the 48 pins that can be extended with the FETs. That is, a signal from any circuit pin or gate may be used as the fault or input signal for the victim pin.

3. Mux B: This multiplexer has the same function as Mux A.

4. Mux C: This multiplexer has the same function as Mux A.

5. f(A,B): This signal is the boolean function of two signals, the outputs of multiplexers A and B. Any of sixteen possible boolean functions may be specified.

6. f(A,B,C): This is a boolean function of f(A,B) and the output of multiplexer C. Any one of sixteen possible functions may be specified.

7. Ground: This provides the stuck-at-zero fault.

8. EXT: In addition to these seven selections, an externally generated signal may be used as a fault.

Each of the above eight signals may also be inverted before being applied to the victim pin, thus providing a choice of sixteen faults. The choice of faults thus includes stuck-at-one and "complemented signal" type of faults.

Multiplexers A, B, and C and the boolean function generators provide an extremely powerful capability to generate any type of fault. For example, certain faults in integrated circuits can change a NAND gate into a NOR gate. It is possible with this fault injector to simulate such a fault by extending all input and output pins of the target gate with FETs, generating the required boolean function using inputs from the gate inputs, and replacing the output with this signal. The main utility of this powerful capability, however, lies in the ability to inject faults on tristate signal lines. The direction of the fault can be made a function of other signals on the device, signals that determine the state of the tristate pin. It is thus possible to inject faults into data pins of memory chips and other tristate devices.

The fault injector hardware is physically packaged as shown in Figures 2-5 and 2-6. The FET pairs are mounted on an implant segment. Two sizes of implants are provided: 4 pin extenders and 8 pin extenders. An 8 pin implant has 16 FETs mounted on it and can extend one side of a 16 pin DIP. Dummy extenders that simply connect the socket and device pins without going through a FET are also provided. These are used to extend those device pins that are too sensitive to sustain that capacitance and/or time of delay of an intervening FET.

In any event, the FET implants are connected to multiplexer boards through a flat ribbon cable. As mentioned earlier, the fault injector has the capability of extending 48 device pins. The signals on each of these pins are controlled by a dedicated pair of multiplexers M1 and M2 (see Figure 2-2). Thus there is a total of 48 pairs of muxes. These are packaged on six multiplexer boards as shown in Figure 2-6. Each board controls 8 pins. One 8 pin implant or two 4 pin implants may be connected to each board. The six boards, labeled A, B, C, D, E, and F, are identical multiwire boards. Each contains one sixth of the multiplexers MA, MB, and MC. That is, each of the three 48:1 muxes (A, B, and C) is logically partitioned into six 8:1 muxes. Since a board handles 8 pins, a signal from one of these eight pins can be selected through the 8:1 mux (A, B, or C) on that board. The outputs of six logical parts of each 48:1 mux are OR'ed and distributed to all six circuit cards via the backplane. All 48 signals are then made available to each board. Each board also has its own copy of the three boolean function generators as shown in Figure 2-3. Functions f(A,B), F(A,B,C), and S(A,B,C) can be produced on any board. These signals, along with the outputs of muxes MA, MB, and MC form the inputs to the muxes M1 and M2.

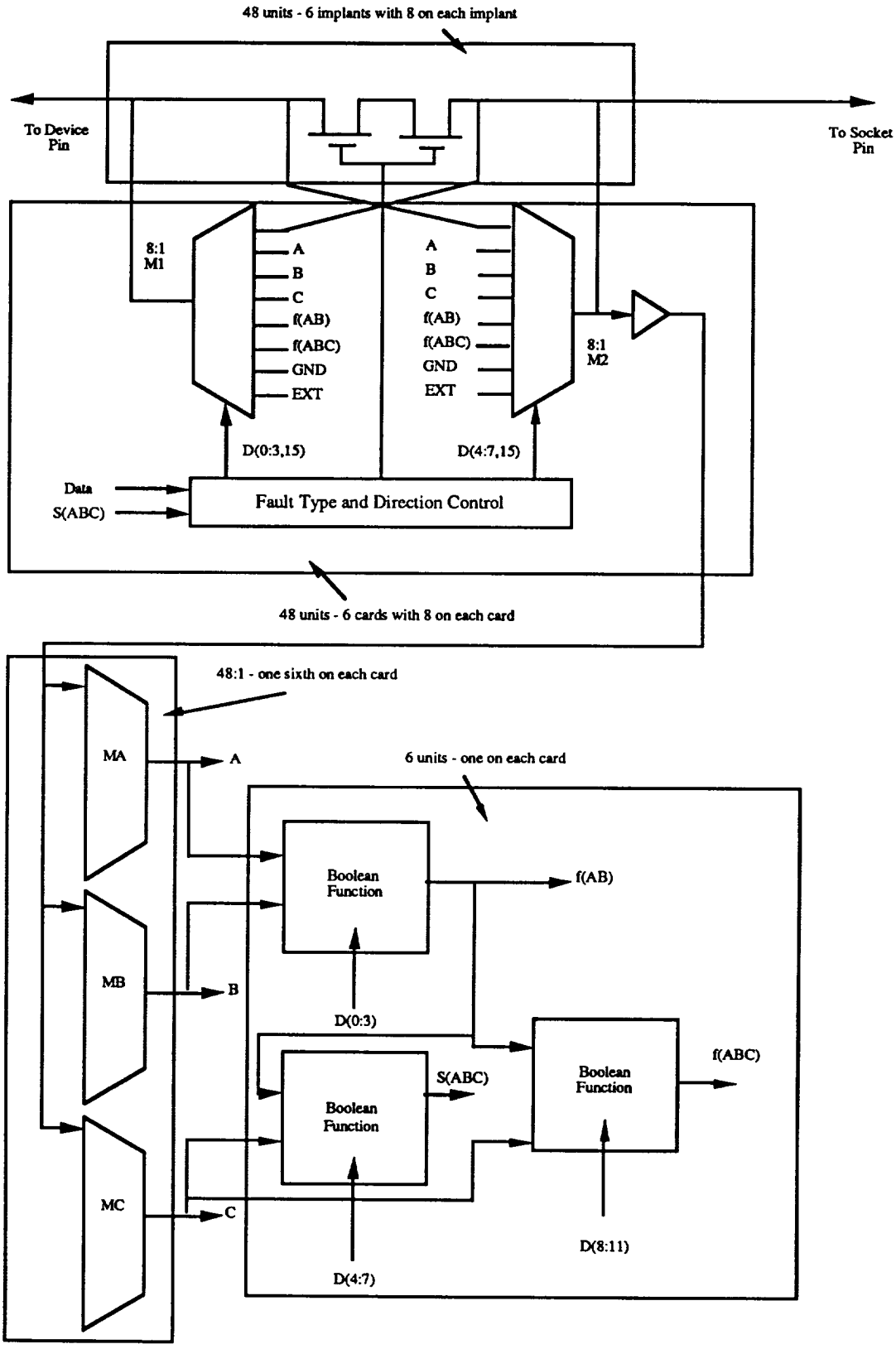

Figure 2-2. Fault Injector Logical Organization

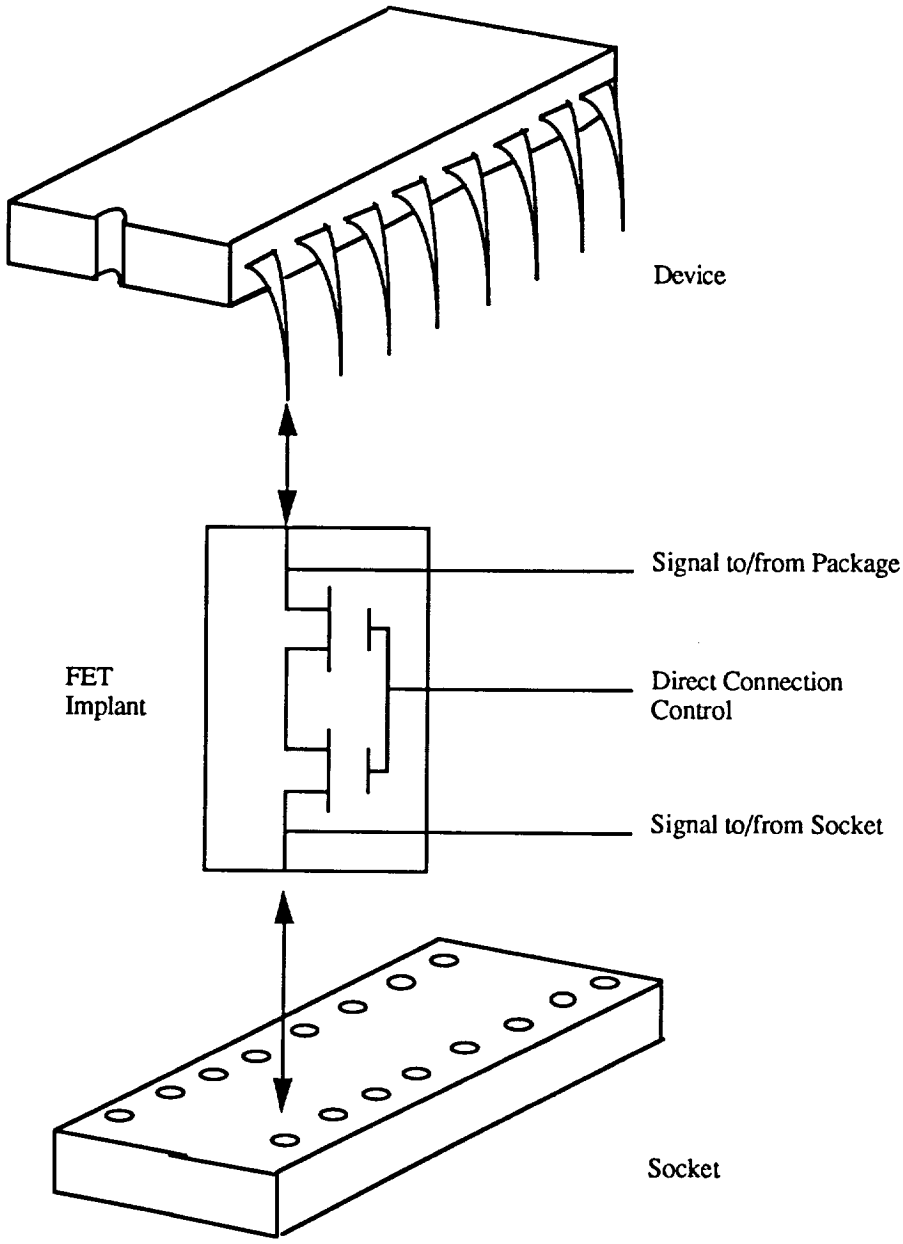

Figure 2-3. Insertion of FETs between Socket and Device

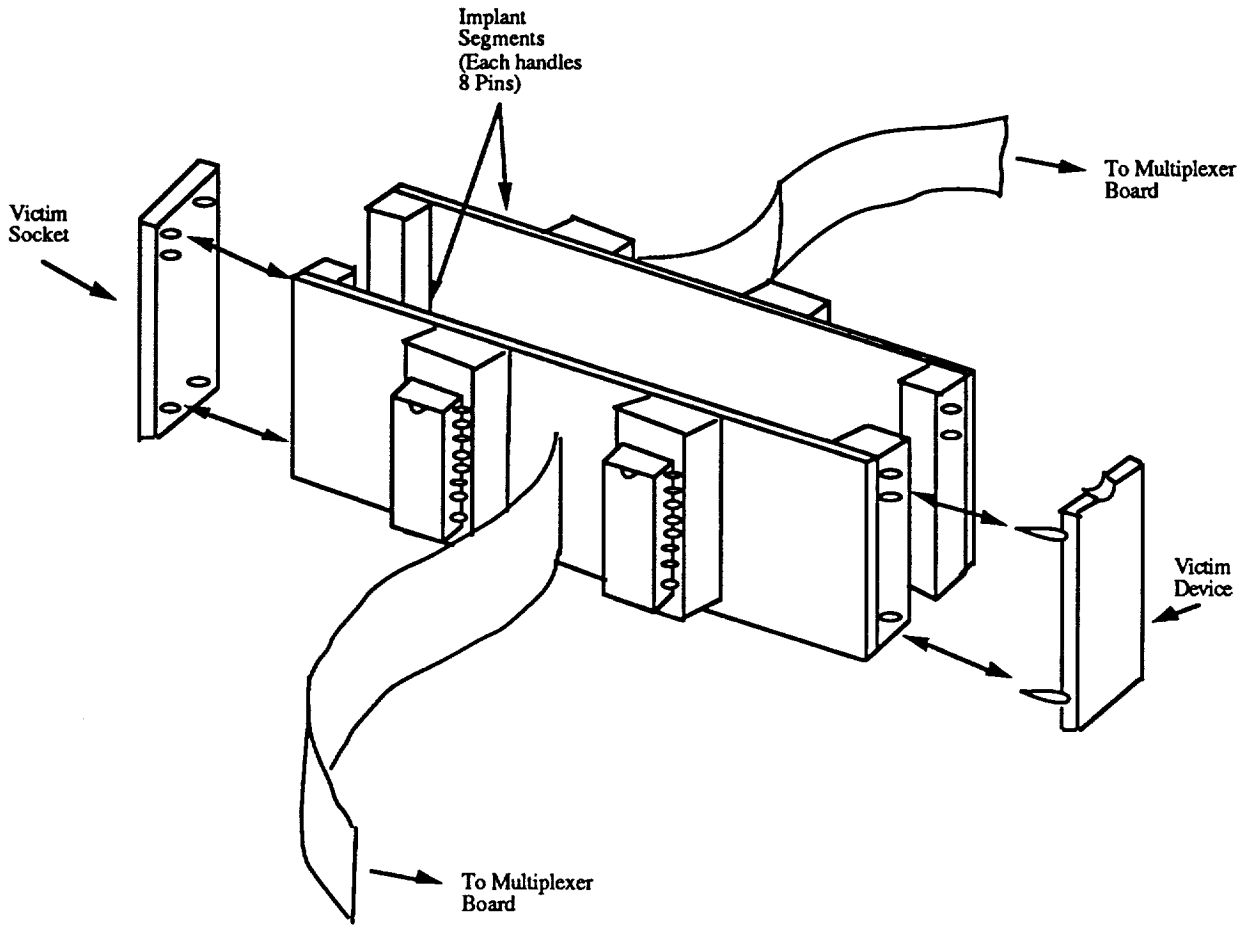

Figure 2-4. Fault Injector Implant

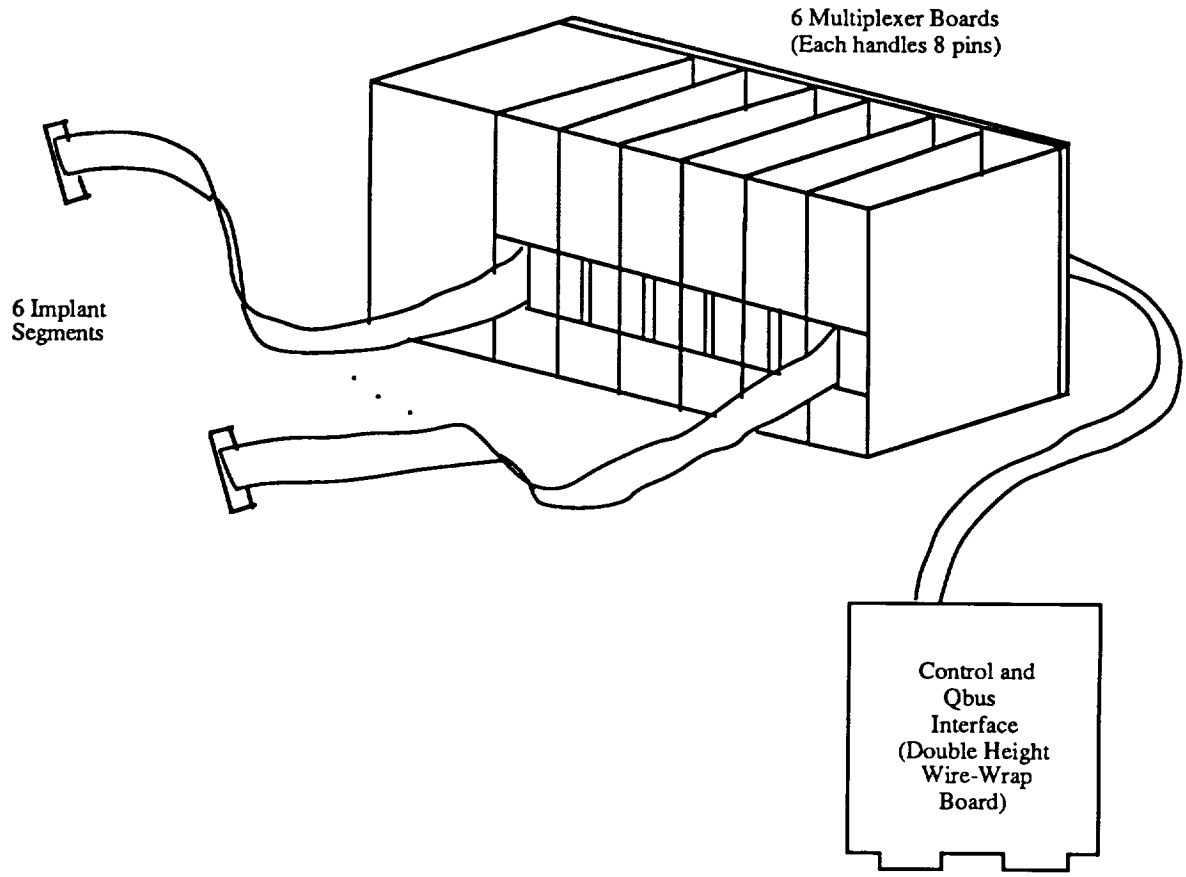

Figure 2-5. Fault Injector Hardware

Last, but not least, is the selection and control of the FETs, multiplexers, boolean function generators, and fault enabling and clearing logic. The fault injector has been designed such that it can be addressed as a Qbus device by a VAX 3900 computer. The data written to the Qbus address space of the fault injector is used to perform the selection and control functions. As shown in Figure 2-5, the backplane of the multiplexer boards is connected by flat-ribbon cables to a control and Qbus interface card. This is a double-height wire-wrap board that can be plugged into the VAX Qbus. It has the standard Qbus protocol and address decoding circuitry. The fault injector occupies the address space $3E980_{16}$ - $3E9FF_{16}$. This address space is mapped as shown in Table 2-1.

Address 3E9xx	Mux Board	Pin
80-8E	A	1-8
90-9E	B	1-8
A0-AE	C	1-8
B0-BE	D	1-8
C0-CE	E	1-8
D0-DE	F	1-8
E0	Mux A	
E2	Mux B	
E4	Mux C	
E6	Boolean Function Select	
EA	Execute/Clear Fault	
E8 and EC-FF	Unused	

Table 2-1. Fault Injector Address Space

Circuitry controlling signals on each of the 48 pins (muxes M1, M2, and FETs) is addressed individually (addresses $3E98_{16}$ to $3E9DE_{16}$). Data written to these addresses selects one of the eight inputs to mux M1 or M2 and controls the point of fault insertion (device or socket) by choosing mux M1 or M2. This is static operation. That is, the data written to these addresses is latched in the fault injector. The type and direction of the fault is thus determined, but the signal is not yet applied to the victim. To actually break the device-socket connection and inject the fault signal, one must write to the Execute/Clear address $3E9EA_{16}$. Writing a "0001" to this address enables the chosen multiplexer M1 or M2 on the chosen pin. It also turns off the pair of FETs on that pin. Faults on all the previously "enabled" pins are asserted simultaneously. The most significant bit of the fault selection data word determines if the pin is enabled. Writing a "0002" to the Execute/Clear address disables the muxes and turns on the FETs, thus clearing the fault condition.

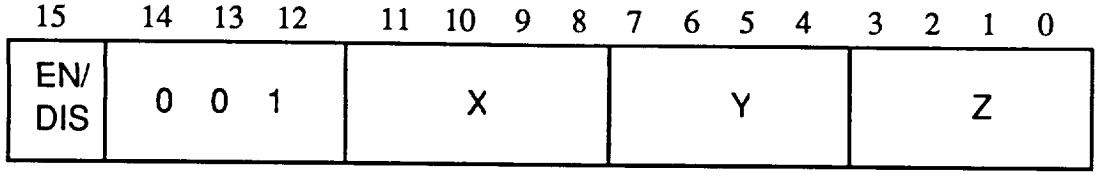

Figure 2-6. Fault Description Word

Bits 0-3 of the fault description word select the type of fault going to the device pin, bits 4-7 select the fault going to the socket, and bits 8-11 determine the direction of the fault (to device or to socket) as shown in Figure 2-6. Bit 15 enables and disables the pin. A pin must be enabled before a fault defined on it can be asserted. Bit 15 must be 1 for the pin to be enabled. Bits 12, 13, and 14 should always be as shown in Figure 2-6.

If the fault direction is 0, the fault as determined by the data bits Y is sent to the socket pin and data bits Z are ignored. If X is 8, then the fault as determined by Z is sent to the device pin and Y is ignored. In addition to 0 and 8, there are fourteen other values that can be assigned to X. Fault direction selected for these values of X is explained later in this Section (illustrated in Table 2-5).

Y and Z select the fault signal as shown in Table 2-2. If Y/Z is 8, the original signal is passed through the multiplexer unchanged. Stuck-at-1 and 0 faults can be generated by a value of 6 and "E", respectively. The signal can be inverted if Y/Z is 0. Other more complex faults can be chosen as outputs of multiplexers A, B, C, or a boolean function of their inputs (Y/Z = 1 to 5, 9 to "D").

If multiplexer A, B, or C output is either used directly as a fault or as input to a boolean function generator, it is necessary to select the multiplexer source. This is done by writing to the Qbus address of the multiplexer. Data written to the multiplexer address is interpreted as shown in Figure 2-7. Bits 3-5 select one of six boards A to F. Bits 0-2 select one of eight pins on that board as the mux output. These are shown in Table 2-3.

Y/Z	Fault Signal
0	Inverted Signal
1	F(A,B,C)
2	A
3	B
4	C
5	F(A,B)
6	1
7	EXT
8	Original Signal
9	/ F(A,B,C)
A	/ A
B	/ B
C	/ C
D	/ F(A,B)
E	0
F	/ EXT

Table 2-2. Fault Type Selection

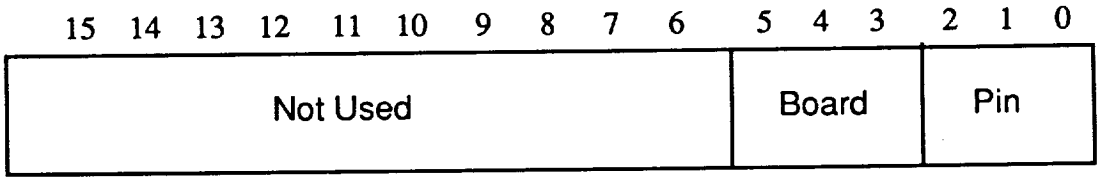

Figure 2-7. Mux A, B, C Selection Word

Data Bits 3 4 5	Board Selected	Bits 0 1 2	Pin Selected
1	A	0	1
2	B	1	2
3	C	2	3
4	D	3	4
5	E	4	5
6	F	5	6
		6	7
		7	8

Table 2-3. Mux A, B, C Source Selection

If a boolean function such as f(A,B) or f(A,B,C) is chosen as the desired fault, then one must also define the boolean function by writing to the function select address $3E9E6_{16}$. The data written to this address is interpreted as shown in Figure 2-8.

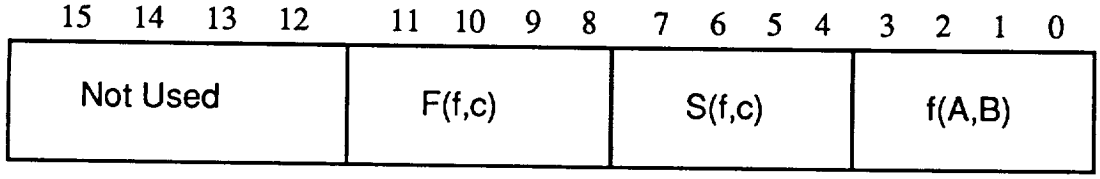

Figure 2-8. Boolean Function Generator Data Word

Bits 0-3 of the boolean function data word are used to select one of sixteen functions of signals A and B. Bits 4-7 are used to select f(A,B,C), which is one of the sixteen boolean functions of f(A,B) and C. Further, bits 8-11 are used to select S(A,B,C), which is also one of sixteen boolean functions of f(A,B) and C. The sixteen possible boolean functions of two variables are shown in Table 2-4.

As noted earlier, the fault direction (to device or to socket) is controlled by a 4-bit field X as shown in Figure 2-6. X can assume one of sixteen possible values. These are interpreted as shown in Table 2-5.

If the fault direction signal chosen by X is high, the fault is asserted on a socket pin. If it is low, the fault is asserted on a device pin.

Data	Boolean Function of A, B
0	0
1	/A * /B
2	A * /B
3	/B
4	/A * B
5	/A
6	A + B
7	/A + /B
8	A * B
9	/(A + B)
10	A
11	A + /B
12	B
13	/A + B
14	A + B
15	1

Table 2-4. Boolean Functions of Two Variables

For X equal to 0, the fault direction signal is high and the fault is sent to the socket. For X equal to 8, the fault is applied to the device. The fault direction in these two cases is static. For other values of X, the fault would be dynamically applied to the socket or device pin depending upon whether the chosen signal is high or low, respectively. The signals that can be used for direction control are the outputs of the multiplexers A, B, C, or their boolean functions f(A,B), S(A,B,C) and their complements. This allows one to dynamically control fault direction on tristate pins.

As explained earlier, fields Y and Z in the fault description word determine the type of fault to be applied to socket and device pins, respectively (see Figure 2-6). When X is equal to 0 or 8, only one of these two fields (Y when X is 0 and Z when X is 8) needs to be defined. Nevertheless, both Y and Z must be defined when X is not 0 or 8. But Y and Z need not be the same. That is, different faults can be applied to socket and device pins. In fact, by an appropriate choice of Y and Z, a fault can be applied in one direction while the original signal is passed through unchanged in the other direction. One may, for example, wish to insert a fault in a data pin of a memory chip only when data is being read but not when data is being written into the memory. This can be done by selecting a fault direction

signal that is high during the memory read cycle. The fault selected by Y would be applied to the socket pin during the read cycle. By choosing Z to be 8, correct data would be written to the memory during memory write cycle since Z equal to 8 passes the original signal to the device pin.

X	Fault Direction
0	To Socket
1	/S(A,B,C)
2	/A
3	/B
4	/C
5	/F(A,B)
6	Not Used
7	Not Used
8	To Device
9	S(A,B,C)
A	A
B	B
C	C
D	F(A,B)
E	Not Used
F	Not Used

Table 2-5. Fault Direction Control

Multiplexer output selection and boolean function definition are static functions. Data written to these addresses is latched in the fault injector.

It should be mentioned that the fault injector is a "write-only" device. The state of the fault injector can not be determined by reading its address space.

It is not necessary to remember various addresses of the fault injector since the fault injector software maintains these addresses in a database. This software provides appropriate commands to define fault types and select mux outputs. The next section describes the fault injector software.

2.3 Fault Injector Software

The fault injection software (FIS) resident on the VAX 3900 provides commands to perform all functions necessary to inject faults into the AIPS Distributed Engineering Model and record the results. The FIS is a menu driven program that permits the selection of fault types and their subsequent insertion into one or more victim devices.

2.3.1 The FIS Main Menu

The main menu of the fault injection software is illustrated in Figure 2-9. As shown, it permits the fault injection supervisor to edit faults, select input signals for the multiplexers, choose one or more boolean functions, insert a set of faults, save and load fault files, and reset the fault injection hardware. The aforementioned options are discussed in the following sections.

Selection	Fault Injector Main Menu
A	Edit Faults
B	Select Signal for Multiplexers
C	Choose a Boolean Function
D	Apply Faults to Victim
E	Save Fault File
F	Load Fault File
G	Reset
H	Quit

Figure 2-9. The FIS Main Menu

2.3.1.1 The FIS Edit Faults Menu

The Edit Faults menu of the fault injection software is depicted in Figure 2-10. It allows the creation, deletion, and confirmation of a fault or a set of faults. In order to create a fault, the selection A must be chosen. Subsequently, the FIS requests the associated multiplexer board (A-F), the pin (1-8), the fault direction, and fault type. The multiplexer and pin number identify the desired fault injector mux board, implant segment, and pin through which to apply the fault (described in Section 2.2 - Fault Injector Hardware). The fault direction indicates one of twelve options, each of which is presented in Figure 2-11.

Furthermore, one of fourteen faults can be inserted into the victim device. The fourteen possible fault types are indicated in Figure 2-12.

Selection	Edit Faults Menu
AB CD	Create FaultDelete FaultView Fault PackageReturn to Main Menu

Figure 2-10. The FIS Edit Faults Menu

If a previously created fault is not desired, it can be removed by selecting option B in the Edit Faults menu. In order for this extraneous fault type to be deleted, the corresponding multiplexer and pin must be identified.

The fault set that is currently defined can be viewed by choosing option C. This capability is incorporated into the FIS to permit the confirmation of a fault suite.

To return to the main FIS menu, option D of the Edit Faults menu must be selected.

Selection	Fault Direction
1	To Socket
2	/S(A,B,C)
3	/A
4	/B
5	/C
6	/F(A,B)
7	To Device
8	S(A,B,C)
9	A
10	B
11	C
12	F(A,B)

Figure 2-11. The Fault Direction Options

Selection	Type of Fault
1	Original Signal
2	Inverted Signal
3	1
4	0
5	F(A,B,C)
6	A
7	B
8	C
9	F(A,B)
10	/ F(A,B,C)
11	/ A
12	/ B
13	/ C
14	/ F(A,B)

Figure 2-12. The Fault Type Options

2.3.1.2 The FIS Multiplexer Signal Selection Menu

The input signal to the fault injector multiplexers A, B, and C can be specified by choosing option B of the main menu. This selection displays the Multiplexer Signal Selection menu which is illustrated in Figure 2-13. As expected, this menu has an entry for each of the three multiplexers. After the desired multiplexer is selected, the FIS requests information concerning the source of the multiplexer's input. That is, the corresponding multiplexer board (A-F) and pin (1-8) that should be used as the input signal.

To determine the connectivity between the multiplexers (A, B, and C) and pins (1-8) of the multiplexer boards (A-F), see Figure 2-2, the Fault Injector Logical Organization.

2.3.1.3 The FIS Boolean Function Selection Menu

It may be desirable to inject a signal into a victim device that is the boolean function of one or more input signals. The Boolean Function Selection menu, shown in Figure 2-14, which is entered by selecting option C of the main menu, permits the specification of the fault injector functions f(A,B), f(f,C), and S(f,C). As described in the Section 2.1 and illustrated in Figure 2-2, f(A,B) is a boolean function of the outputs of multiplexers A and

B. Alternatively, f(f,C) and S(f,C) are functions of the output of f(A,B) and multiplexer C.

To define f(A,B), f(f,C), or S(f,C), the associated option of the Boolean Function Selection menu must be selected. The FI software then prompts the fault injection supervisor for the corresponding function. The function can be one of the fifteen possibilities shown in Figure 2-15.

2.3.1.4 The FIS Fault Application Option

To apply a fault to a victim device, a fault set must be created or a previously defined fault package must be loaded. The creation of a fault is discussed in Section 2.2.1.1, and the method to load a fault set is described in Section 2.2.1.5. After a fault package has been devised and downloaded to the fault injector, option D of the main menu should be selected. This option applies the fault set to the victim device.

Selection	Signal for Mux Menu
A	Select Signal for Mux A
B	Select Signal for Mux B
C	Select Signal for Mux C
D	Return to Main Menu

Figure 2-13. The FIS Mux Signals Menu

Selection	Select Boolean Function Menu
A	Select Function for f(A,B)
B	Select Function for f(f,C)
C	Select Function for S(f,C)

Figure 2-14. The FIS Boolean Function Menu

Selection	Boolean Function of A, B
1	0
2	/A * /B
3	A * /B
4	/B
5	/A * B
6	/A
7	A + B
8	/A + /B
9	A * B
10	/(A + B)
11	A
12	A + /B
13	B
14	/A + B
15	A + B

Figure 2-15. The Boolean Function Options

To inject, observe, vary, and accurately record the fault injection test, the FIS requests the following information:

1. The fault detection bucket size.

   The fault detection times are grouped into "buckets", which are collections of times that fall in a particular range, to create a frequency histogram. The size of the bucket is selectable by the fault injection supervisor.

2. The reconfiguration detection bucket size.
   Similar to the fault detection times, the reconfiguration metrics are placed into "buckets". The size of the reconfiguration groupings is selectable by the fault injection supervisor.

3. Hardware or Software Fault Injection.

   The Fault Injection Software can insert hardware faults via the Fault Injector or software faults by corrupting the FTP's memory. The type of fault application is selectable by the user. If software faults are desired,

the FIS program requests the channel and memory range to be victimized and the numerical value of the fault.

4. The channel in which the fault will be inserted.

   When faults are applied to the core Fault Tolerant Processor (FTP), the FIS must know the channel through which the fault is being applied. This information assists the FIS in re-synchronizing the FTP in preparation for a successive test.

5. Whether a fixed or random time between successive faults is desired.

   The FIS permits the fault injection supervisor to insert a fixed amount of time between successive tests or to have the FIS randomly select the interval. If a fixed time is desired, then the supervisor is prompted for the corresponding number of seconds.

6. Number of times to insert the fault.

   The FIS sequentially executes a series of tests by repeatedly injecting the fault suite into the victim device. The number of times that the fault is applied is selectable.

7. Number of milliseconds to apply the fault.

   The length of time that a fault is inserted is selectable in quanta of 10 milliseconds. This is the least count of the Microvax 3900 clock. Since the periodicity of the different modules of the Redundancy Management Software varies considerably, the capability of changing the application interval is desirable.

8. The "Reconfiguration Time-Out".

   The "Reconfiguration Time-Out" is the maximum number of seconds to wait for the completion of the reconfiguration phase.

   It is possible that a fault is not detected (due to a software bug or a "don't care" condition). Accordingly, it is desirable to have a facility that permits the continuation of the series of tests albeit that one or more fault tests were not detected. This FIS facility is the Reconfiguration Time-Out. If it is exceeded, the FIS logs its occurrence and begins the execution of the next test. The length of this time-out is specified by the fault injection supervisor.

9. The "Fault Re-insertion Time-Out".

   The "Fault Re-Insertion Time-Out" is the maximum number of seconds to wait for the AIPS Engineering Model to return to a known state.

   After a fault has been detected and reconfigured around, the state of the AIPS Engineering Model must be returned to its state prior to the application of the fault. It is possible that a fault could cause the Model to enter an "unknown state" (due to a software bug). Accordingly, it is desirable to have a facility that permits the abortion of the series of tests if an inconsistent state is entered. This FIS facility is the Fault Re-Insertion Time-Out. If it is exceeded, then the FIS program logs this occurrence and stops. The length of this time-out is selected by the fault injection supervisor.

10. Names of the load modules that are executing in the computational and I/O processors of the AIPS Engineering Model.

    In order to maintain configuration control over the experimental test set up, the load modules that are involved must be noted. Accordingly, the FIS prompts the supervisor for the names of these files.

11. Name of the Output File.

    Since the AIPS Fault Injection plan involves many diverse tests and each test will be unique, the results of each test should be stored in a file with a name that identifies the test. Accordingly, the FIS prompts the fault injection supervisor for the name of the output file.

2.3.1.5 The FIS Save Fault File Option

It is often desirable to save a fault suite for later use. Option E of the main menu allows the fault injection supervisor to save a file that was created using option A (described in Section 2.2.1.1). After option E is selected, the supervisor is prompted for the name of the file in which to store the fault set.

2.3.1.6 The FIS Load Fault File Option

In order to use a previously defined fault package, it must be downloaded to the fault injector. Option F of the main menu allows the fault injection supervisor to load a fault file, that was saved using option E (described in Section 2.3.1.5), into the FI. After option F is selected, then the supervisor is prompted for the name of the file in which the associated fault set is stored.

2.3.1.7 The FIS Reset Option

If it is desirable to reset the fault injector, it can be accomplished by selecting option G of the main menu. This process resets Multiplexer boards A - F.

2.3.1.8 The FIS Quit Option

To exit the fault injection software, option H should be selected.

2.3.2 Fault Injection Software - Multiple Fault Application

The process of applying one fault and collecting the detection and reconfiguration times was described earlier. Before applying another fault, the AIPS software must return to the state that existed prior to the fault injection. In the context of the I/O Network fault insertion, this process entails the regrowth of the previous I/O virtual path. As a result, the configuration of the I/O nodes and ports is identical to that which existed before the application of the fault. After the I/O network path has been restored, the AIPS software stops executing. When the FIS recognizes that AIPS has stopped, it resumes the execution of the AIPS software and subsequently injects the next fault.

Between the injection of faults into the core FTP, the channel that is corrupted must be brought on-line (from a failed status). This entails realigning its volatile memory to ensure that all channels' internal state is identical. As with the I/O network fault injection, the AIPS software stops after the FTP has returned to the state that existed previous to the fault application and the next fault is inserted.

3.0 Applying Faults to the I/O Network

The I/O Fault Insertion Plan is a set of faults to be injected into the AIPS I/O network which is attached to the AIPS Distributed Engineering Model. The Plan was designed to apply various types of faults that would exercise several network configurations. The following sections describe the AIPS I/O network and the I/O Fault Insertion Plan, respectively.

3.1 Overview of the AIPS I/O Network

For communication between an AIPS FTP and I/O devices, damage and fault-tolerant networks are employed. These AIPS I/O networks are designed to provide both high throughput and high reliability. Each network consists of a number of full duplex links that are interconnected by circuit switched nodes (shown in Figure 3-1). Sensors and actuators are attached to these nodes. In steady-state, the circuit switched nodes route information along a fixed communication path, or "virtual bus", without the delays that are associated with packet-switched networks. Once the virtual bus is set up within the network, the protocols and operation of the network are similar to typical multiplex buses. Although the hardware implementation of this "virtual bus" is a circuit-switched network, the FTP communication and protocol view it as a conventional bus.

Albeit the AIPS I/O network performs exactly like a bus, it is far more reliable and damage tolerant than a linear bus. The network architecture provides coverage for many well known failure modes that would cause a standard linear bus to either fail completely or provide service to a reduced set of subscribers. A single fault or limited damage (for example, caused by weapons, electrical shorts, or overheating) can disable only a small fraction of the virtual bus, typically a node or a link connecting two nodes. The rest of the network, and the subscribers on it, can continue to operate normally. If the sensors and effectors are physically dispersed for damage-tolerance and the damage event does not affect the inherent capability of the vehicle to continue to fly, then the AIPS I/O system would continue to function in a normal manner or in some degraded mode as determined by sensor/effector availability.

The ability of the network to tolerate such faults comes from the design of the node. An AIPS node has five ports, each of which can be enabled or disabled. When the ports on either end of a link are enabled, data is routed along that link of the network. Each node in a properly configured fault-free network receives transmissions on exactly one of its enabled ports and simultaneously retransmits this data on all of its other enabled ports. The nodes provide a richness of spare interconnections that can be brought into service after a hardware fault or damage event occurs.

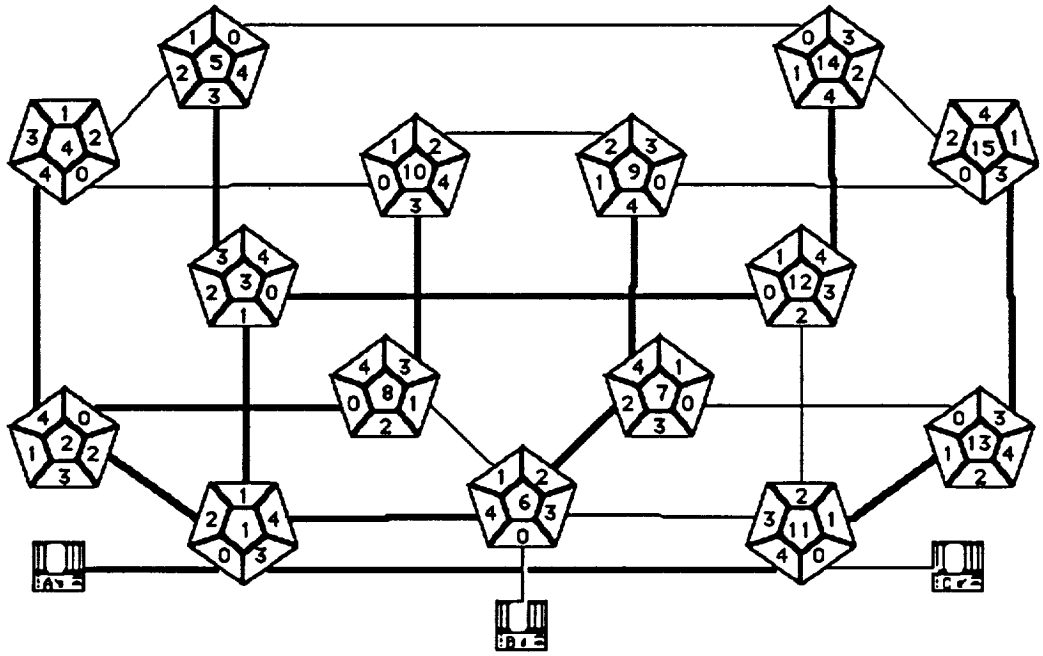

Figure 3-1. 15 Node I/O Network - No Fault Configuration

3.2 Specification of I/O Network Faults

3.2.1 Creating Node and Link Faults

As discussed in Section 3.1, the AIPS I/O network is comprised of nodes and interconnecting full duplex links. Accordingly, the I/O Fault Insertion Plan involves the application of node and link faults.

The AIPS I/O nodes are functionally divided into six sections: Node Sequencer and Control, Port Enable Register, Message Buffer, Port Activity Register, Transmit FIFO, and Protocol Encoder. To corrupt the operation of an I/O node, a fault is applied to one or more of these components. Similarly, each port of the AIPS I/O node is divided into five functional components: Receiver, Protocol Decoder, Clock Extractor, Signal Regeneration Logic, and Transmitter. To interrupt the operation of a link, a fault is applied to a component of the associated node port.

The I/O Fault Insertion Plan only applied faults to some of the node and port subsections, because it was more concerned with a general characterization of the I/O Management process' performance rather than a comprehensive one. An extensive I/O Plan can be developed and applied the AIPS I/O Networks; however, currently it is not being performed.

Four types of node faults were applied, corrupting:

1. The node's control microsequencer, a component of the Node Sequencer and Control subsection that is responsible for decoding and processing the node's microinstructions.
2. The Port Enable Register, a buffer that indicates which node ports are enabled.
3. The I/O FIFO, a component of the Transmit FIFO subsection which buffers the data that is processed and transmitted by the node.
4. The node's address decoding logic, an element of the Node Sequencer and Control system that is responsible for uniquely identifying the node.

Three types of port faults were inserted, affecting:

1. The port's Protocol Decoder that is partly responsible for correctly receiving the input data.
2. The port's input buffer, the part of the Receiver logic that regenerates the input data.
3. The port's input receiver, a component of the Receiver subsection that accepts the input data.

These node and port faults were selected because of their accessibility, their coverage (50 percent of nodes' and 60 percent of the ports' major functional sections were affected), and their complexity (simple, intermediate, and complex error symptoms were generated).

Tables 3-1 and 3-2 detail the node and link faults, specifying how the faults affect the components and the location of the victim hardware elements with respect to the node schematics. (It should be noted here that the terms "link fault" and "port fault" have been used in this document interchangeably.)

3.2.2 Specification of Test Cases

The I/O Fault Insertion Plan is comprised of 47 Tests. Each test applies a fault to one or more nodes of the AIPS 15 Node I/O Network (its topology is illustrated in Figure 3-1). The tests were designed to apply six basic types of topological faults.

A. Failed Link Causing Disjoint Leaf Node
B. Failed Link Causing Disjoint Branch
C. Failed Leaf Node
D. Failed Node Causing Disjoint Branch
E. Failed Root Node
F. Simultaneous Node Failures

Methods to Insert Generic Node Faults	IC Location	Pin Number	Logic Level	Schematic Page #
1. *Corrupt the Node Sequencer* - The node sequencer is reset.	2357	3,12	H	5/6
2. *Corrupt the Port Enable Register* - The input enable of the Port Enable register is corrupted such that arbitrary data is read into the register.	1130	11	H	5/6
3. *Corrupt the I/O FIFO* - The decoder that controls the resetting of the I/O FIFO is corrupted.	2047	4	H	5/6
4. *Corrupt the Address Decoding Logic* - The input enable of the register that outputs the node address is corrupted such that the node address is not placed on the data bus at the correct time.	1130	14	H	5/6

Table 3-1. Victim Node Components - Schematic Location, Pin Number, and Fault Logic Level

A comprehensive I/O fault plan would apply millions of different I/O faults, causing the victim I/O network to reconfigure into thousands of different I/O topologies. The I/O Fault Insertion plan performed by Draper Laboratory only utilized 47 tests, applying faults that caused the 15 node network to reconfigure into 19 degraded topologies. As mentioned earlier, this I/O Fault Insertion Plan was more concerned with a general characterization of the I/O Management process' performance rather than a comprehensive one. Consequently, only some of the vast number of possible tests were performed. The tests completed were selected because of their topological coverage and complexity.

The I/O Fault Insertion Plan is segmented into six fault categories corresponding to the six topological fault types. Each fault category was applied to several network nodes, thereby exercising different I/O configurations. Accordingly, each category is subdivided into groups, each group uniquely identifying and depicting a victim link or node. Furthermore, as previously discussed, each fault type (or fault category) may be instrumented in several ways. Consequently, each group is further segmented into subsections, each indicating the specific fault that was applied.

Methods to Insert Generic Link Faults	IC Location	Pin Number	Logic Level	Schematic Page #
1. *Corrupt the Node's Protocol Decoder*				
- Corruption of the input data counter such that it does not correctly load data.				
Port 0.	1310	7	H	1/6
Port 1.	1110	7	H	1/6
Port 2.	1301	7	H	1/6
Port 3.	1101	7	H	1/6
Port 4.	0901	7	H	1/6
2. *Corrupt the Node's Input Buffer*				
- Corruption of the input data such that the data is stuck-at-one.				
Port 0.	0301	3	H	1/6
Port 1.	0101	11	H	1/6
Port 2.	0101	5	H	1/6
Port 3.	0101	13	H	1/6
Port 4.	0101	3	H	1/6
3. *Corrupt the Node's Input Receiver*				
- The input data FIFO is reset.				
Port 0.	2047	11	L	5/6
Port 1.	2047	12	L	5/6
Port 2.	2047	13	L	5/6
Port 3.	2047	14	L	5/6
Port 4.	2047	15	L	5/6

Table 3-2. Victim Port Components - Schematic Location, Pin Number, and Fault Logic Level

A test is identified by specifying the category, group, and subsection that is associated with the fault that was applied. For example, "test B.2.a" is the test that corrupts the connection between nodes 1 and 2 by disrupting the operation of protocol decoder of port 2 of node 2 (see test outline on the following pages). Alternatively, a test may be identified by indicating the node, integrated circuit, and pin that was victimized. Consequently, "test B.2.a" can also be referred to as "node 2, chip 1301, pin 7" (or simply N2_CH1301_P7).

The following paragraphs detail the I/O Fault Insertion Plan.

A. Failed Link - Disjoint Leaf Node

1. Cause Node 15 to be a Disjoint Leaf Node
   a. Fail the link between nodes 13 and 15 by corrupting the protocol decoder of port 3 of node 15.
      - N15_CH1101_P7
   b. Fail the link between nodes 13 and 15 by corrupting the input data of port 3 of node 15.
      - N15_CH0101_P13
   c. Fail the link between nodes 13 and 15 by corrupting the input receiver of port 3 of node 15.
      - N15_CH2047_P14

2. Cause Node 5 to be a Disjoint Leaf Node
   a. Fail the link between nodes 3 and 5 by corrupting the protocol decoder of port 3 of node 5.
      - N5_CH1101_P7
   b. Fail the link between nodes 3 and 5 by corrupting the input data of port 3 of node 5.
      - N5_CH0101_P13
   c. Fail the link between nodes 3 and 5 by corrupting the input receiver of port 3 of node 5.
      - N5_CH2047_P14

3. Cause Node 14 to be a Disjoint Leaf Node
   a. Fail the link between nodes 12 and 14 by corrupting the protocol decoder of port 4 of node 14.
      - N14_CH0901_P7
   b. Fail the link between nodes 12 and 14 by corrupting the input data of port 4 of node 14.
      - N14_CH0101_P3
   c. Fail the link between nodes 12 and 14 by corrupting the input receiver of port 4 of node 14.
      - N14_CH2047_P15

B. Failed Link - Disjoint Branch

1. Cause Nodes 7 and 9 to be a Disjoint Branch
   a. Fail the link between nodes 6 and 7 by corrupting the protocol decoder of port 2 of node 7.

- N7_CH1301_P7

b. Fail the link between nodes 6 and 7 by corrupting the input data of port 2 of node 7.
   - N7_CH0101_P5

c. Fail the link between nodes 6 and 7 by corrupting the input receiver of port 2 of node 7.
   - N7_CH2047_P13

2. Cause Nodes 2, 4, 8 and 10 to be a Disjoint Branch
   a. Fail the link between nodes 1 and 2 by corrupting the protocol decoder of port 2 of node 2.
      - N2_CH1301_P7
   b. Fail the link between nodes 1 and 2 by corrupting the input data of port 2 of node 2.
      - N2_CH0101_P5
   c. Fail the link between nodes 1 and 2 by corrupting the input receiver of port 2 of node 2.
      - N2_CH2047_P13

3. Cause Nodes 3, 5, 12 and 14 to be a Disjoint Branch
   a. Fail the link between nodes 1 and 3 by corrupting the protocol decoder of port 1 of node 3.
      - N3_CH1101_P7
   b. Fail the link between nodes 1 and 3 by corrupting the input data of port 1 of node 3.
      - N3_CH0101_P11
   c. Fail the link between nodes 1 and 3 by corrupting the input receiver of port 1 of node 3.
      - N3_CH2047_P12

4. Cause all Nodes to be a Disjoint Branch
   a. Fail the link between node 1 and channel A by corrupting the input data of port 0 of node 1.
      - N1_CH0301_P3

C. Failed Leaf Node

1. Fail Node 15 which is a Leaf Node
   a. Fail node 15 by corrupting its control sequencer.
      - N15_CH2357_P12
   b. Fail node 15 by corrupting its port enable register.
      - N15_CH1130_P11
   c. Fail node 15 by corrupting its I/O FIFO.
      - N15_CH2047_P4

d. Fail node 15 by corrupting its node address decoding logic.
   - N15_CH1130_P14

2. Fail Node 4 which is a Leaf Node
   a. Fail node 4 by corrupting its control sequencer.
      - N4_CH2357_P3
   b. Fail node 4 by corrupting its I/O FIFO.
      - N4_CH2047_P4
   c. Fail node 4 by corrupting its node address decoding logic.
      - N4_CH1130_P14

3. Fail Node 10 which is a Leaf Node
   a. Fail node 10 by corrupting its control sequencer.
      - N10_CH2357_P3
   b. Fail node 10 by corrupting its I/O FIFO.
      - N10_CH2047_P4

D. Failed Node - Disjoint Branch

1. Cause Node 9 to be a Disjoint Branch
   a. Fail node 7 by corrupting its control sequencer.
      - N7_CH2357_P3
   b. Fail node 7 by corrupting its port enable register.
      - N7_CH1130_P11
   c. Fail node 7 by corrupting its I/O FIFO.
      - N7_CH2047_P4
   d. Fail node 7 by corrupting its node address decoding logic.
      - N7_CH1130_P14

2. Cause Node 13 and 15 to be a Disjoint Branch
   a. Fail node 11 by corrupting its control sequencer.
      - N11_CH2357_P3
   b. Fail node 11 by corrupting its port enable register.
      - N11_CH1130_P11
   c. Fail node 11 by corrupting its I/O FIFO.
      - N11_CH2047_P4

3. Cause Node 7 and 9 to be a Disjoint Branch
   a. Fail node 6 by corrupting its control sequencer.
      - N6_CH2357_P3
   b. Fail node 6 by corrupting its I/O FIFO.
      - N6_CH2047_P4

4. Cause Node 4 to be a Disjoint Branch and Nodes 8 and 10 to be a Disjoint Branch.
    a. Fail node 2 by corrupting its control sequencer.
        - N2_CH2357_P3
    b. Fail node 2 by corrupting its I/O FIFO.
        - N2_CH2047_P4

5. Cause Node 5 to be a Disjoint Branch and Nodes 12 and 14 to be a Disjoint Branch.
    a. Fail node 3 by corrupting its control sequencer.
        - N3_CH2357_P3
    b. Fail node 3 by corrupting its port enable register.
        - N3_CH1130_P11
    c. Fail node 3 by corrupting its I/O FIFO.
        - N3_CH2047_P4
    d. Fail node 3 by corrupting its node address decoding logic.
        - N3_CH1130_P14

E. Failed Root Node

1. Cause all Nodes to be Disjoint
    a. Fail node 1 by corrupting its control sequencer.
        - N1_CH2357_P3

F. Simultaneous Node Failures

1. Simultaneously Fail Nodes 8 and 10
    a. Fail nodes 8 and 10 by corrupting their control sequencer.
        - N8_10_CH2357_P3

2. Simultaneously Fail Nodes 2 and 8
    a. Fail nodes 2 and 8 by corrupting their control sequencer.
        - N2_10_CH2357_P3

3. Simultaneously Fail Nodes 4 and 10
    a. Fail nodes 4 and 10 by corrupting their control sequencer.
        - N4_10_CH2357_P3

3.3 Test Results

As discussed in Section 3.2, the I/O Fault Insertion Plan is comprised of 47 tests. Each test consists of 25 iterations. An iteration involves the application of a fault, its detection

by the AIPS I/O Network Redundancy Management software, and the associated network reconfiguration. For each iteration, the fault detection and reconfiguration times were recorded by the Fault Insertion Software (FIS). Accordingly, 25 sets of data were recorded for each of the 47 tests (a total of 1175 data sets).

The detection and reconfiguration times recorded during the I/O Fault Insertion are not optimal. These "non-optimal" times result because the I/O Network Redundancy Management software logs debug information during the fault detection, identification, and reconfiguration process. The overhead due to this logging process is less than 0.5 percent of the detection time and less than 10 percent of the reconfiguration time. Accordingly, the times recorded during the I/O Fault Insertion plan are upper bounds.

The I/O Fault Insertion results are segmented into four groups. Section 3.3.1 presents the maximum and average times for each test. Section 3.3.2 details each test's time/frequency histogram. Next, Section 3.3.3 illustrates and discusses the I/O Fault Insertion probability and cumulative density functions. Finally, Section 3.4 provides conclusions for the I/O Fault Insertion plan.

3.3.1 Maximum and Average Times

A fault is detected by the I/O Network Redundancy Management software by requesting and analyzing the status of the network nodes. If all nodes correctly respond to this status query, then the Redundancy Management software assumes that the network is fault-free. If the status response from one or more nodes has errors, then this software presumes that a hardware fault exists and begins the fault identification and reconfiguration stage.

To isolate and reconfigure around a hardware fault, the I/O Network Redundancy Management software examines the status response from the nodes. By analyzing this data and subsequently requested information, the software can determine the location of the fault and activate spare network components to bypass it.

In the AIPS Distributed Engineering Model, the I/O Network Management software checks for the presence of a fault every two seconds (2000 ms.). This fault detection check requires approximately 75 ms. to execute. Consequently, the typical maximum detection time will be about 2075 ms. plus any latency from the time the fault is applied to the time the associated error symptom is generated. (Some exceptions were observed and they are discussed later in this section.) As a result, the typical average detection time is expected to be 1040 ms. plus error latency.

The time necessary to perform the fault isolation and reconfiguration process varies considerably. Simple faults may be identified and bypassed in 100 ms. Alternatively, more complex faults require the regrowth of the I/O network, thereby utilizing thousands of milliseconds to reconfigure the network.

The following paragraphs provide the maximum and average detection and reconfiguration times for each test. Further, the number of faults detected (with respect to the 25 that were inserted) are provided.

A. Failed Link - Disjoint Leaf Node

1. Cause Node 15 to be a Disjoint Leaf Node

   a. Fail the link between nodes 13 and 15 by corrupting the protocol decoder of port 3 of node 15.
      - N15_CH1101_P7

      i. Maximum Detection Time (ms.) 2032.4
      ii. Average Detection Time (ms.) 996.1
      iii. Maximum Reconfiguration Time (ms.) 154.2
      iv. Average Reconfiguration Time (ms.) 147.7
      v. Number of Faults Detected 25

   b. Fail the link between nodes 13 and 15 by corrupting the input data of port 3 of node 15.
      - N15_CH0101_P13

      i. Maximum Detection Time (ms.) 1942.1
      ii. Average Detection Time (ms.) 843.2
      iii. Maximum Reconfiguration Time (ms.) 154.7
      iv. Average Reconfiguration Time (ms.) 147.9
      v. Number of Faults Detected 25

   c. Fail the link between nodes 13 and 15 by corrupting the input receiver of port 3 of node 15.
      - N15_CH2047_P14

      i. Maximum Detection Time (ms.) 2036.6
      ii. Average Detection Time (ms.) 1235.5
      iii. Maximum Reconfiguration Time (ms.) 153.9
      iv. Average Reconfiguration Time (ms.) 145.8
      v. Number of Faults Detected 25

2. Cause Node 5 to be a Disjoint Leaf Node

   a. Fail the link between nodes 3 and 5 by corrupting the protocol decoder of port 3 of node 5.

- N5_CH1101_P7

i. Maximum Detection Time (ms.)	2031.6
ii. Average Detection Time (ms.)	1141.0
iii. Maximum Reconfiguration Time (ms.)	155.5
iv. Average Reconfiguration Time (ms.)	151.2
v. Number of Faults Detected	25

b. Fail the link between nodes 3 and 5 by corrupting the input data of port 3 of node 5.
- N5_CH0101_P13

i. Maximum Detection Time (ms.)	1947.1
ii. Average Detection Time (ms.)	1007.4
iii. Maximum Reconfiguration Time (ms.)	155.7
iv. Average Reconfiguration Time (ms.)	149.2
v. Number of Faults Detected	25

c. Fail the link between nodes 3 and 5 by corrupting the input receiver of port 3 of node 5.
- N5_CH2047_P14

i. Maximum Detection Time (ms.)	2029.2
ii. Average Detection Time (ms.)	946.3
iii. Maximum Reconfiguration Time (ms.)	155.0
iv. Average Reconfiguration Time (ms.)	151.1
v. Number of Faults Detected	25

3. Cause Node 14 to be a Disjoint Leaf Node

a. Fail the link between nodes 12 and 14 by corrupting the protocol decoder of port 4 of node 14.
- N14_CH0901_P7

i. Maximum Detection Time (ms.)	1783.3
ii. Average Detection Time (ms.)	889.3
iii. Maximum Reconfiguration Time (ms.)	154.6
iv. Average Reconfiguration Time (ms.)	150.0
v. Number of Faults Detected	25

b. Fail the link between nodes 12 and 14 by corrupting the input data of port 4 of node 14.
- N14_CH0101_P3

i. Maximum Detection Time (ms.) 2041.5
ii. Average Detection Time (ms.) 1116.3
iii. Maximum Reconfiguration Time (ms.) 176.3
iv. Average Reconfiguration Time (ms.) 142.1
v. Number of Faults Detected 25

c. Fail the link between nodes 12 and 14 by corrupting the input receiver of port 4 of node 14.
- N14_CH2047_P15

i. Maximum Detection Time (ms.) 2039.5
ii. Average Detection Time (ms.) 982.0
iii. Maximum Reconfiguration Time (ms.) 154.8
iv. Average Reconfiguration Time (ms.) 149.9
v. Number of Faults Detected 25

B. Failed Link - Disjoint Branch

1. Cause Nodes 7 and 9 to be a Disjoint Branch

a. Fail the link between nodes 6 and 7 by corrupting the protocol decoder of port 2 of node 7.
- N7_CH1301_P7

i. Maximum Detection Time (ms.) 1914.4
ii. Average Detection Time (ms.) 944.7
iii. Maximum Reconfiguration Time (ms.) 130.8
iv. Average Reconfiguration Time (ms.) 126.7
v. Number of Faults Detected 25

b. Fail the link between nodes 6 and 7 by corrupting the input data of port 2 of node 7.
- N7_CH0101_P5

i. Maximum Detection Time (ms.) 1466.6
ii. Average Detection Time (ms.) 797.8
iii. Maximum Reconfiguration Time (ms.) 214.4
iv. Average Reconfiguration Time (ms.) 130.3
v. Number of Faults Detected 25

c. Fail the link between nodes 6 and 7 by corrupting the input receiver of port 2 of node 7.
- N7_CH2047_P13

i. Maximum Detection Time (ms.) 1326.9
ii. Average Detection Time (ms.) 741.3
iii. Maximum Reconfiguration Time (ms.) 186.5
iv. Average Reconfiguration Time (ms.) 183.6
v. Number of Faults Detected 25

2. Cause Nodes 2, 4, 8 and 10 to be a Disjoint Branch

a. Fail the link between nodes 1 and 2 by corrupting the protocol decoder of port 2 of node 2.
- N2_CH1301_P7

i. Maximum Detection Time (ms.) 2065.5
ii. Average Detection Time (ms.) 1034.5
iii. Maximum Reconfiguration Time (ms.) 196.3
iv. Average Reconfiguration Time (ms.) 187.8
v. Number of Faults Detected 25

b. Fail the link between nodes 1 and 2 by corrupting the input data of port 2 of node 2.
- N2_CH0101_P5

i. Maximum Detection Time (ms.) 1641.5
ii. Average Detection Time (ms.) 811.0
iii. Maximum Reconfiguration Time (ms.) 293.6
iv. Average Reconfiguration Time (ms.) 218.9
v. Number of Faults Detected 25

c. Fail the link between nodes 1 and 2 by corrupting the input receiver of port 2 of node 2.
- N2_CH2047_P13

i. Maximum Detection Time (ms.) 1863.1
ii. Average Detection Time (ms.) 841.9
iii. Maximum Reconfiguration Time (ms.) 226.6
iv. Average Reconfiguration Time (ms.) 221.8
v. Number of Faults Detected 25

3. Cause Nodes 3, 5, 12 and 14 to be a Disjoint Branch

a. Fail the link between nodes 1 and 3 by corrupting the protocol decoder of port 1 of node 3.
   - N3_CH1101_P7

i.	Maximum Detection Time (ms.)	2051.5
ii.	Average Detection Time (ms.)	726.2
iii.	Maximum Reconfiguration Time (ms.)	145.0
iv.	Average Reconfiguration Time (ms.)	137.3
v.	Number of Faults Detected	25

b. Fail the link between nodes 1 and 3 by corrupting the input data of port 1 of node 3.
   - N3_CH0101_P11

i.	Maximum Detection Time (ms.)	2044.4
ii.	Average Detection Time (ms.)	1046.6
iii.	Maximum Reconfiguration Time (ms.)	298.3
iv.	Average Reconfiguration Time (ms.)	220.1
v.	Number of Faults Detected	25

c. Fail the link between nodes 1 and 3 by corrupting the input receiver of port 1 of node 3.
   - N3_CH2047_P12

i.	Maximum Detection Time (ms.)	1869.4
ii.	Average Detection Time (ms.)	758.3
iii.	Maximum Reconfiguration Time (ms.)	230.1
iv.	Average Reconfiguration Time (ms.)	227.4
v.	Number of Faults Detected	25

4. Cause all Nodes to be a Disjoint Branch

a. Fail the link between node 1 and channel A by corrupting the input data of port 0 of node 1.
   - N1_CH0301_P3

i.	Maximum Detection Time (ms.)	2060.0
ii.	Average Detection Time (ms.)	1339.4
iii.	Maximum Reconfiguration Time (ms.)	1090.6
iv.	Average Reconfiguration Time (ms.)	1066.8
v.	Number of Faults Detected	25

C. Failed Leaf Node

1. Fail Node 15 which is a Leaf Node

   a. Fail node 15 by corrupting its control sequencer.
      - N15_CH2357_P12

      i. Maximum Detection Time (ms.) 2272.7
      ii. Average Detection Time (ms.) 1102.8
      iii. Maximum Reconfiguration Time (ms.) 216.4
      iv. Average Reconfiguration Time (ms.) 191.4
      v. Number of Faults Detected 25

      - The fault that was applied to pin 12 of IC 2357 does not manifest itself immediately because a relatively large resistor/capacitor network must first be charged. Consequently, the maximum detection time is exceeded by a few hundred milliseconds.

   b. Fail node 15 by corrupting its port enable register.
      - N15_CH1130_P11

      i. Maximum Detection Time (ms.) 3094.0
      ii. Average Detection Time (ms.) 1142.4
      iii. Maximum Reconfiguration Time (ms.) 1303.1
      iv. Average Reconfiguration Time (ms.) 271.5
      v. Number of Faults Detected 23 (2 Don't Cares)

      - The fault that was applied to pin 11 of IC 1130 causes the port enable register to continuously read a byte from the data bus. As a result, at any given time while the fault is injected, the ports that are enabled (the ports through which data is transmitted) depend on the value of the data bus. Consequently, when this port enable information is read by the control sequencer, it may: (1) correctly depict the ports that should be enabled, (2) falsely specify that one or more ports are enabled when they should be disabled and thus cause the manifestation of a fault, or (3) falsely indicate that disconnected ports are enabled and thereby create a "don't care" condition (again the fault does not manifest itself). If situation (1) or (3) occurs during the I/O Redundancy Management software's initial fault detection check, the software will not see a fault because the fault does not produce an error. Since the fault may not cause visible symptoms, the maximum detection time is not constrained. Furthermore, it is possible that the applied fault is not detected at all (because the fault is only injected for approximately 4 seconds). Conversely, if the fault does cause error symptoms and is detected, the time required to reconfigure the I/O

network may vary considerably because the fault may produce simple, complex, or dynamic error symptoms.

c. Fail node 15 by corrupting its transmit FIFO.
   - N15_CH2047_P4

i. Maximum Detection Time (ms.)	1159.2
ii. Average Detection Time (ms.)	638.9
iii. Maximum Reconfiguration Time (ms.)	211.5
iv. Average Reconfiguration Time (ms.)	208.0
v. Number of Faults Detected	25

d. Fail node 15 by corrupting its node address decoding logic.
   - N15_CH1130_P14

i. Maximum Detection Time (ms.)	1885.0
ii. Average Detection Time (ms.)	997.5
iii. Maximum Reconfiguration Time (ms.)	1459.6
iv. Average Reconfiguration Time (ms.)	1051.2
v. Number of Faults Detected	25

2. Fail Node 4 which is a Leaf Node

a. Fail node 4 by corrupting its control sequencer.
   - N4_CH2357_P3

i. Maximum Detection Time (ms.)	1839.0
ii. Average Detection Time (ms.)	576.2
iii. Maximum Reconfiguration Time (ms.)	220.0
iv. Average Reconfiguration Time (ms.)	213.2
v. Number of Faults Detected	25

b. Fail node 4 by corrupting its transmit FIFO.
   - N4_CH2047_P4

i. Maximum Detection Time (ms.)	1527.3
ii. Average Detection Time (ms.)	682.0
iii. Maximum Reconfiguration Time (ms.)	224.4
iv. Average Reconfiguration Time (ms.)	217.3
v. Number of Faults Detected	25

c. Fail node 4 by corrupting its node address decoding logic.
   - N4_CH1130_P14

i. Maximum Detection Time (ms.) 1711.5
ii. Average Detection Time (ms.) 752.1
iii. Maximum Reconfiguration Time (ms.) 484.0
iv. Average Reconfiguration Time (ms.) 228.4
v. Number of Faults Detected 25

3. Fail Node 10 which is a Leaf Node

a. Fail node 10 by corrupting its control sequencer.
- N10_CH2357_P3

i. Maximum Detection Time (ms.) 2051.9
ii. Average Detection Time (ms.) 1076.9
iii. Maximum Reconfiguration Time (ms.) 215.7
iv. Average Reconfiguration Time (ms.) 214.3
v. Number of Faults Detected 25

b. Fail node 10 by corrupting its transmit FIFO.
- N10_CH2047_P4

i. Maximum Detection Time (ms.) 1315.7
ii. Average Detection Time (ms.) 804.8
iii. Maximum Reconfiguration Time (ms.) 491.7
iv. Average Reconfiguration Time (ms.) 227.8
v. Number of Faults Detected 25

D. Failed Node - Disjoint Branch

1. Cause Node 9 to be a Disjoint Branch

a. Fail node 7 by corrupting its control sequencer.
- N7_CH2357_P3

i. Maximum Detection Time (ms.) 1928.7
ii. Average Detection Time (ms.) 718.6
iii. Maximum Reconfiguration Time (ms.) 255.9
iv. Average Reconfiguration Time (ms.) 252.4
v. Number of Faults Detected 25

b. Fail node 7 by corrupting its port enable register.
- N7_CH1130_P11

i. Maximum Detection Time (ms.) 3372.9
ii. Average Detection Time (ms.) 1088.7
iii. Maximum Reconfiguration Time (ms.) 2869.5
iv. Average Reconfiguration Time (ms.) 478.6
v. Number of Faults Detected 22 (3 Don't Cares)

- See Test C.1.b for explanation concerning the exceptionally large maximum detection and reconfiguration times and the "don't care" conditions.

c. Fail node 7 by corrupting its transmit FIFO.
- N7_CH2047_P4

i.	Maximum Detection Time (ms.)	4009.5
ii.	Average Detection Time (ms.)	1177.1
iii.	Maximum Reconfiguration Time (ms.)	969.4
iv.	Average Reconfiguration Time (ms.)	310.5
v.	Number of Fault Detected	25

IC 2047 is a decoder that controls the resetting of the node's input receivers and the write enable for the transmit FIFO. The insertion of the fault into pin 4 is such that the decoder IC is not enabled and none of its outputs are selected. Therefore no new data can be written into the transmit FIFO. Nevertheless, the transmit FIFO can still transmit data. If the data in the transmit FIFO corresponds to a valid node status response, the I/O Redundancy Management software may not detect the fault during its initial faul check (a valid response may be transmitted to the FTP). Accordingly, as was observed, the maximum detection time may exceed the expected maximum time.

d. Fail node 7 by corrupting its node address decoding logic.
- N7_CH1130_P14

i. Maximum Detection Time (ms.) 1873.9
ii. Average Detection Time (ms.) 1131.8
iii. Maximum Reconfiguration Time (ms.) 1239.4
iv. Average Reconfiguration Time (ms.) 1117.9
v. Number of Faults Detected 25

2. Cause Node 13 and 15 to be a Disjoint Branch

a. Fail node 11 by corrupting its control sequencer.
- N11_CH2357_P3

i. Maximum Detection Time (ms.) 2054.0
ii. Average Detection Time (ms.) 983.2
iii. Maximum Reconfiguration Time (ms.) 296.1
iv. Average Reconfiguration Time (ms.) 289.0
v. Number of Faults Detected 25

b. Fail node 11 by corrupting its port enable register.
- N11_CH1130_P11

i. Maximum Detection Time (ms.) 1906.4
ii. Average Detection Time (ms.) 1171.7
iii. Maximum Reconfiguration Time (ms.) 2206.4
iv. Average Reconfiguration Time (ms.) 1037.8
v. Number of Faults Detected 25

- See Test C.1.b for explanation concerning the exceptionally large maximum reconfiguration time.

c. Fail node 11 by corrupting its transmit FIFO.
- N11_CH2047_P4

i. Maximum Detection Time (ms.) 2053.1
ii. Average Detection Time (ms.) 1013.2
iii. Maximum Reconfiguration Time (ms.) 1070.2
iv. Average Reconfiguration Time (ms.) 528.3
v. Number of Faults Detected 25

3. Cause Node 7 and 9 to be a Disjoint Branch

a. Fail node 6 by corrupting its control sequencer.
- N6_CH2357_P3

i. Maximum Detection Time (ms.) 2051.0
ii. Average Detection Time (ms.) 721.9
iii. Maximum Reconfiguration Time (ms.) 378.6
iv. Average Reconfiguration Time (ms.) 295.6
v. Number of Faults Detected 25

b. Fail node 6 by corrupting its transmit FIFO.
- N6_CH2047_P4

i. Maximum Detection Time (ms.) 1954.4
ii. Average Detection Time (ms.) 799.3
iii. Maximum Reconfiguration Time (ms.) 381.7
iv. Average Reconfiguration Time (ms.) 377.1
v. Number of Faults Detected 25

4. Cause Node 4 to be a Disjoint Branch and Nodes 8 and 10 to be a Disjoint Branch.

a. Fail node 2 by corrupting its control sequencer.
- N2_CH2357_P3

i. Maximum Detection Time (ms.) 1519.8
ii. Average Detection Time (ms.) 727.0
iii. Maximum Reconfiguration Time (ms.) 465.5
iv. Average Reconfiguration Time (ms.) 244.4
v. Number of Faults Detected 25

b. Fail node 2 by corrupting its transmit FIFO.
- N2_CH2047_P4

i. Maximum Detection Time (ms.) 1959.7
ii. Average Detection Time (ms.) 674.1
iii. Maximum Reconfiguration Time (ms.) 292.3
iv. Average Reconfiguration Time (ms.) 288.9
v. Number of Faults Detected 25

5. Cause Node 5 to be a Disjoint Branch and Nodes 12 and 14 to be a Disjoint Branch.

a. Fail node 3 by corrupting its control sequencer.
- N3_CH2357_P3

i. Maximum Detection Time (ms.) 2063.2
ii. Average Detection Time (ms.) 887.8
iii. Maximum Reconfiguration Time (ms.) 296.4
iv. Average Reconfiguration Time (ms.) 262.5
v. Number of Faults Detected 25

b. Fail node 3 by corrupting its port enable register.
- N3_CH1130_P11

i. Maximum Detection Time (ms.) 2794.4

ii. Average Detection Time (ms.) 1035.3
iii. Maximum Reconfiguration Time (ms.) 2897.3
iv. Average Reconfiguration Time (ms.) 1503.6
v. Number of Faults Detected 25

- See Test C.1.b for explanation concerning the exceptionally large maximum detection and reconfiguration times.

c. Fail node 3 by corrupting its transmit FIFO.
- N3_CH2047_P4

i. Maximum Detection Time (ms.) 2050.2
ii. Average Detection Time (ms.) 739.8
iii. Maximum Reconfiguration Time (ms.) 324.8
iv. Average Reconfiguration Time (ms.) 288.1
v. Number of Faults Detected 25

d. Fail node 3 by corrupting its node address decoding logic.
- N3_CH1130_P14

i. Maximum Detection Time (ms.) 2002.6
ii. Average Detection Time (ms.) 808.8
iii. Maximum Reconfiguration Time (ms.) 557.6
iv. Average Reconfiguration Time (ms.) 353.1
v. Number of Faults Detected 25

E. Failed Root Node

1. Cause all Nodes to be Disjoint

a. Fail node 1 by corrupting its control sequencer.
- N1_CH2357_P3

i. Maximum Detection Time (ms.) 2032.0
ii. Average Detection Time (ms.) 928.5
iii. Maximum Reconfiguration Time (ms.) 400.4
iv. Average Reconfiguration Time (ms.) 390.1
v. Number of Faults Detected 25

F. Simultaneous Node Failures

1. Simultaneously Fail Nodes 8 and 10

a. Fail nodes 8 and 10 by corrupting their control sequencer.
   - N8_10_CH2357_P3

i.	Maximum Detection Time (ms.)	2046.8
ii.	Average Detection Time (ms.)	822.8
iii.	Maximum Reconfiguration Time (ms.)	461.2
iv.	Average Reconfiguration Time (ms.)	458.8
v.	Number of Faults Detected	25

2. Simultaneously Fail Nodes 2 and 8

   a. Fail nodes 2 and 8 by corrupting their control sequencer.
      - N2_10_CH2357_P3

i.	Maximum Detection Time (ms.)	2055.3
ii.	Average Detection Time (ms.)	1283.7
iii.	Maximum Reconfiguration Time (ms.)	1237.2
iv.	Average Reconfiguration Time (ms.)	1078.4
v.	Number of Faults Detected	25

3. Simultaneously Fail Nodes 4 and 10
   a. Fail nodes 4 and 10 by corrupting their control sequencer.
      - N4_10_CH2357_P3

i.	Maximum Detection Time (ms.)	2060.5
ii.	Average Detection Time (ms.)	1523.9
iii.	Maximum Reconfiguration Time (ms.)	997.4
iv.	Average Reconfiguration Time (ms.)	992.5
v.	Number of Faults Detected	25

3.3.2 Frequency Histograms

Ideally, for each fault insertion test, the fault detection and reconfiguration times for each iteration of the test should be constant. That is, each inserted fault should be detected in X ms. and reconfigured around in Y ms. Such consistency will occur only if the I/O fault detection check is performed frequently and each fault produces invariable error symptoms. Nonetheless, since the I/O Redundancy Management software only runs every two seconds in the Engineering Model, large variations may exist in the detection times. Further, it is possible, and likely, that multiple insertions of a given fault cause different error symptoms. Consequently, the time required to reconfigure around one application of the fault may be Y ms. while the time necessary to bypass another insertion of the same fault may be Y + Z ms. As a result, histograms of the detection and reconfiguration times are

valuable because they illustrate the repeatability of the I/O Fault Insertion and I/O Redundancy Management processes.

Figures 3-2 through 3-48 present the frequency histograms for each test. Each illustration is comprised of two graphs. The upper graph represents the fault detection distribution while the lower one shows the distribution of the reconfiguration times. The detection and reconfiguration times were grouped into "buckets" which are collections of times that fall in a particular range. The times associated with each bucket are indicated on the horizontal axis of each graph. The "frequency distribution" or the number of entries in each bucket is depicted on the vertical axis of each graph.

The I/O fault detection and reconfiguration histograms are detailed in Sections 3.3.2.1 and 3.3.2.2, respectively.

3.3.2.1 Variance of the Detection Times

As mentioned earlier, the I/O Redundancy Management software performs a fault detection check every two seconds. The I/O Fault Insertion Software is executed such that the interval between the successive applications of a given fault varies. Consequently, the time at which the fault is inserted, with respect to the I/O fault detection check, differs between successive iterations of the test. If the fault is injected into the I/O network just prior to the check and it manifests as an error before the detection check, then it will be detected in a few hundred milliseconds. Alternatively, if the fault is applied just after the completion of the check and it manifests as an error before the next detection check, then approximately two seconds will be required to detect it. Faults, for which error manifestation latency is just over 2 seconds or larger, would be detected by subsequent detection checks.

Since the detection cycle is two seconds and the relative fault insertion times differ from iteration to iteration, it was expected that a given test's detection times would vary considerably. As anticipated, large time variances were observed, and they are portrayed in the fault detection histograms which are shown in Figures 3-2 though 3-48.

3.3.2.2 Variance of the Reconfiguration Times

As discussed in Section 3.3.2, the application of a given fault may or may not produce consistent error symptoms. This inconsistency occurs because the fault insertion times vary relative to the execution of the detection cycle and the activity of the I/O nodes. If repeatable error symptoms are generated by the fault, then the reconfiguration times will be grouped into a few contiguous buckets (for example, as shown in Figure 3-2). If the error symptoms produced vary, then the histogram may have widely differing times and accordingly disjoint groupings (for instance, a primary cluster and an exception as depicted in Figure 3-12). These different reconfiguration times result because the I/O Reconfiguration Management software traverses different code paths.

The variance in the reconfiguration times, however, may also result if one or more software errors exist in the I/O Redundancy Management process. As a result, it was desirable to analyze the results of the I/O fault tests with disjoint buckets in order to confirm the correctness of the I/O Management software. This analysis was performed by modifying the Fault Insertion Software (FIS), such that it suspends when a reconfiguration time significantly differs from the median bucket (for instance, if this difference was greater than 50 ms.). After the FIS stops, the I/O Redundancy trace data, which indicates the path of the reconfiguration software, of this atypical iteration is compared to that recorded after a normal reconfiguration. By comparing the traces, the abnormal error signature and resultant reconfiguration path were determined.

The "normal vs. abnormal" results were carefully examined and in each case, the I/O Redundancy Management software executed correctly. The atypical reconfiguration times resulted because of one of four causes: second attempts, inconclusive analysis, presumed reconnection, and fault signatures of differing complexities. These situations are discussed in the Sections 3.3.2.2.1 through 3.3.2.2.4.

3.3.2.2.1 Reconfiguration Variance - Second Reconfiguration Attempts

During the reconfiguration process, typically only one attempt is necessary to activate a non-faulty path (enable a link between two nodes by actuating the corresponding ports). Nevertheless, it is possible that a non-faulty path returns an erroneous response to the I/O network management software due to the breadboard nature of the laboratory nodes. Therefore, in the process of enabling a link, two attempts are performed. That is, if the first attempt to enable a non-faulty link fails, then a second attempt is tried. If both attempts are unsuccessful, then the link is deemed faulty.

Occasionally, during the reconfiguration process, second attempts were performed. Because of these second attempts, 80 - 90 ms. variations from the median buckets were observed. The histograms displaying such occurrences are illustrated in Figures 3-9, 3-12, 3-15, 3-18, 3-37, 3-43, and 3-47.

3.3.2.2.2 Reconfiguration Variance - Presumed Reconnection

The breadboard nature of the laboratory nodes also contributed to inconsistency in the "effect" of the applied I/O fault. Specifically, the application of a fault that should completely fail a node occasionally only failed its inboard port (the active port through which the node is connected to the I/O virtual path). In such a situation, the I/O Redundancy Management software reconnected to the node, or the associated disjoint branch, through a spare link.

If the I/O reconfiguration process reconnects to a node rather than fail it completely (because the application of the fault does not completed disable the node), then less code is traversed and accordingly, the reconfiguration time is shorter. Consequently, the associated reconfiguration histograms are signified by aberrations that are a negative offsets from the median bucket. These histograms are depicted in Figures 3-21, 3-32, 3-34, 3-39, 3-41, 3-43, and 3-45.

3.3.2.2.3 Reconfiguration Variance - Inconclusive Analysis

The I/O reconfiguration process can only isolate a finite number of fault signatures. If an unidentifiable fault pattern is encountered, then the I/O reconfiguration process regrows the I/O network using one of several I/O growth algorithms.

During the I/O Fault Insertion Analysis, it was observed that a given fault, that usually generated a distinguishable error pattern, sometimes produced a fault signature that was unknown to the I/O Redundancy Management software. When this indeterminate pattern was encountered, the I/O network was regrown (using an algorithm employing minimal diagnostics), thus adding approximately 270 - 280 ms. to the reconfiguration time. The histograms of the tests in which such a regrowth occurred are shown in Figures 3-27, 3-29, 3-39, and 3-44.

3.3.2.2.4 Reconfiguration Variance - Simple and Complex Error Symptoms

The error symptoms that are produced by a fault may have simple or complex signatures. Further, the fault pattern may be time-varying (for example, the corruption of the port enable register). If simple error symptoms occur, then the fault can be isolated and bypassed quickly. Alternatively, if complex symptoms are produced by the fault, then the regrowth of the I/O network may be required. In addition, if the symptoms vary with time, then the I/O redundancy management process may perform multiple regrowth attempts (because the error signature changes during the growth process).

The histograms of tests in which simple, complex, and time-varying fault signatures occur are shown in Figures 3-20, 3-22, 3-24, 3-31, 3-32, 3-35, 3-36, and 3-42.

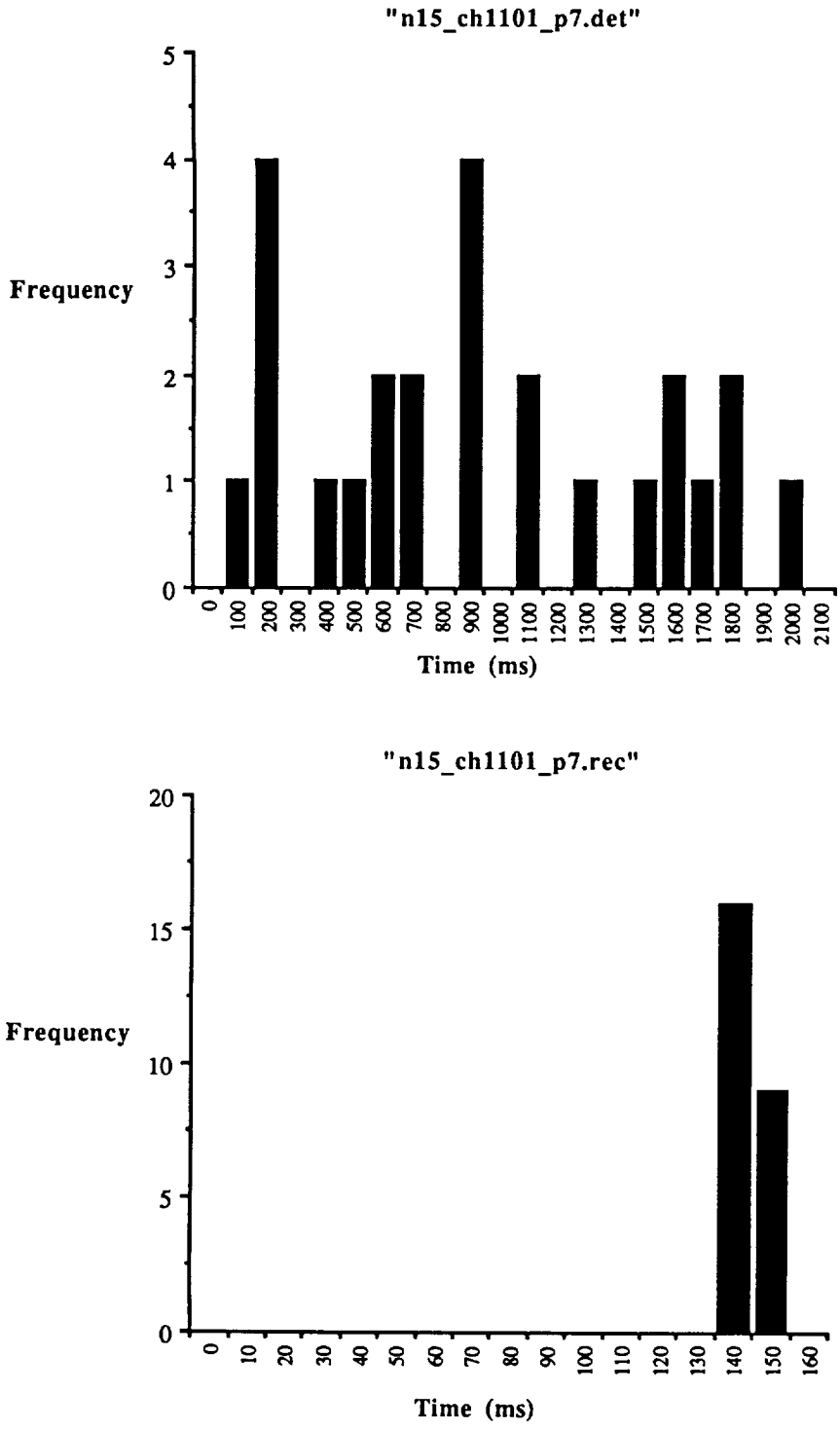

Figure 3-2. Test A.1.a

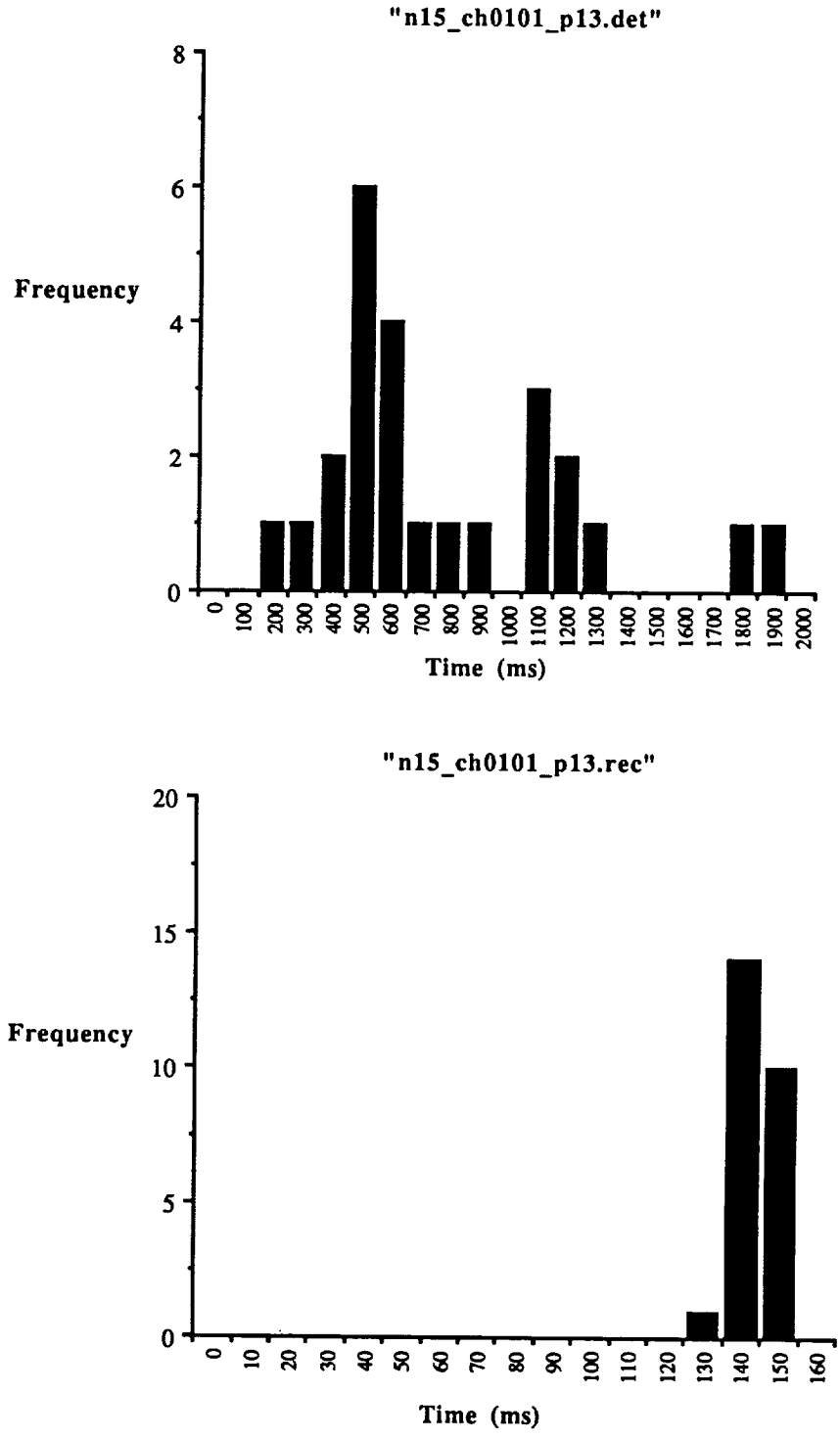

Figure 3-3. Test A.1.b

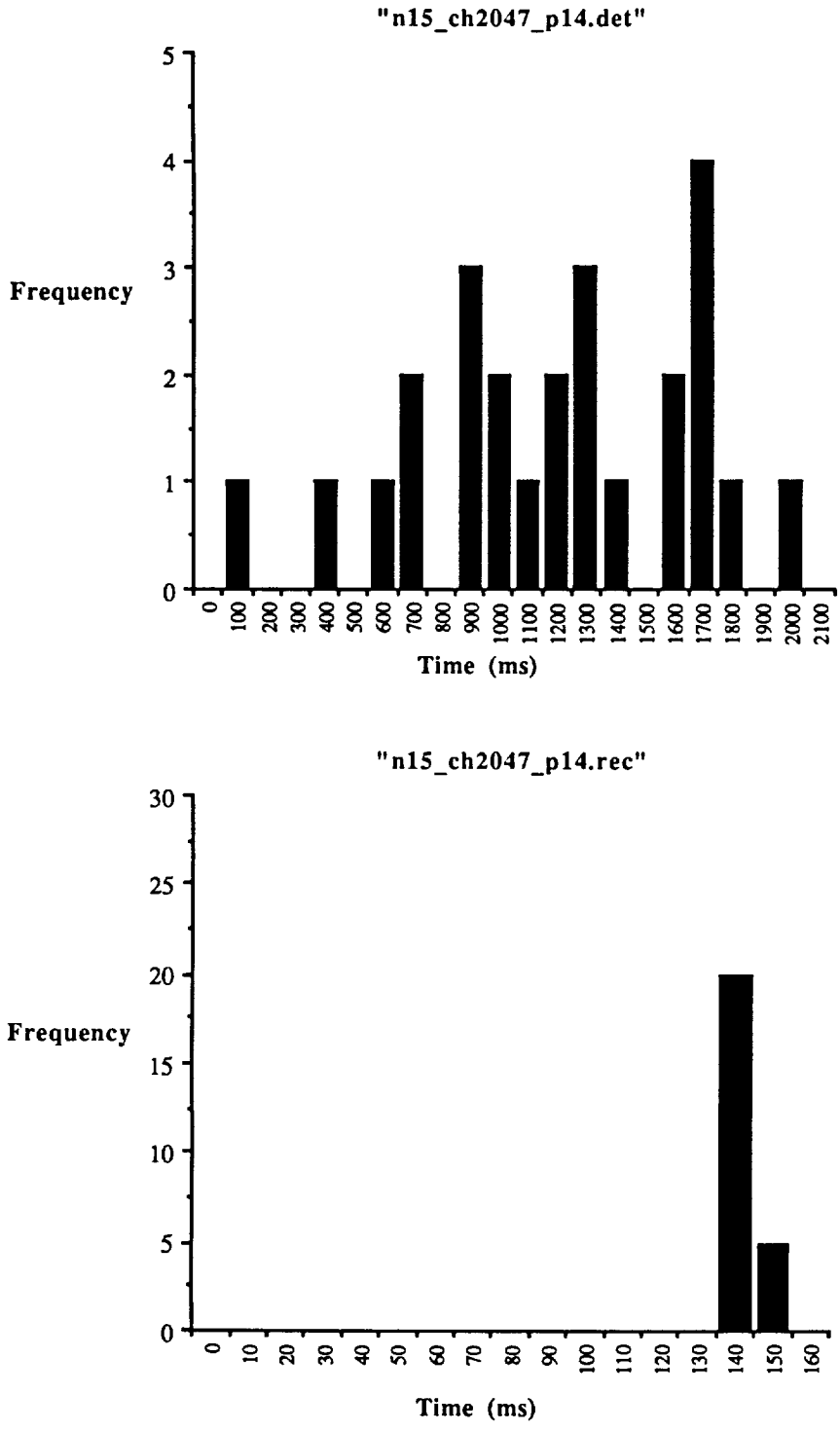

Figure 3-4. Test A.1.c

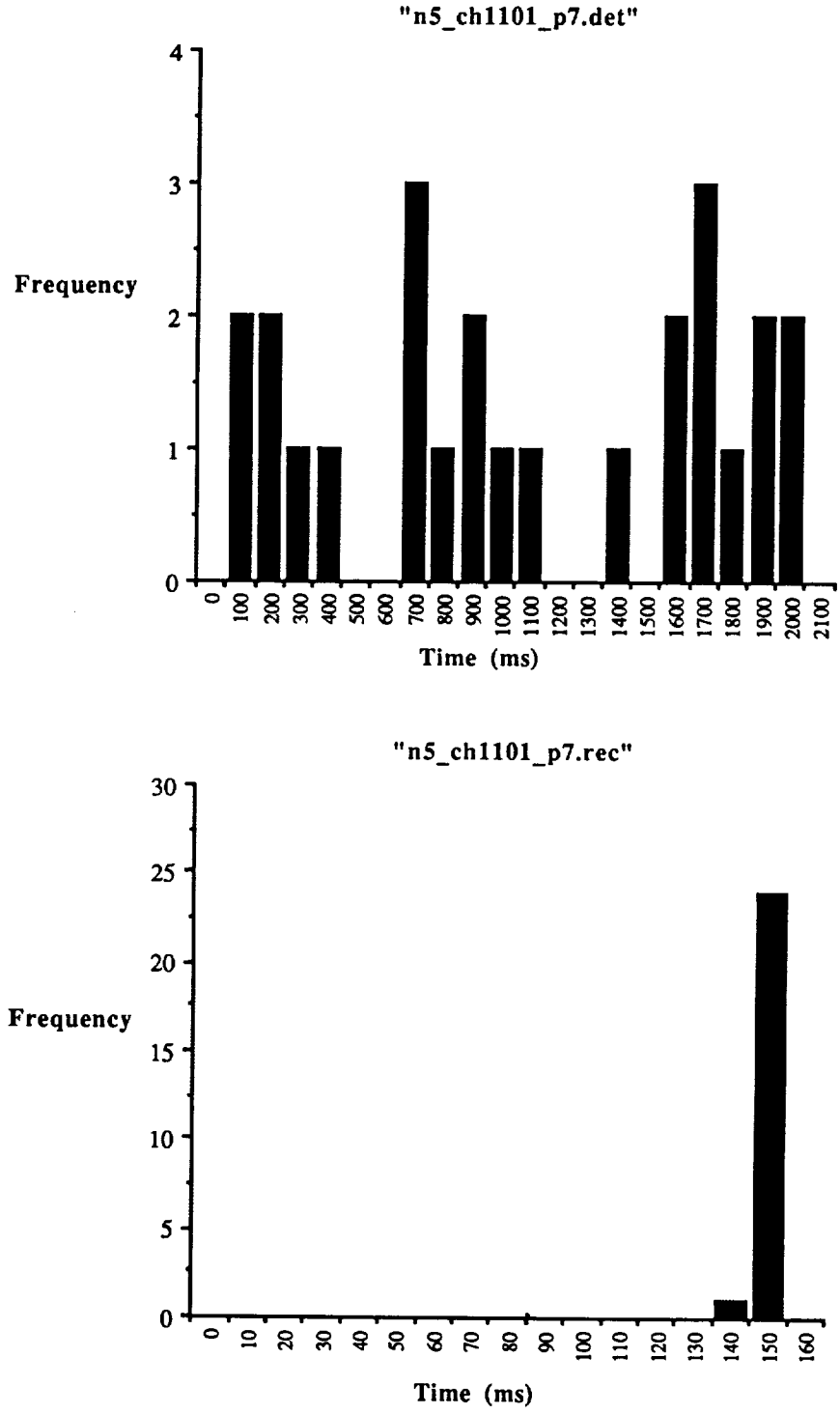

Figure 3-5. Test A.2.a

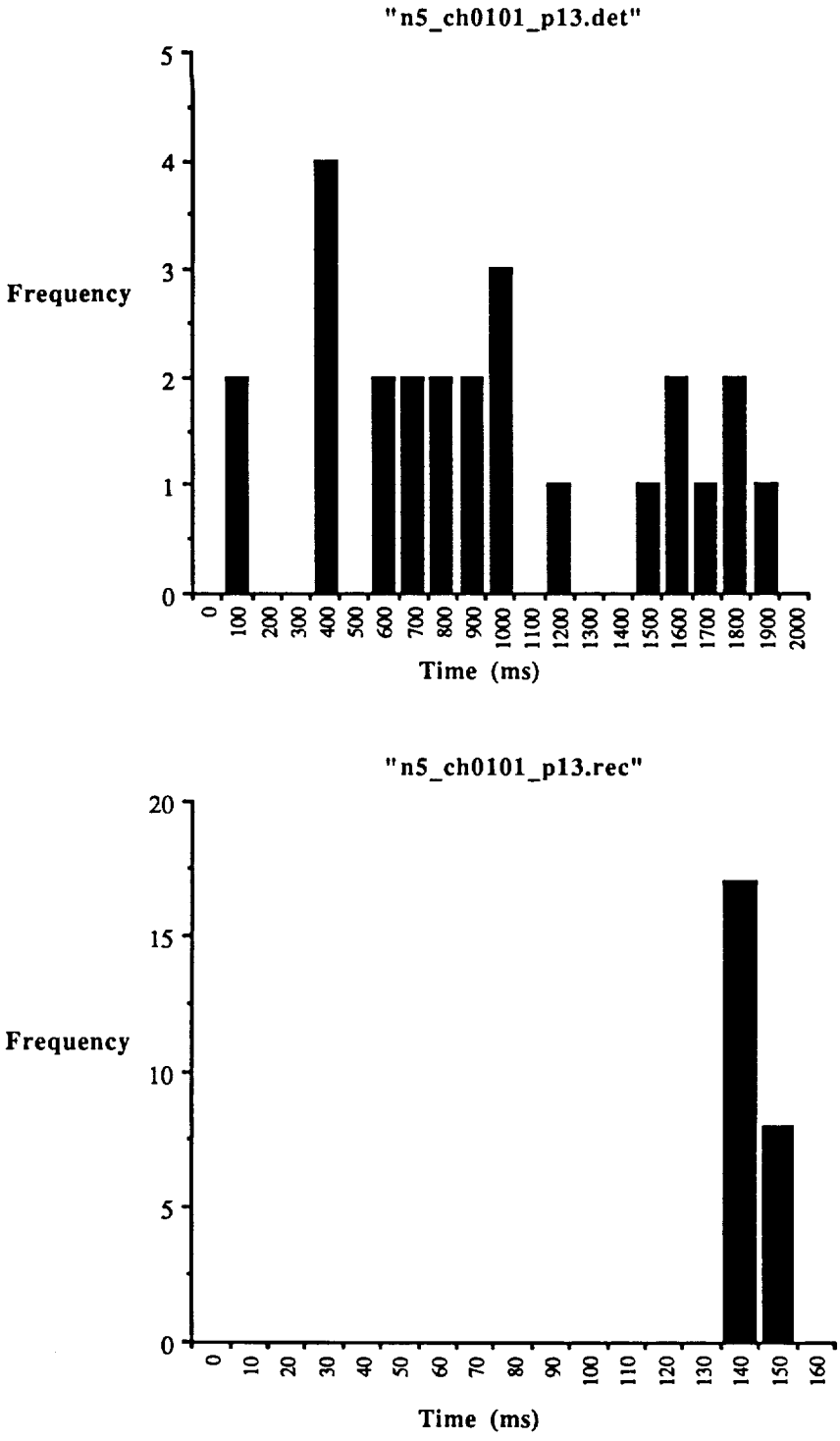

Figure 3-6. Test A.2.b

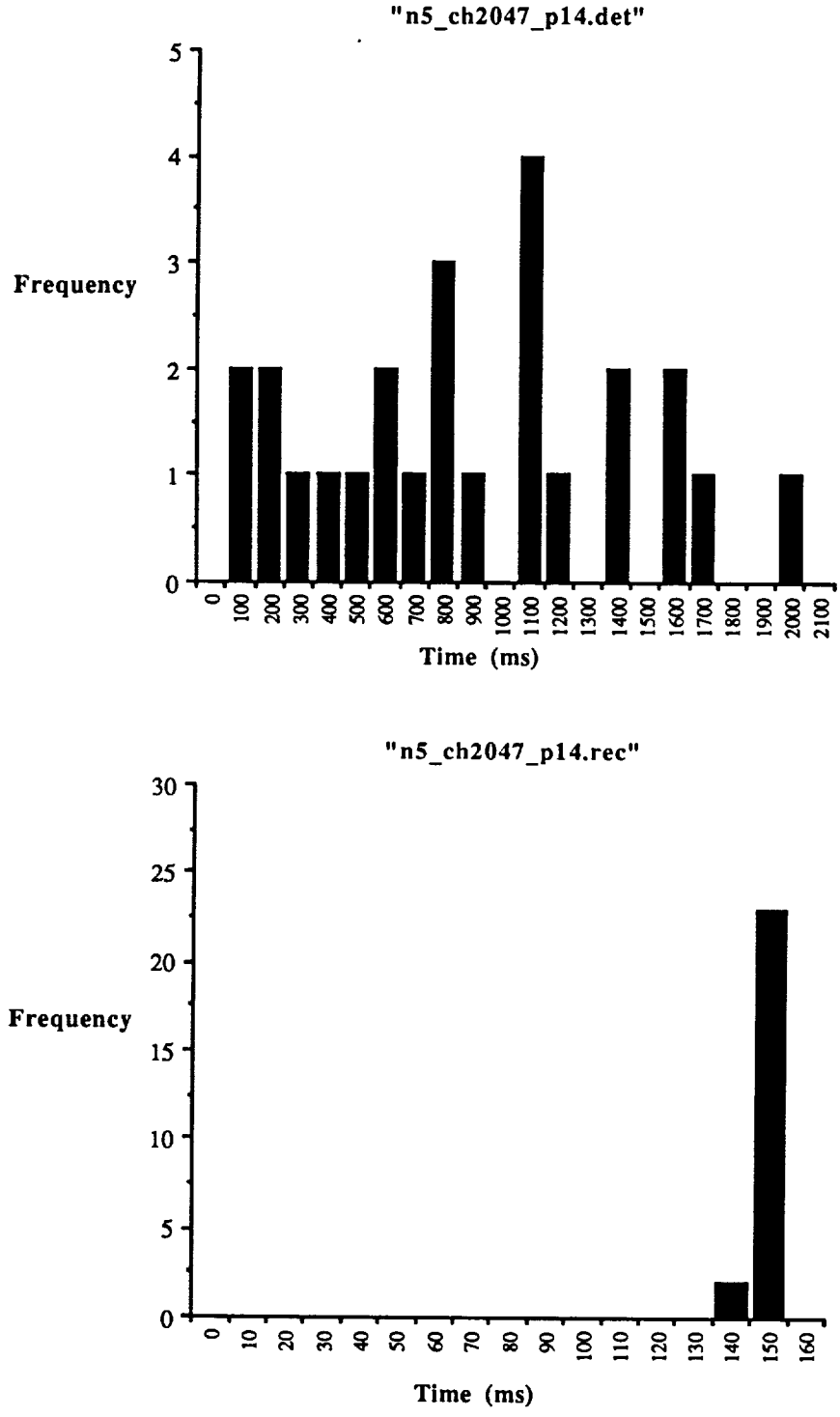

Figure 3-7. Test A.2.c

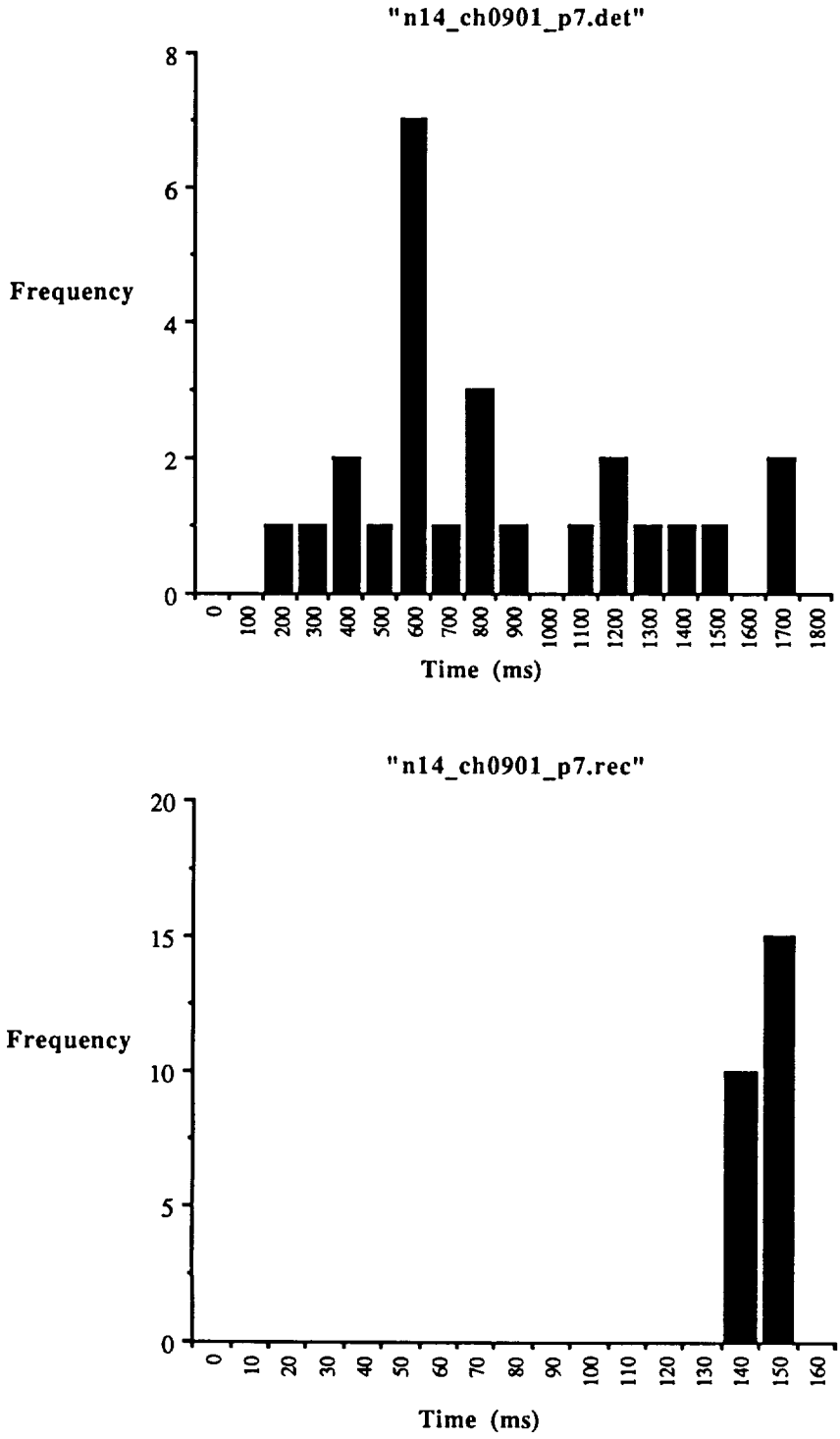

Figure 3-8. Test A.3.a

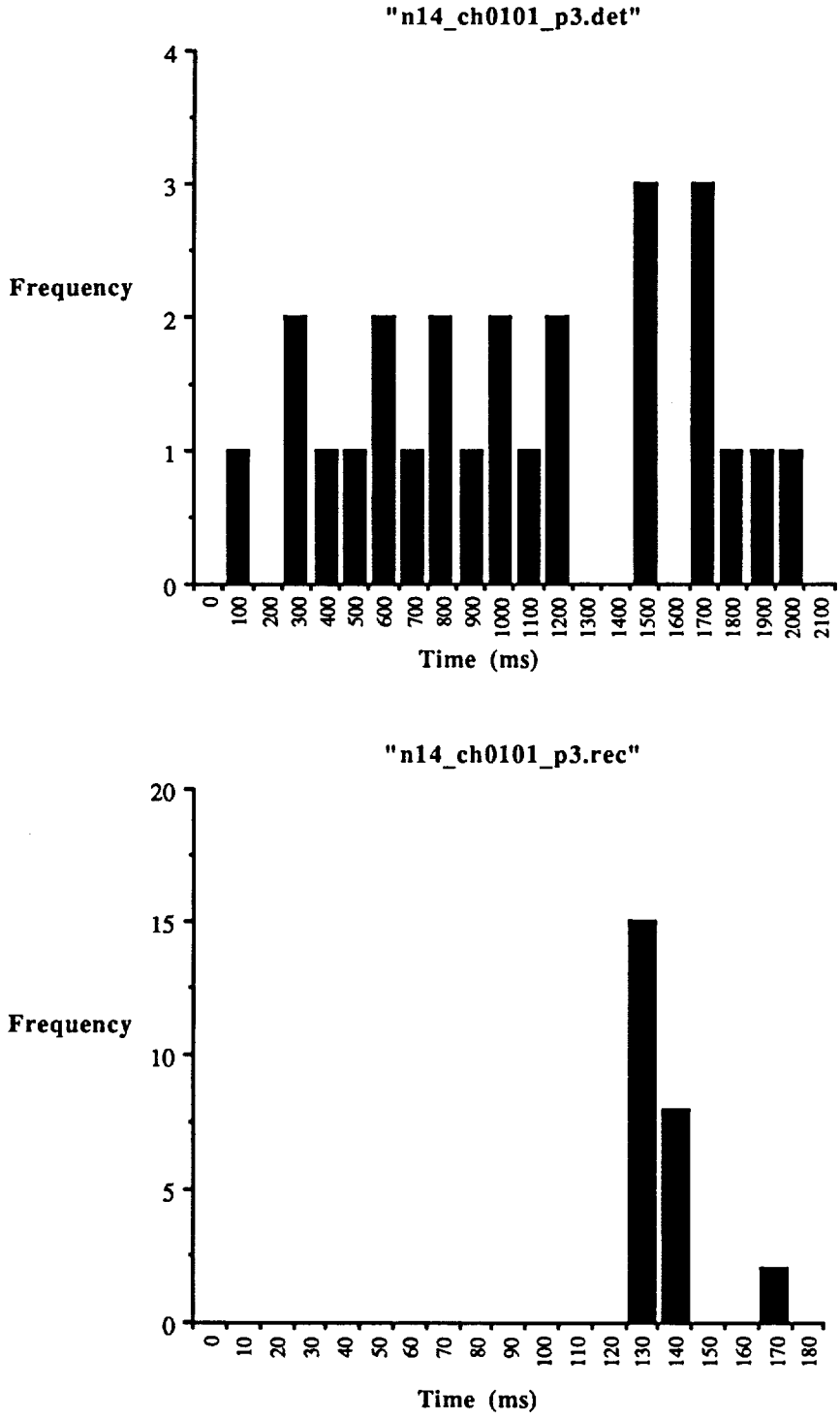

Figure 3-9. Test A.3.b

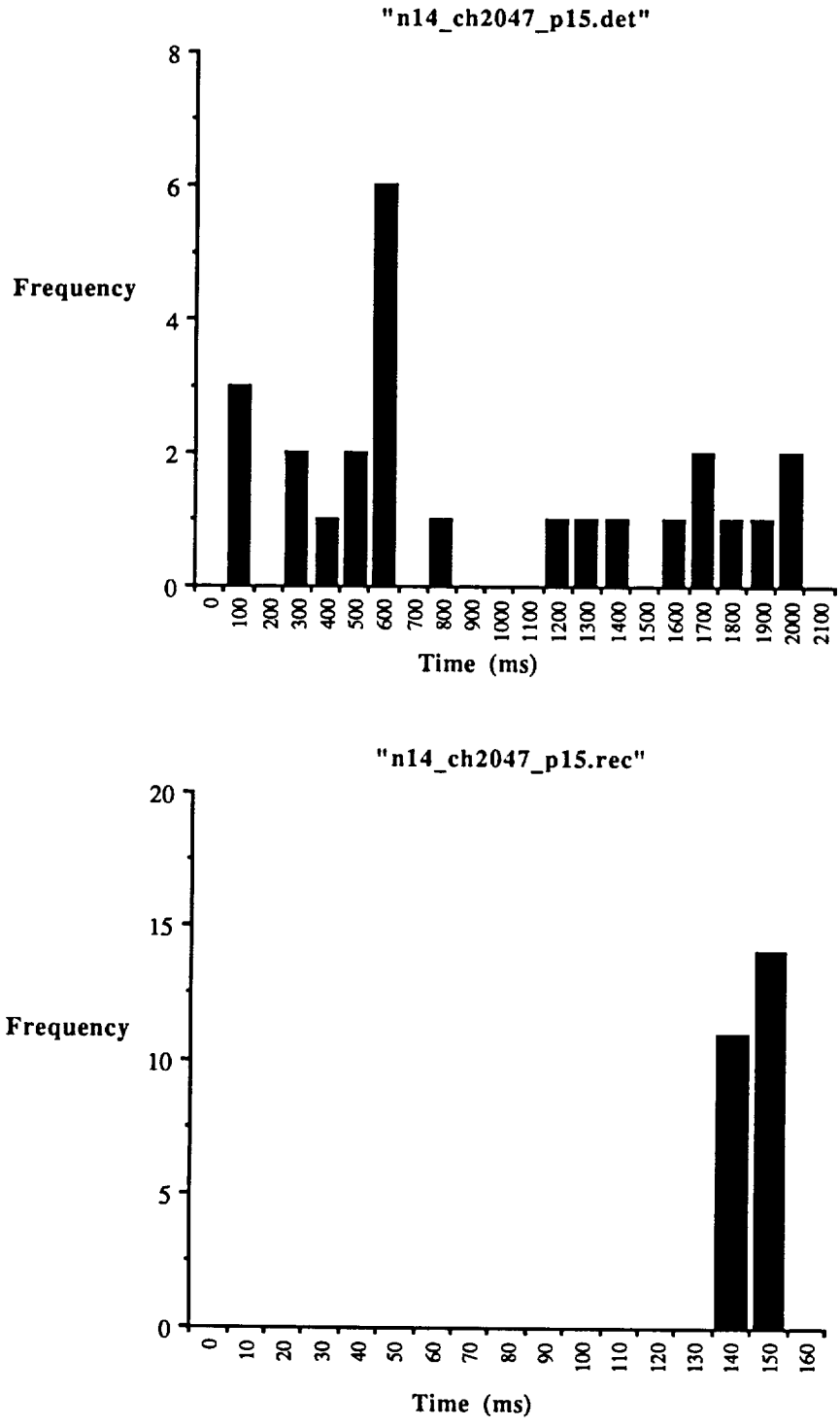

Figure 3-10. Test A.3.c

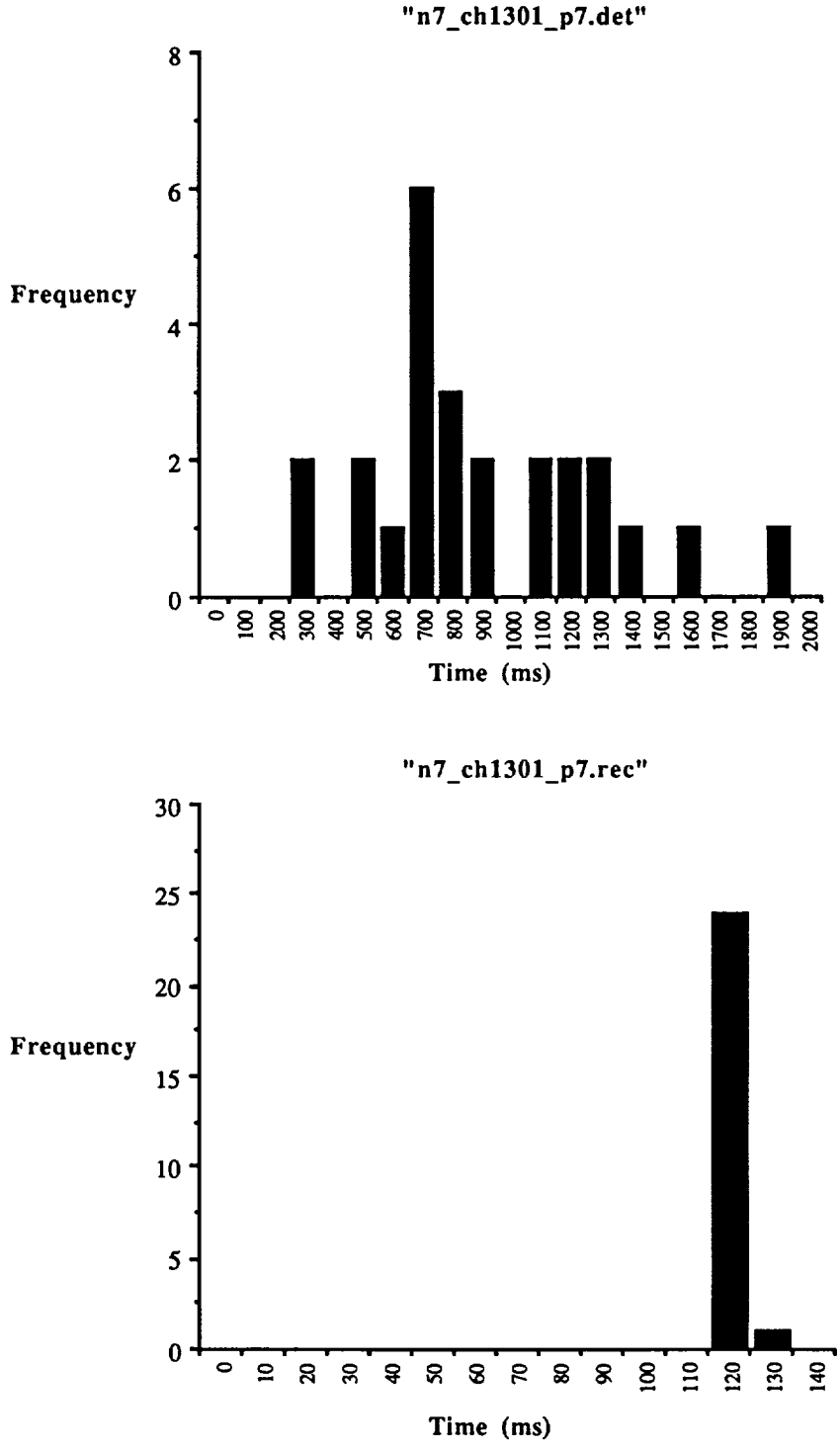

Figure 3-11. Test B.1.a

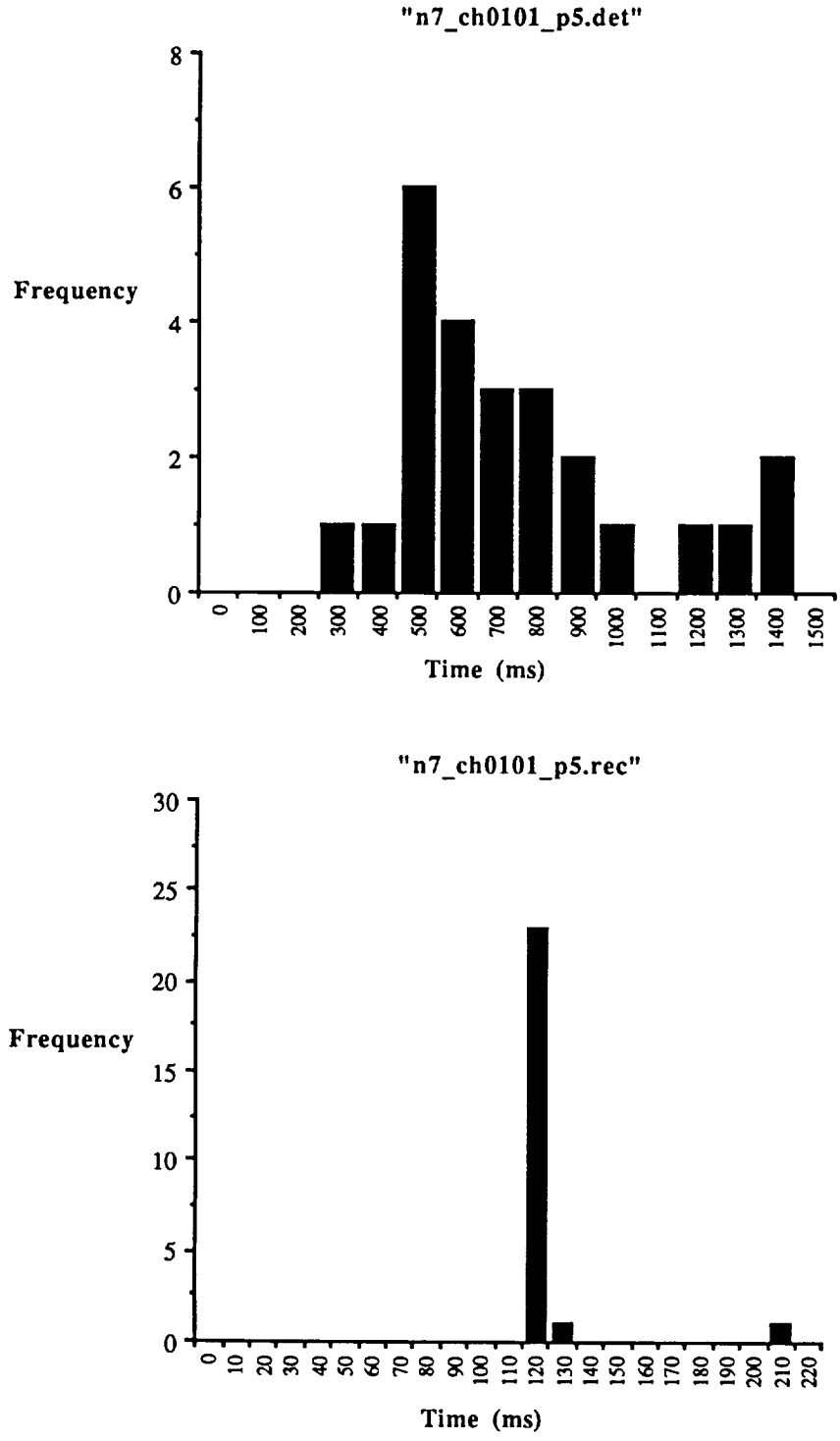

Figure 3-12. Test B.1.b

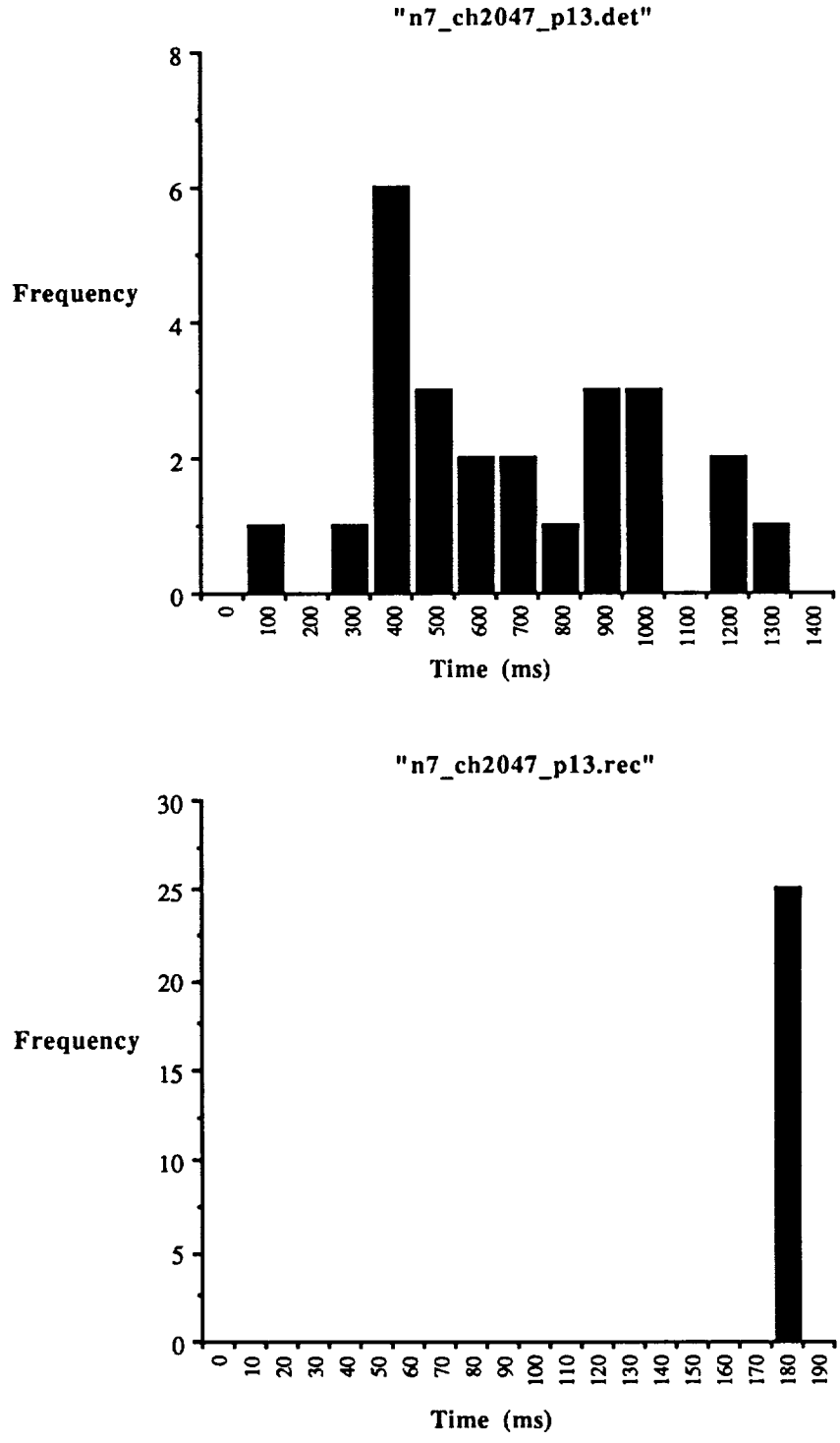

Figure 3-13. Test B.1.c

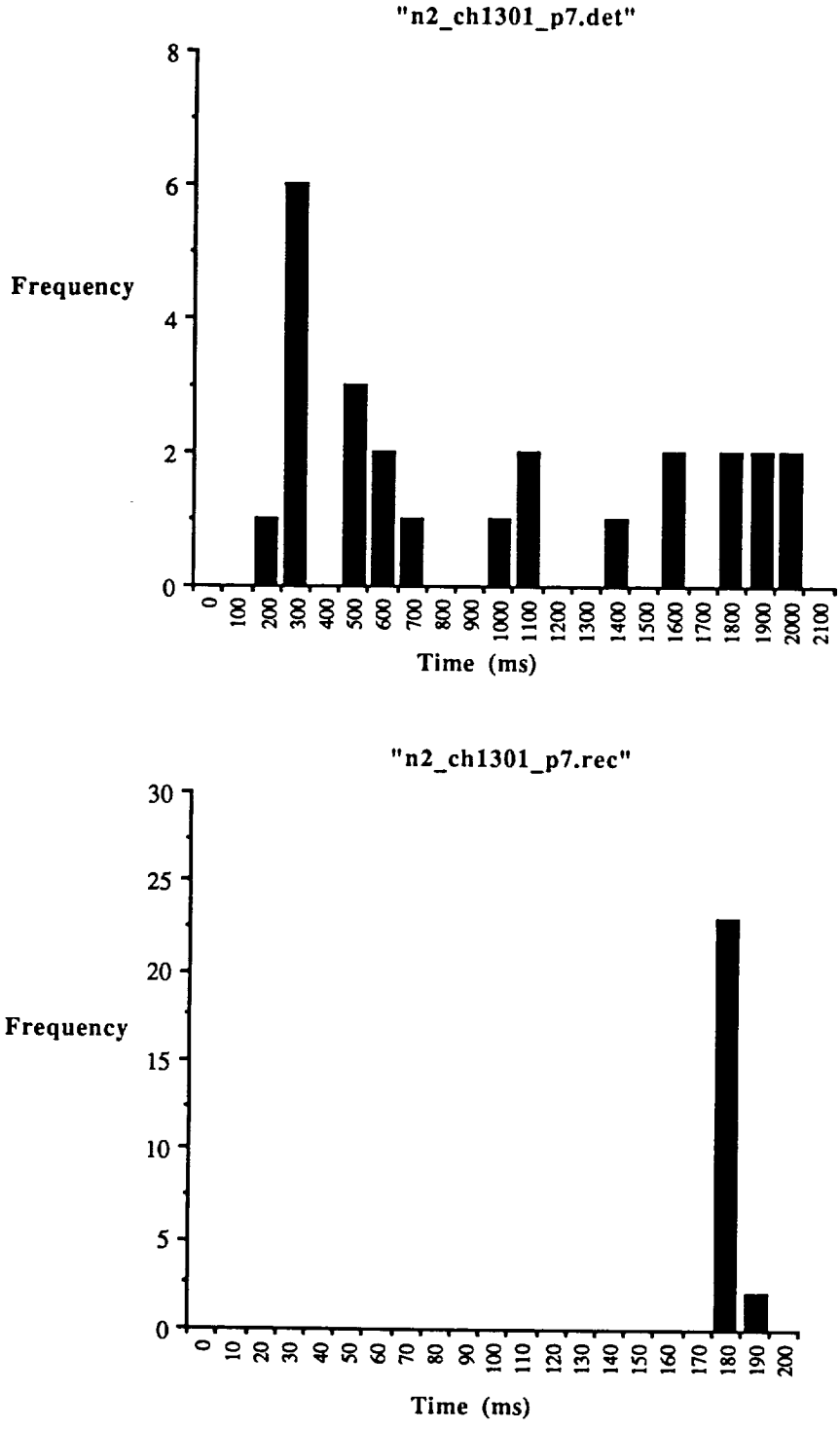

Figure 3-14. Test B.2.a

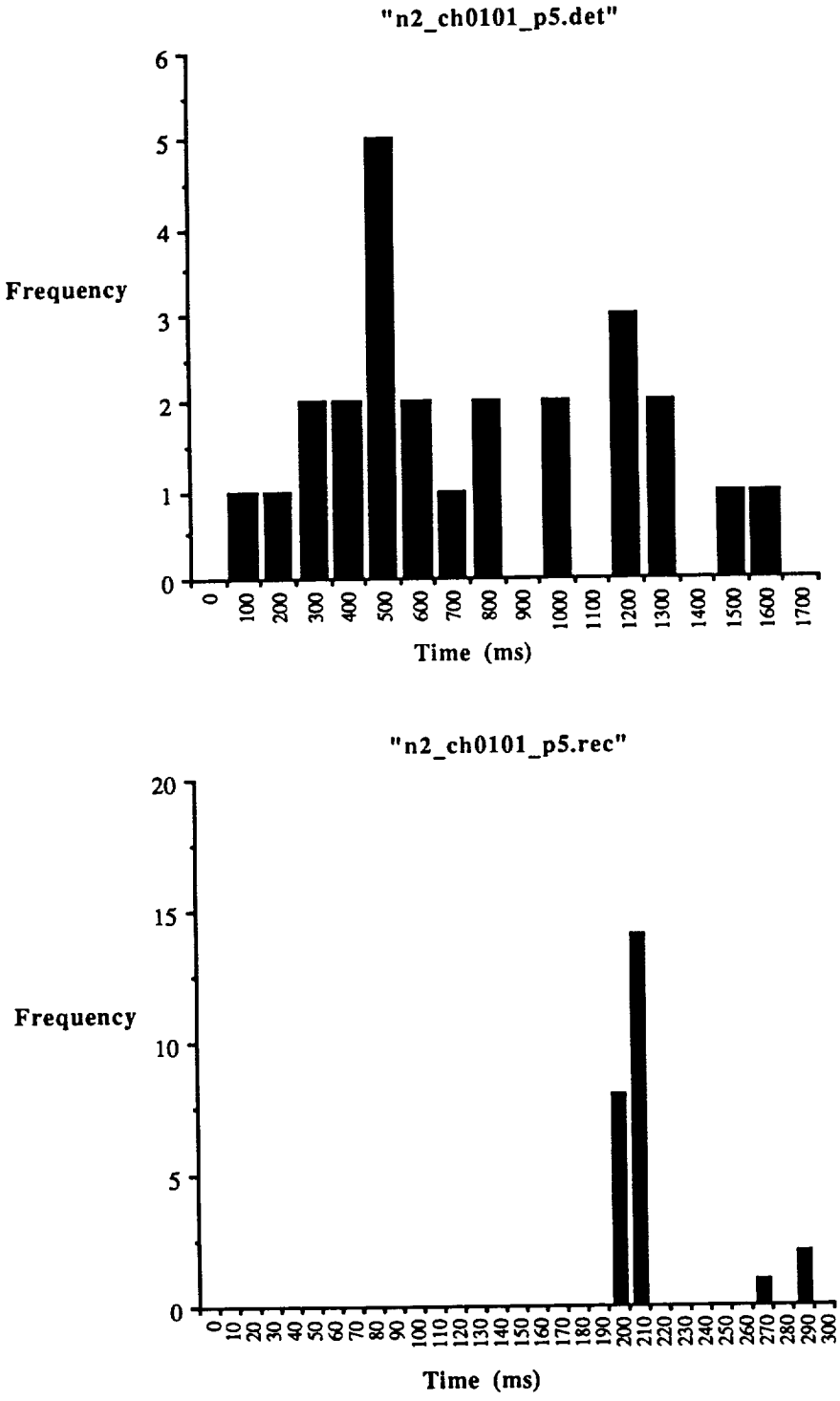

Figure 3-15. Test B.2.b

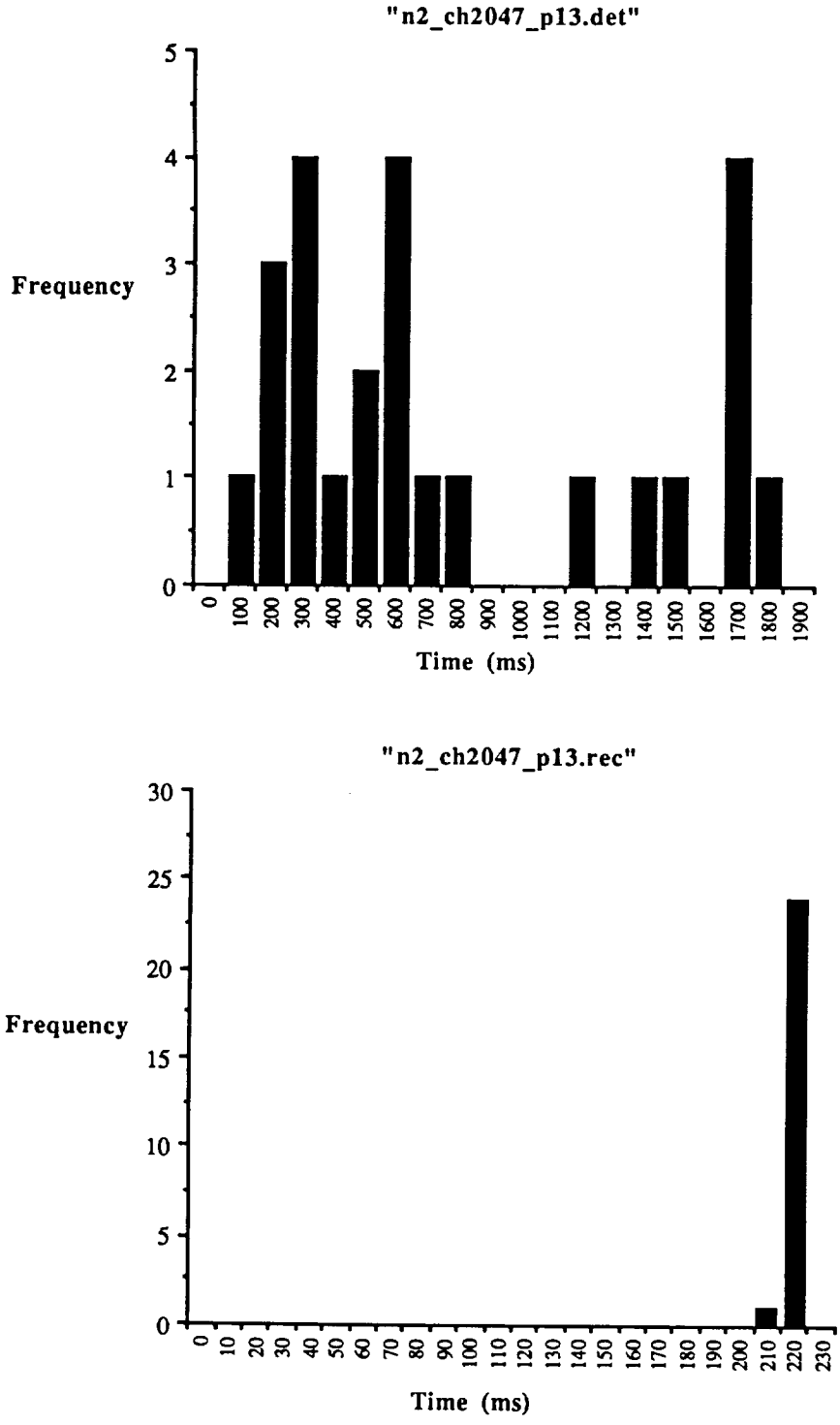

Figure 3-16. Test B.2.c

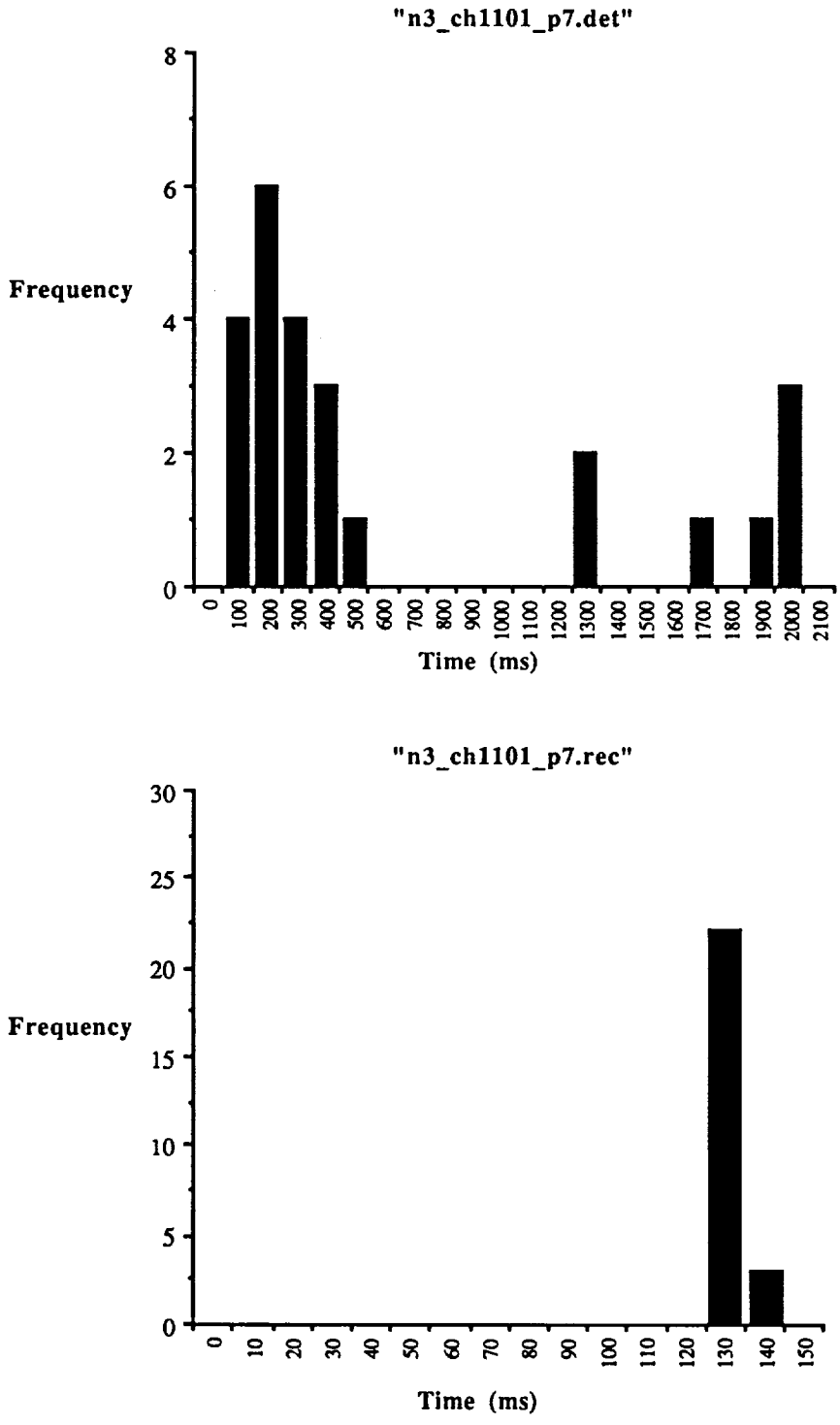

Figure 3-17. Test B.3.a

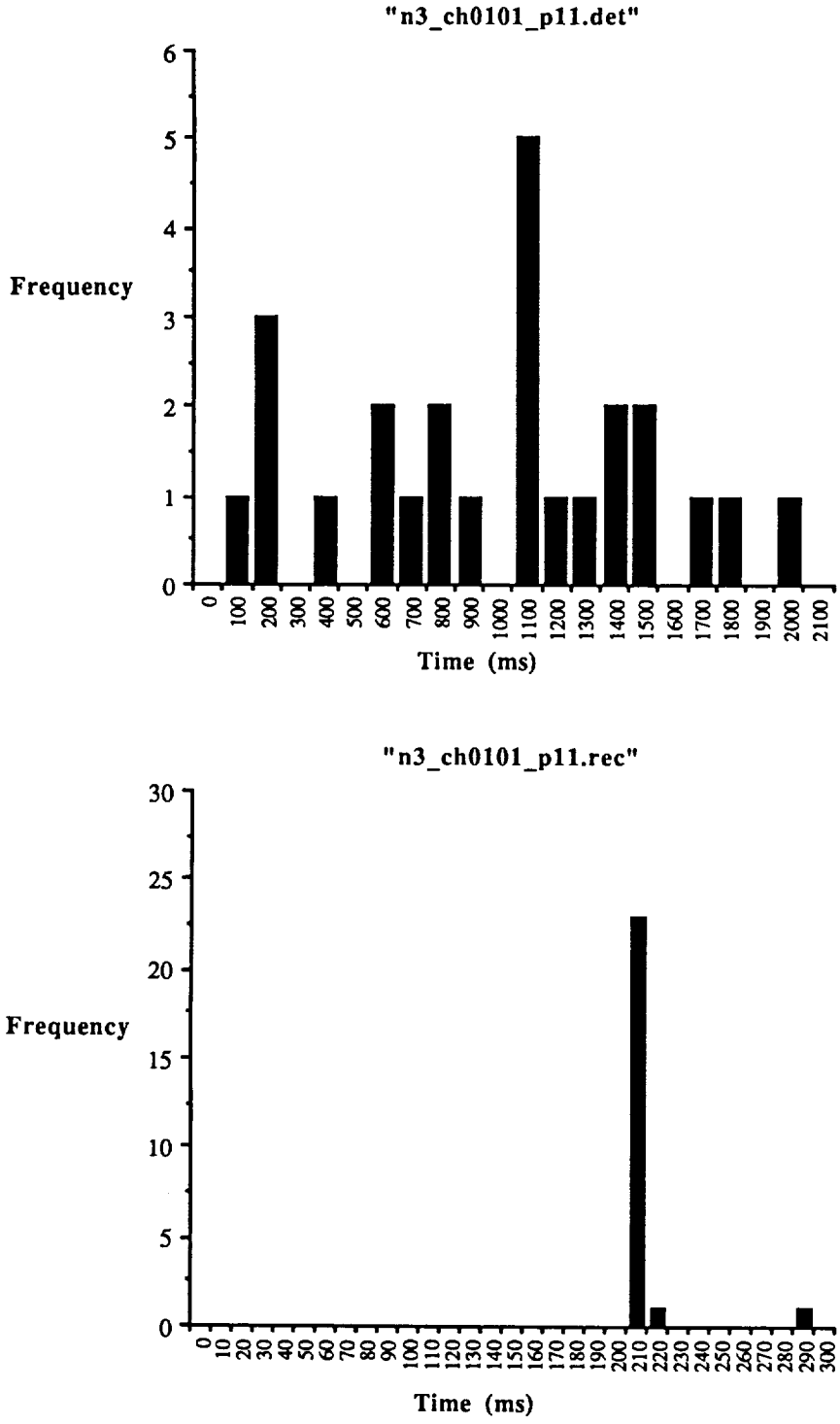

Figure 3-18. Test B.3.b

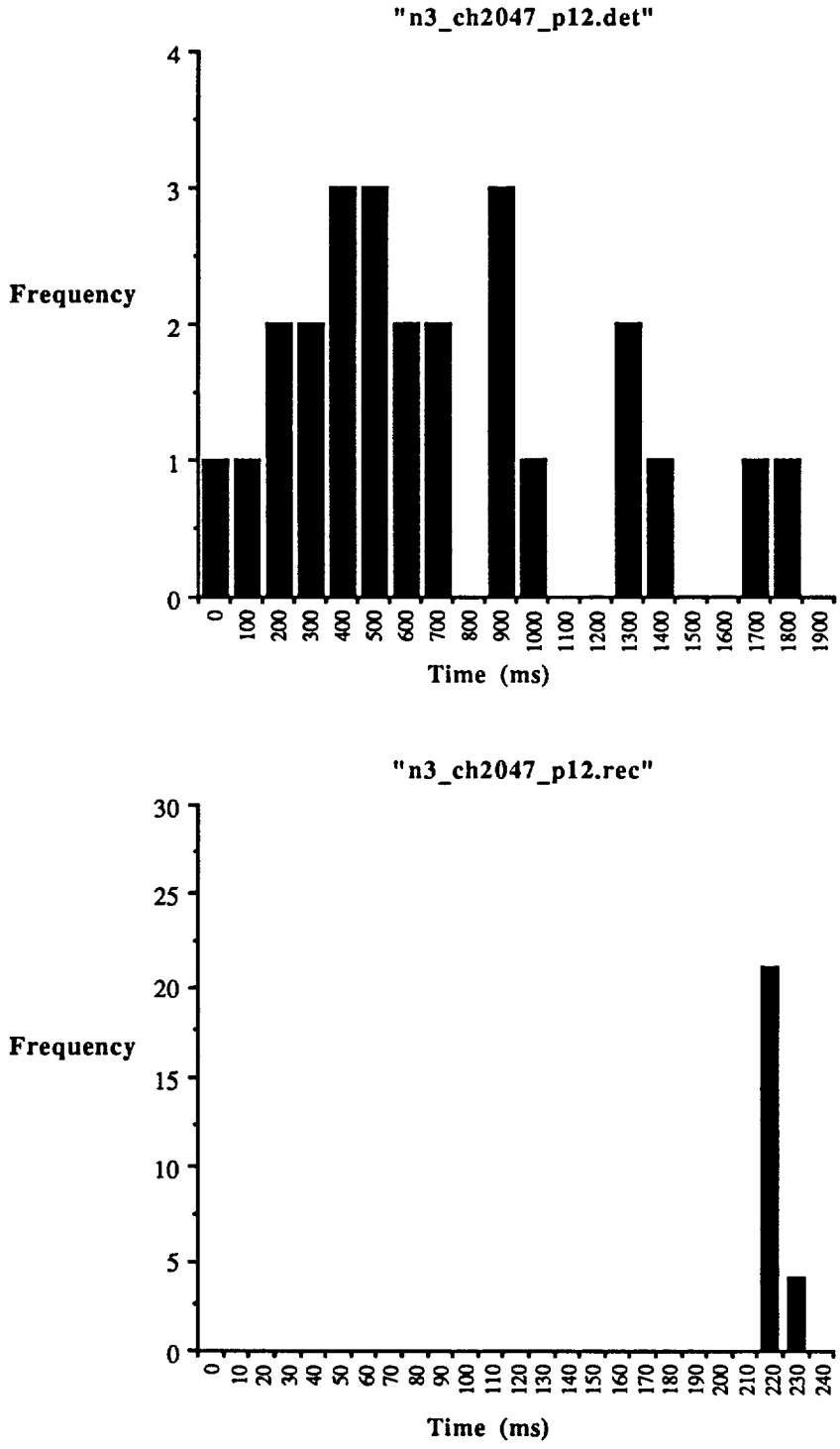

Figure 3-19. Test B.3.c

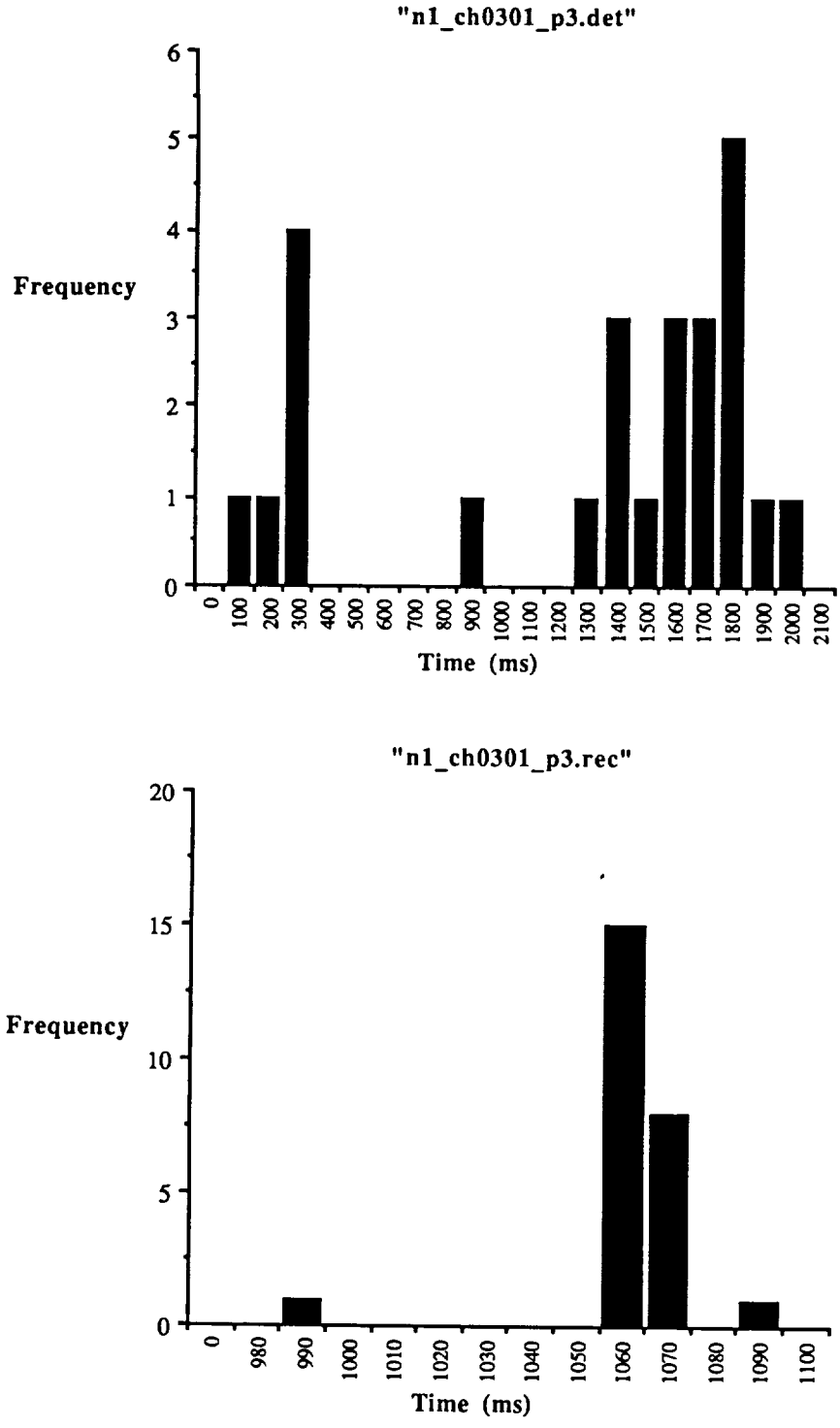

Figure 3-20. Test B.4.a

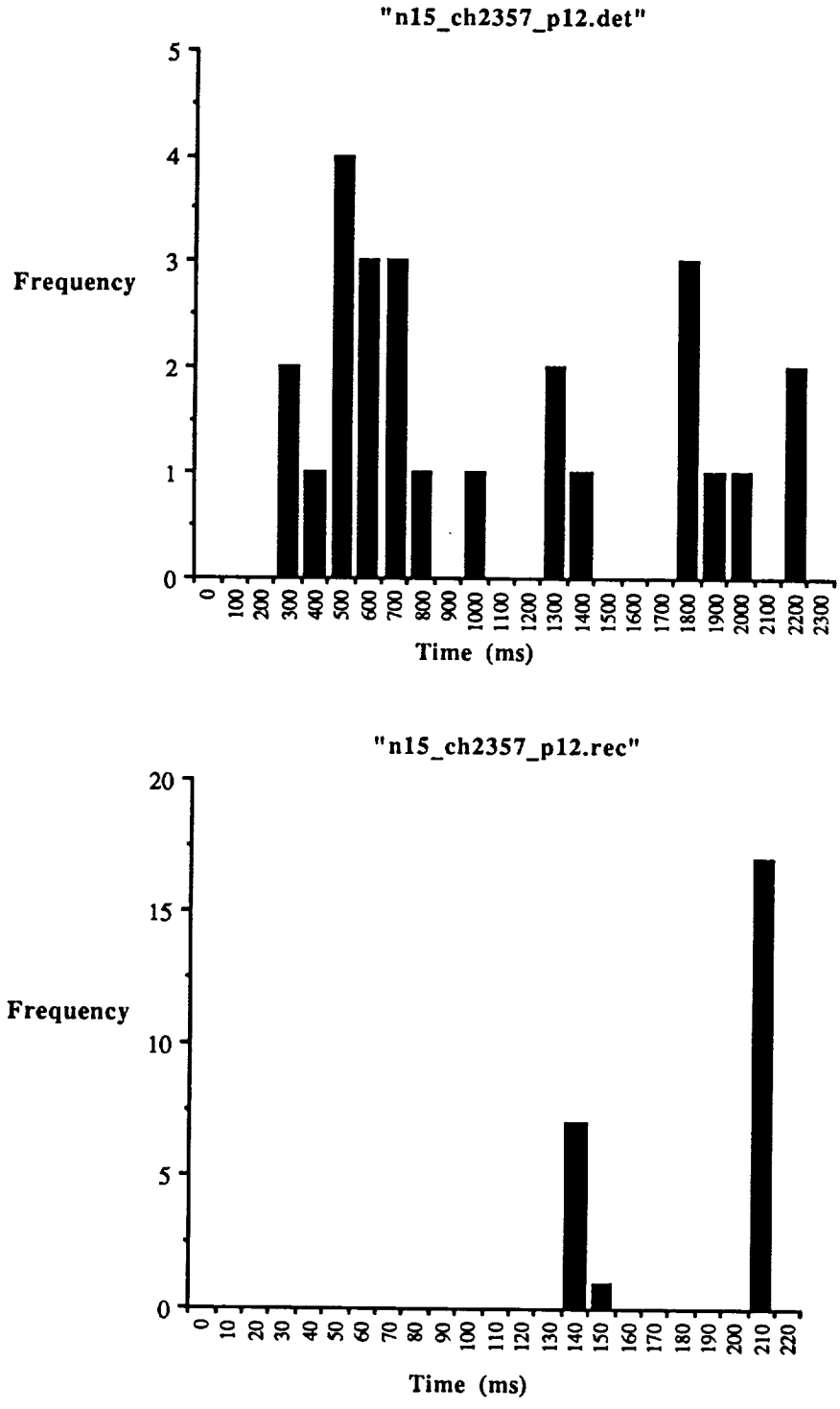

Figure 3-21. Test C.1.a

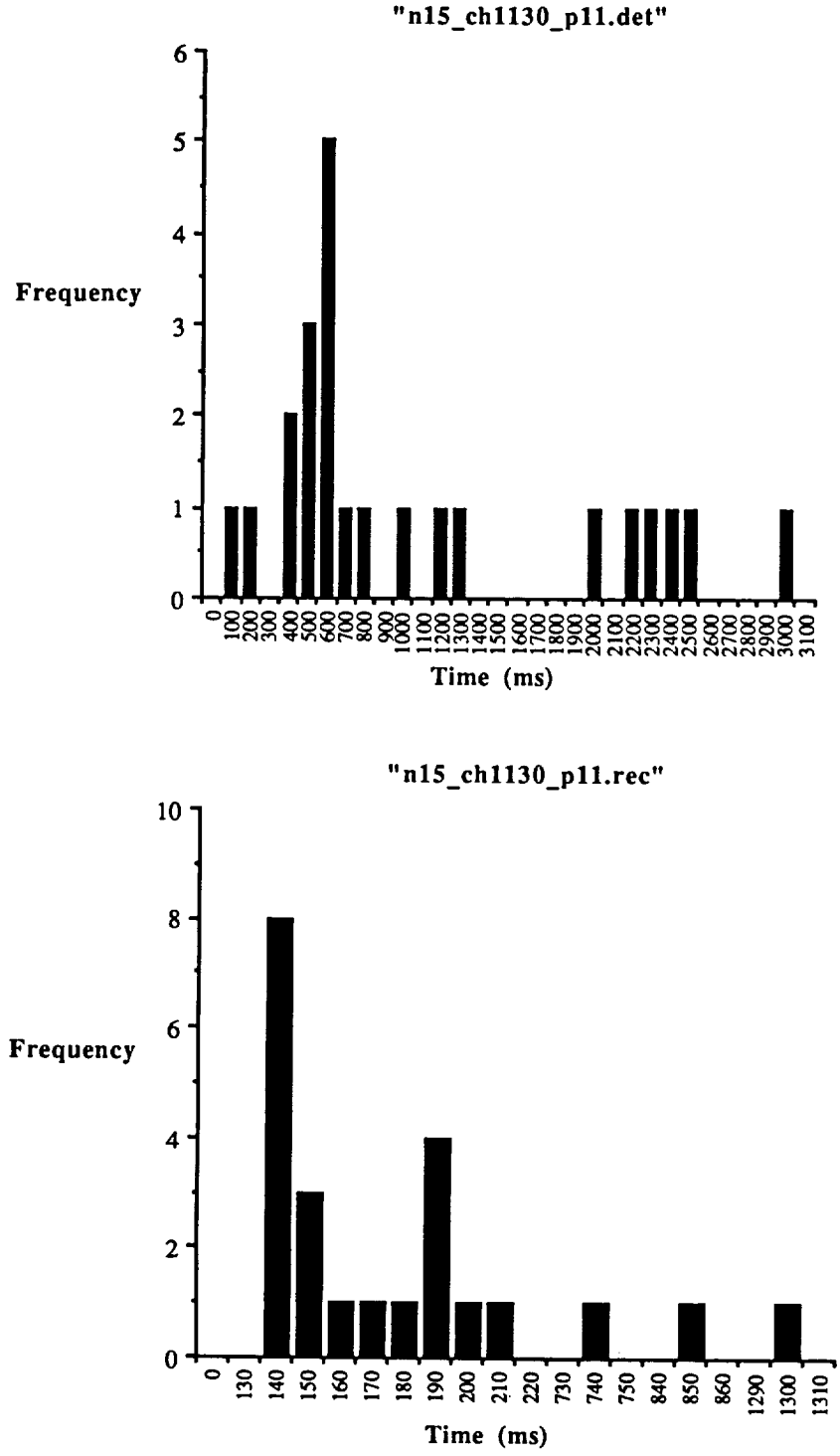

Figure 3-22. Test C.1.b

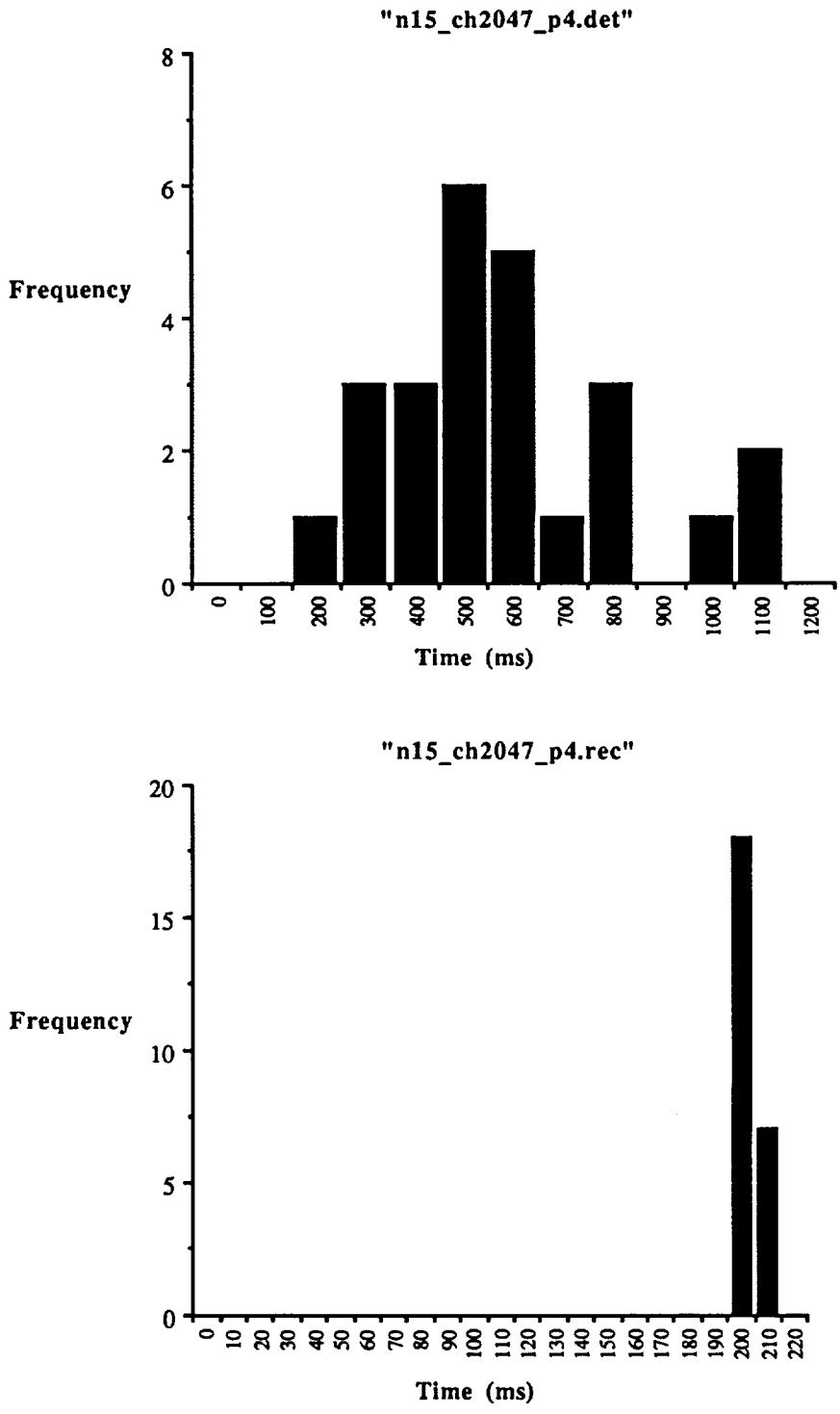

Figure 3-23. Test C.1.c

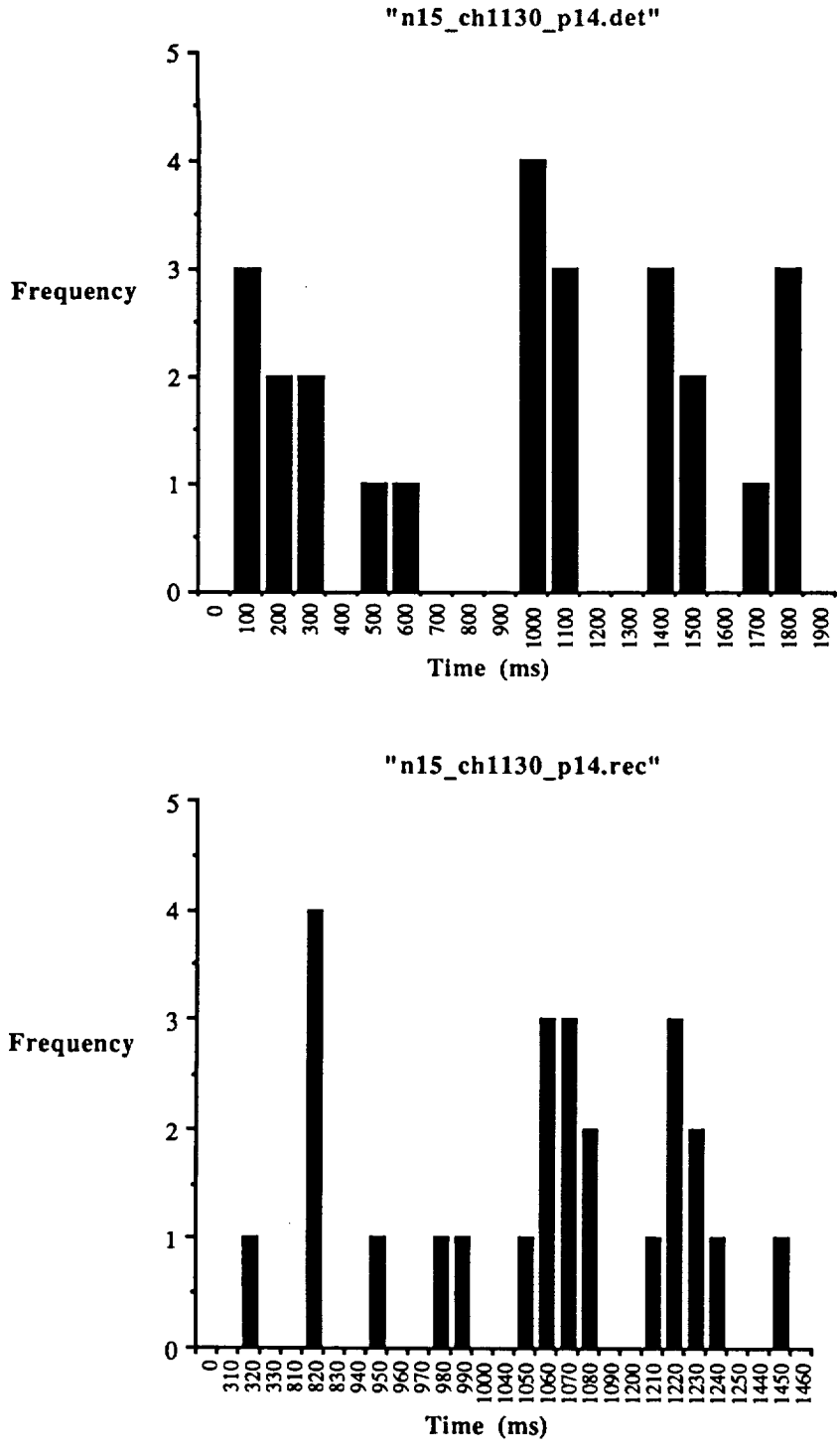

Figure 3-24. Test C.1.d

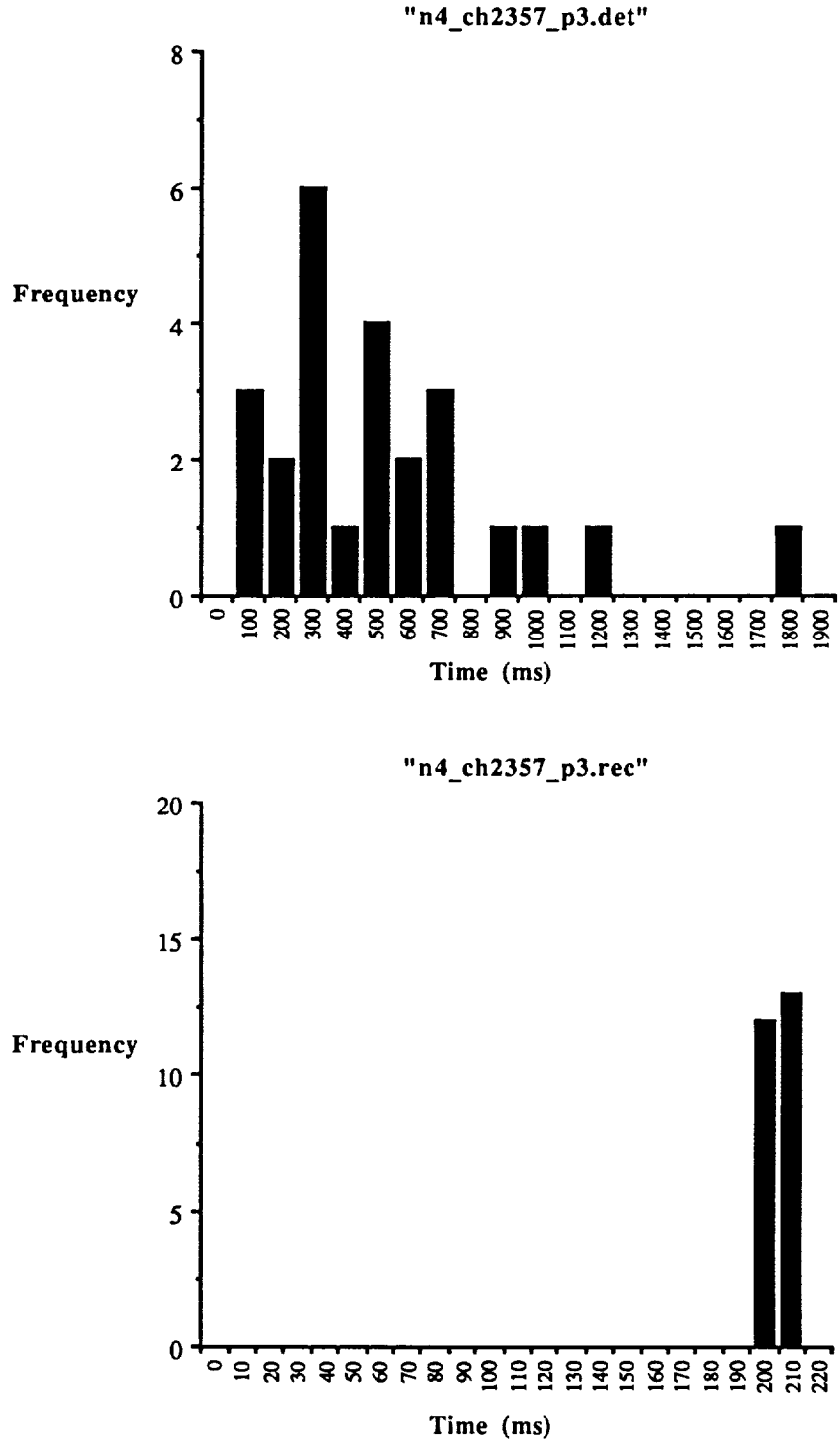

Figure 3-25. Test C.2.a

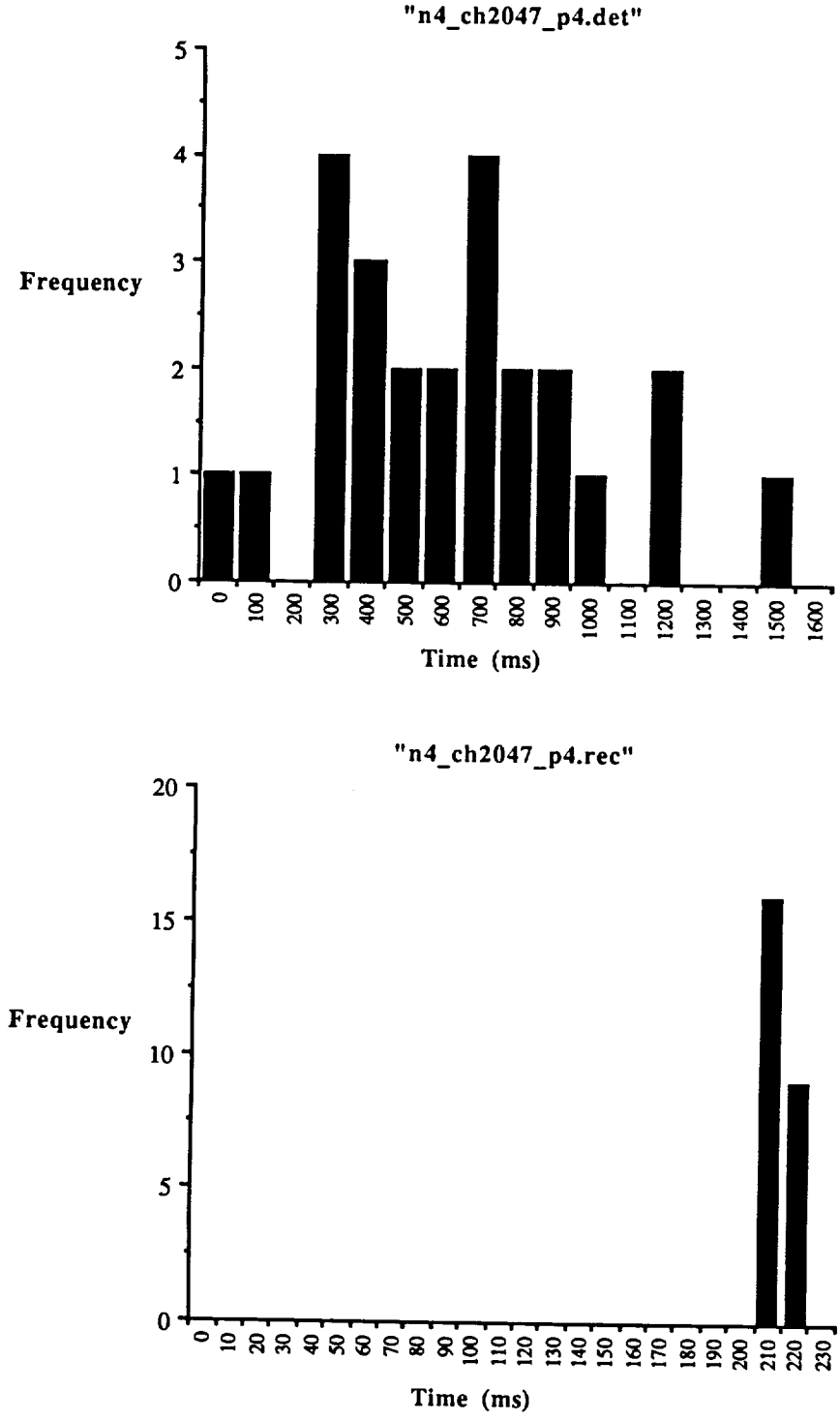

Figure 3-26. Test C.2.b

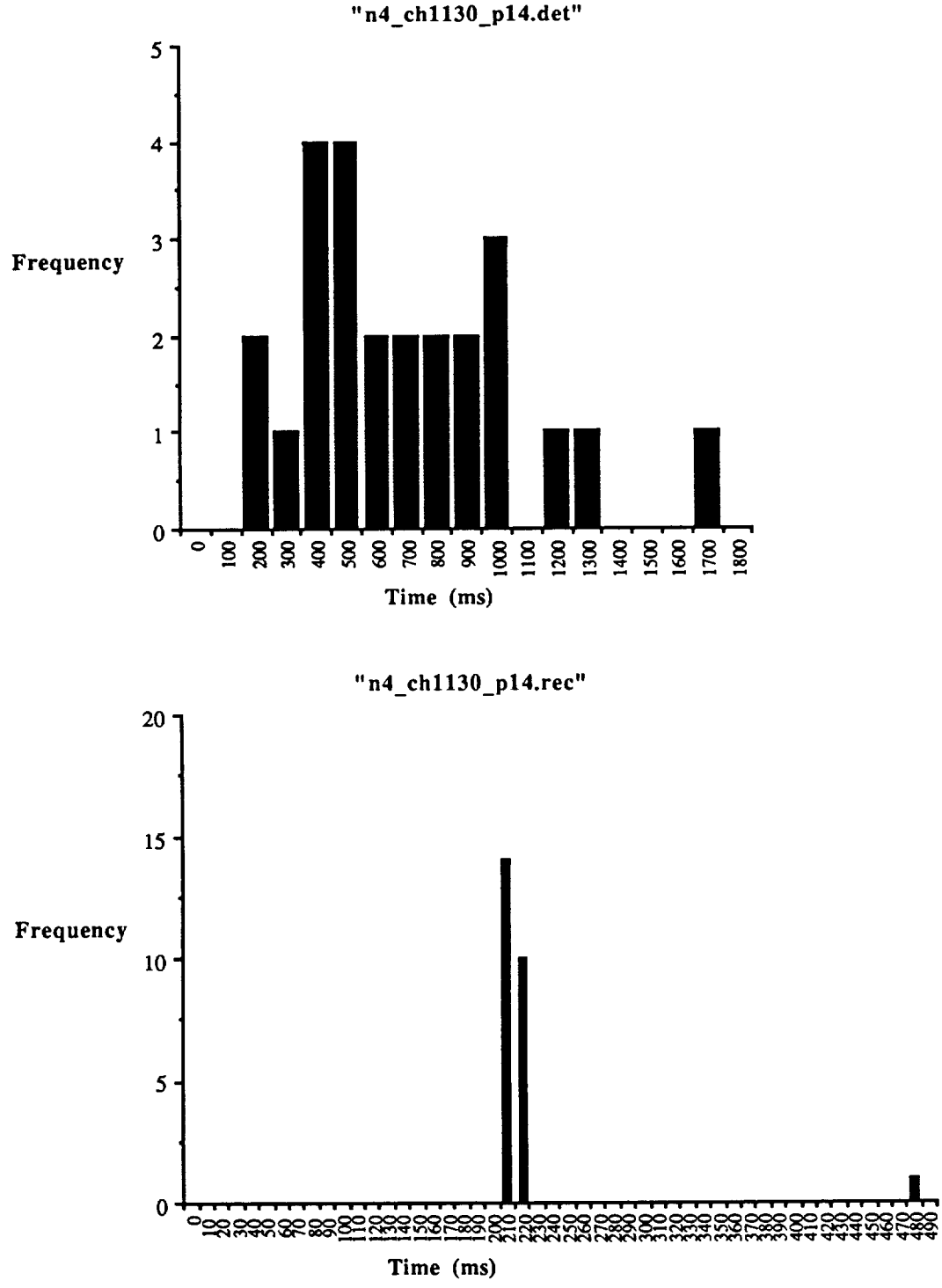

Figure 3-27. Test C.2.c

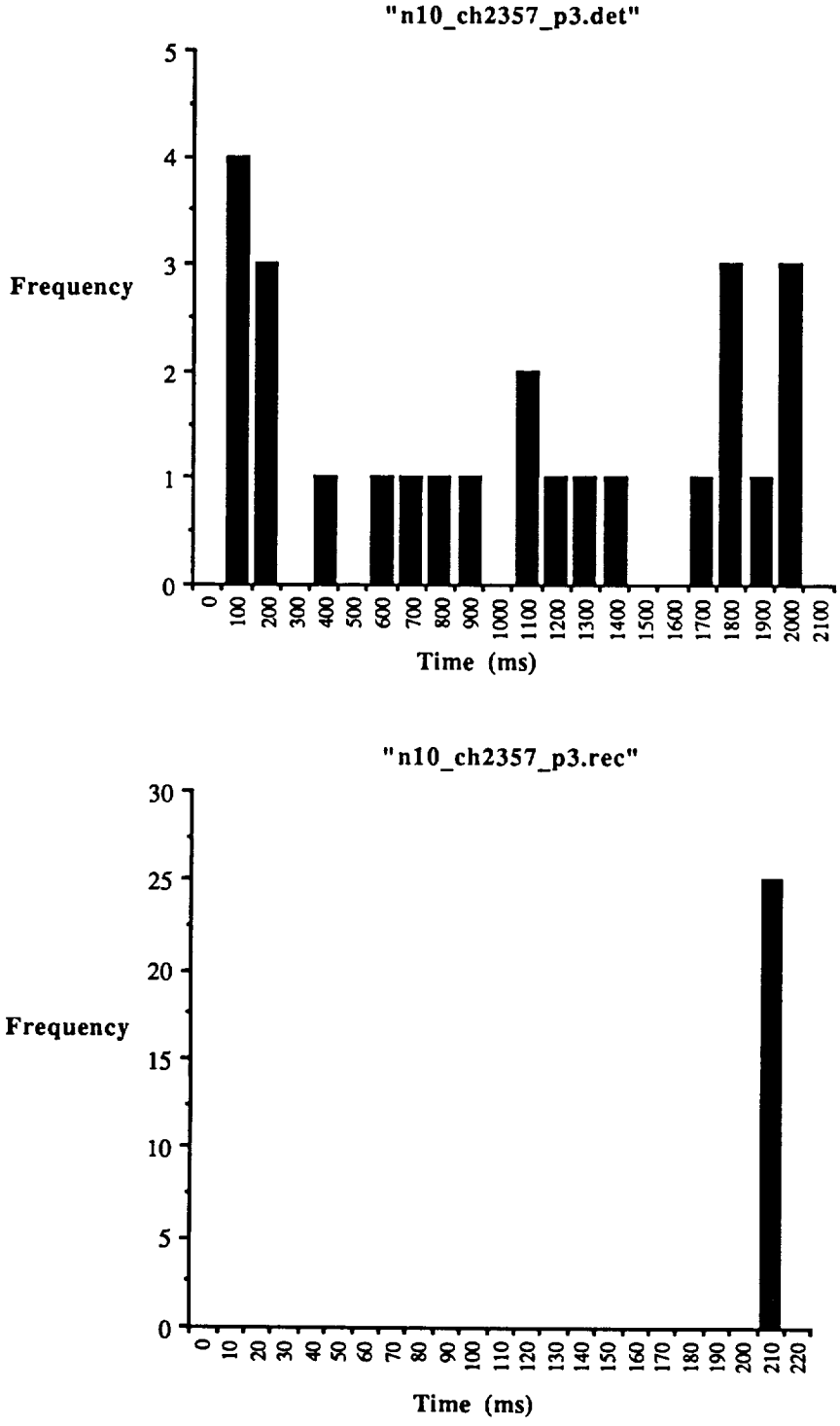

Figure 3-28. Test C.3.a

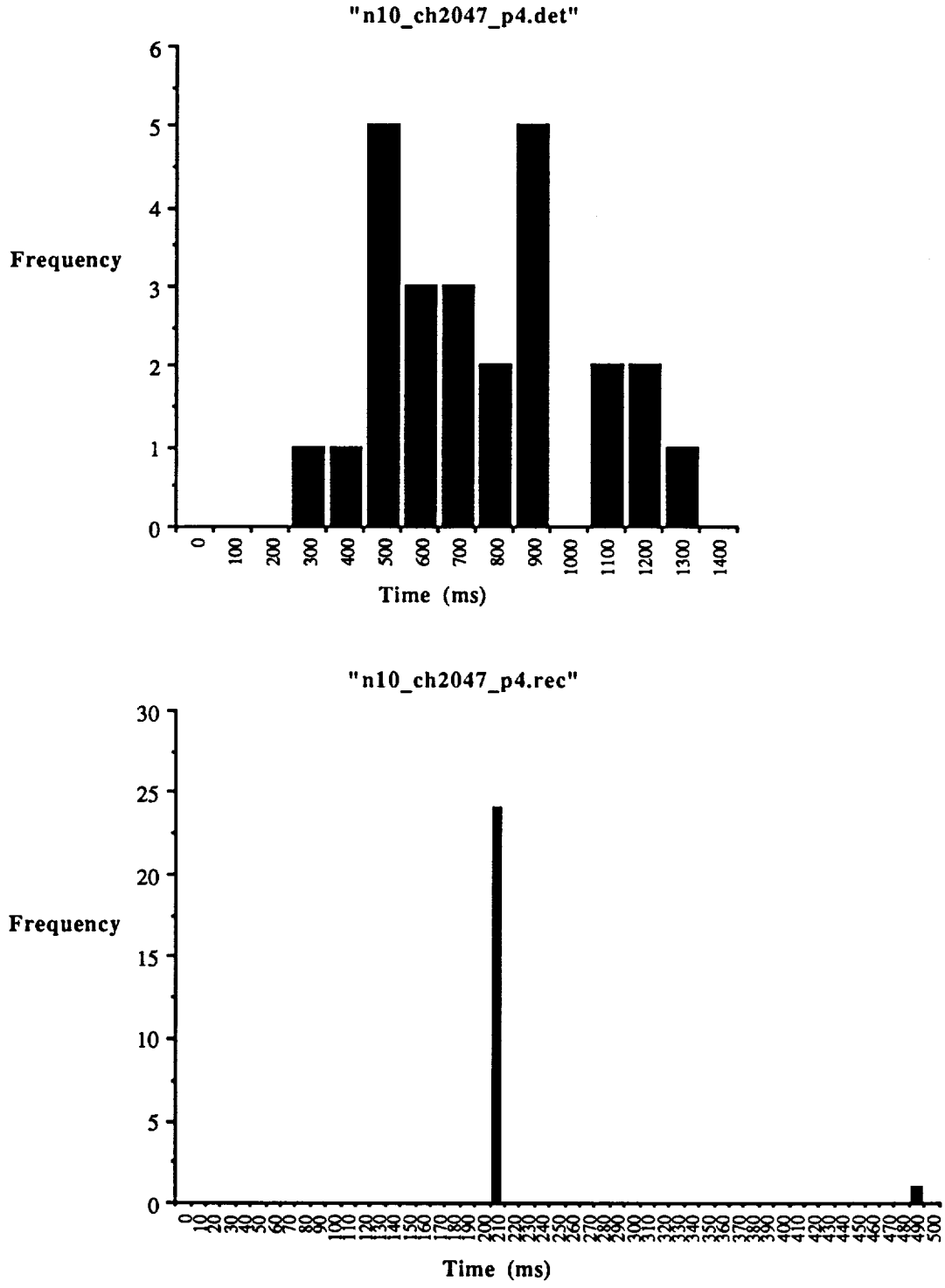

Figure 3-29. Test C.3.b

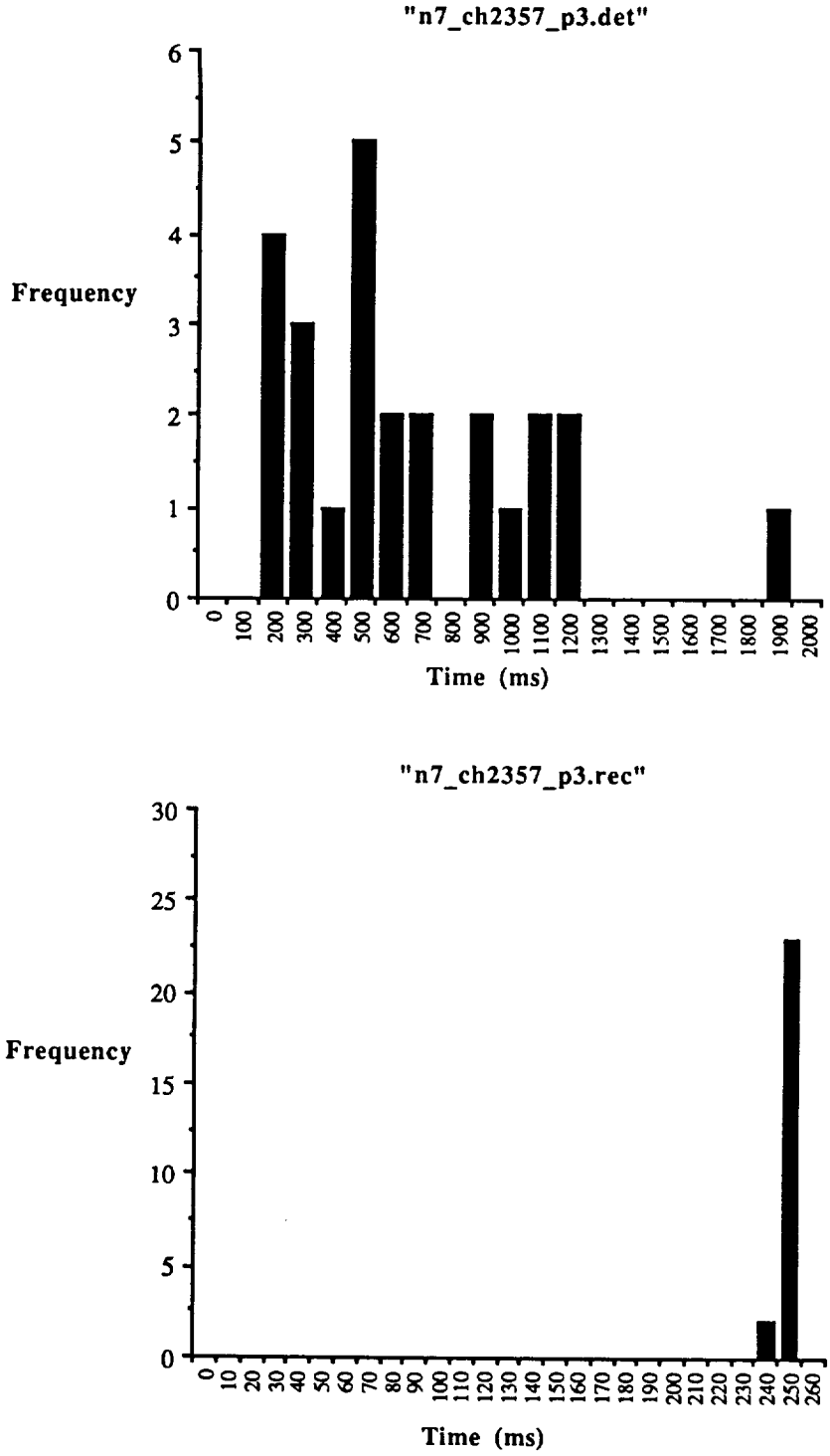

Figure 3-30. Test D.1.a

C-2

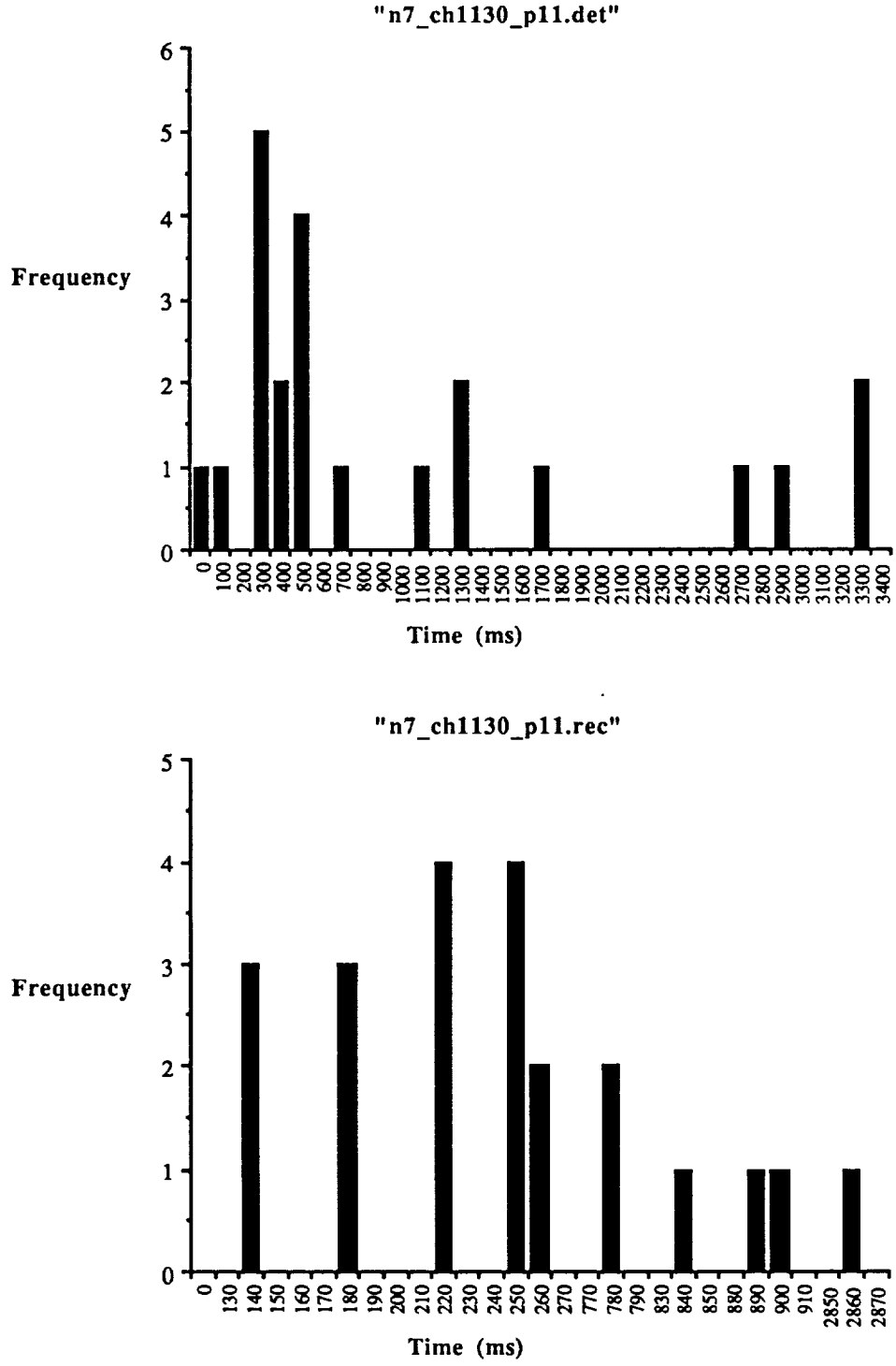

Figure 3-31. Test D.1.b

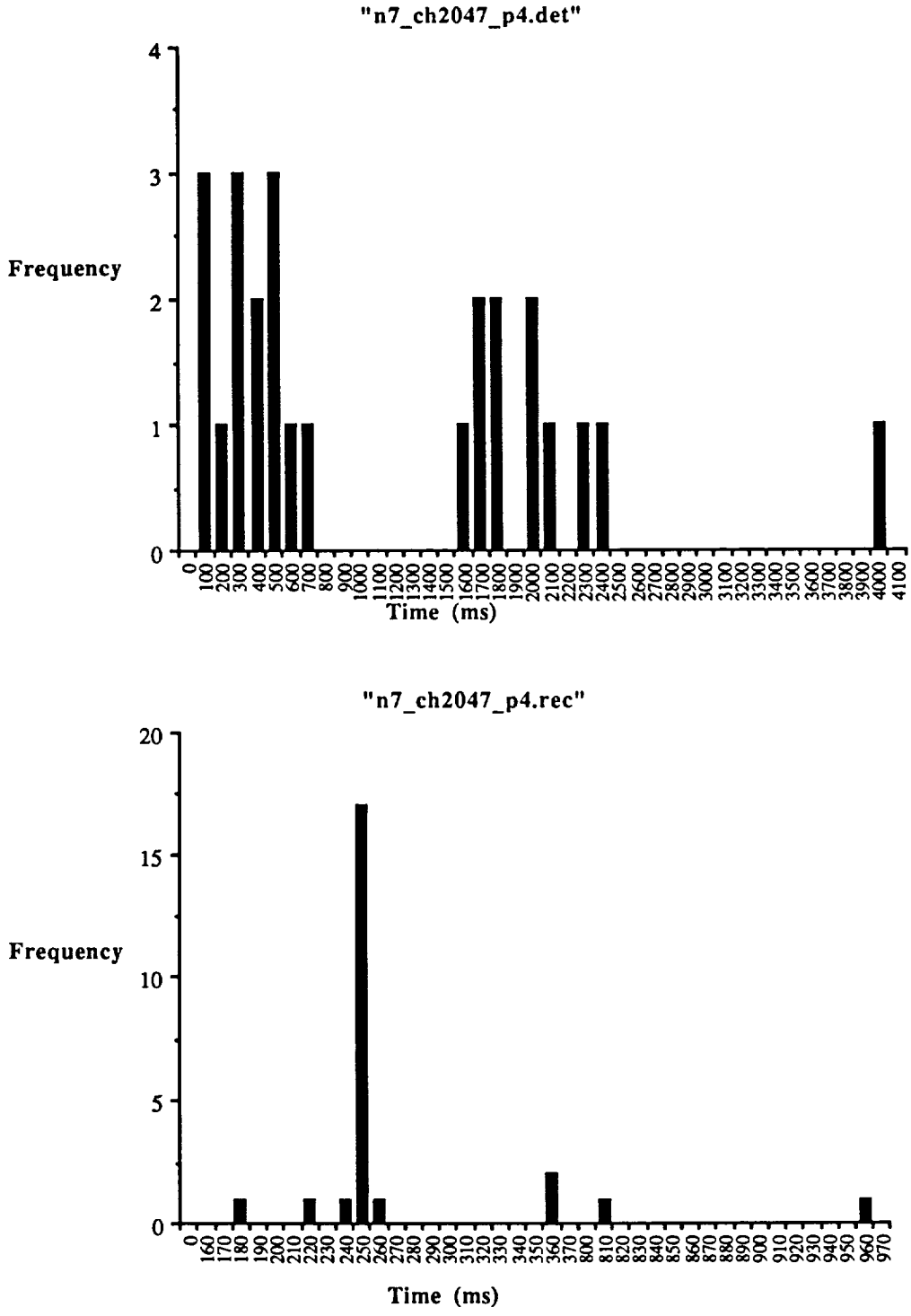

Figure 3-32. Test D.1.c

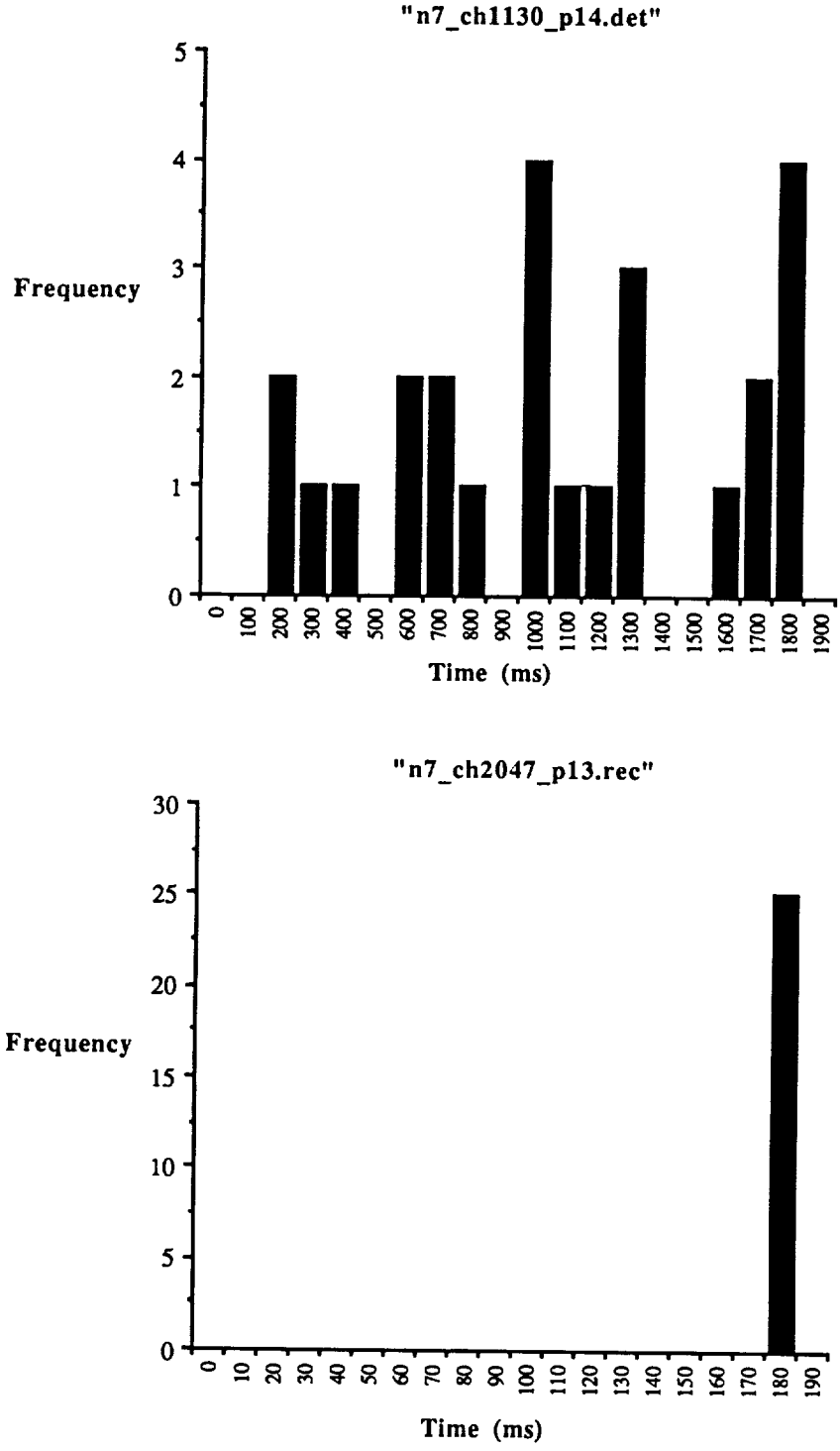

Figure 3-33. Test D.1.d

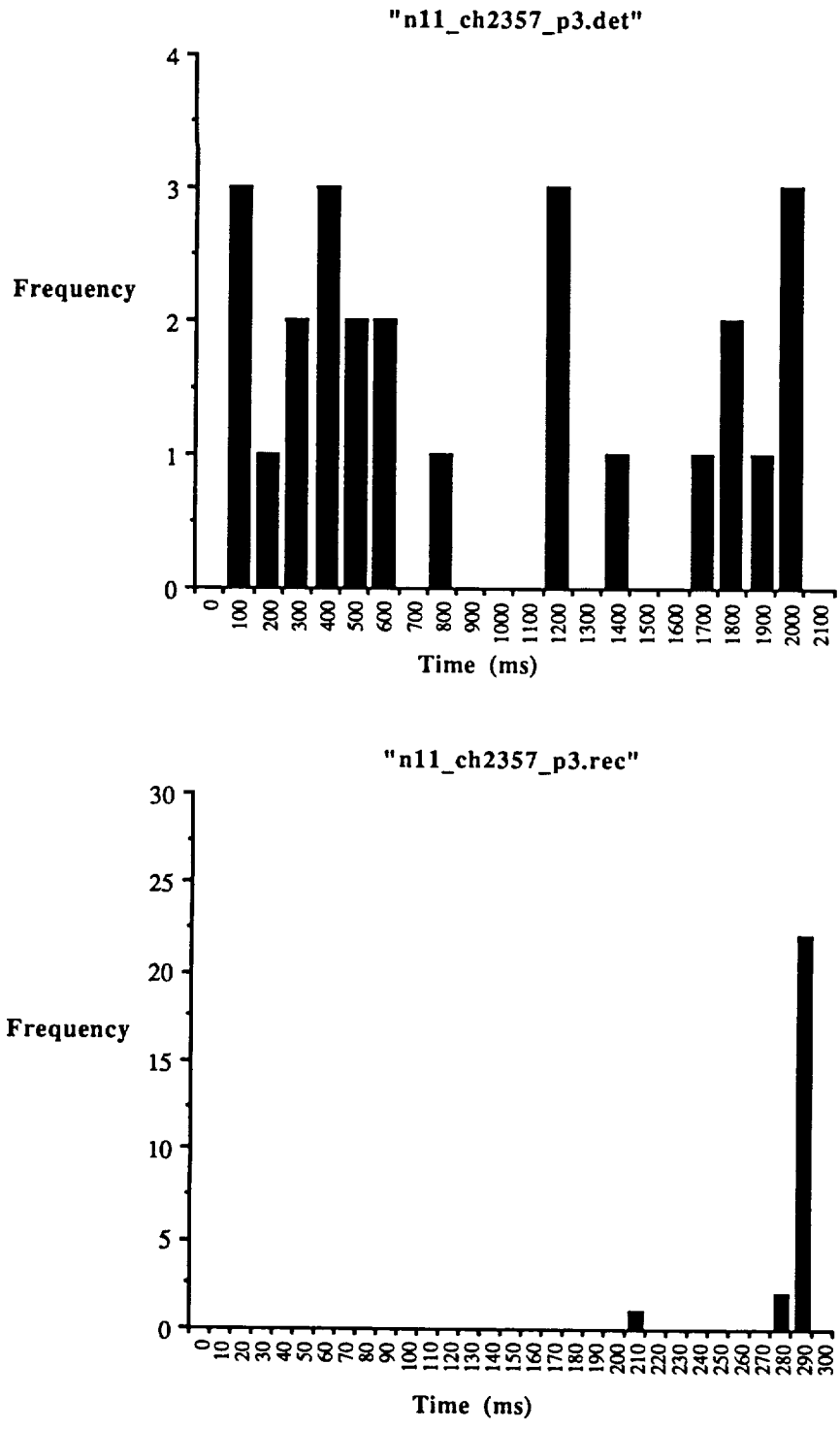

Figure 3-34. Test D.2.a

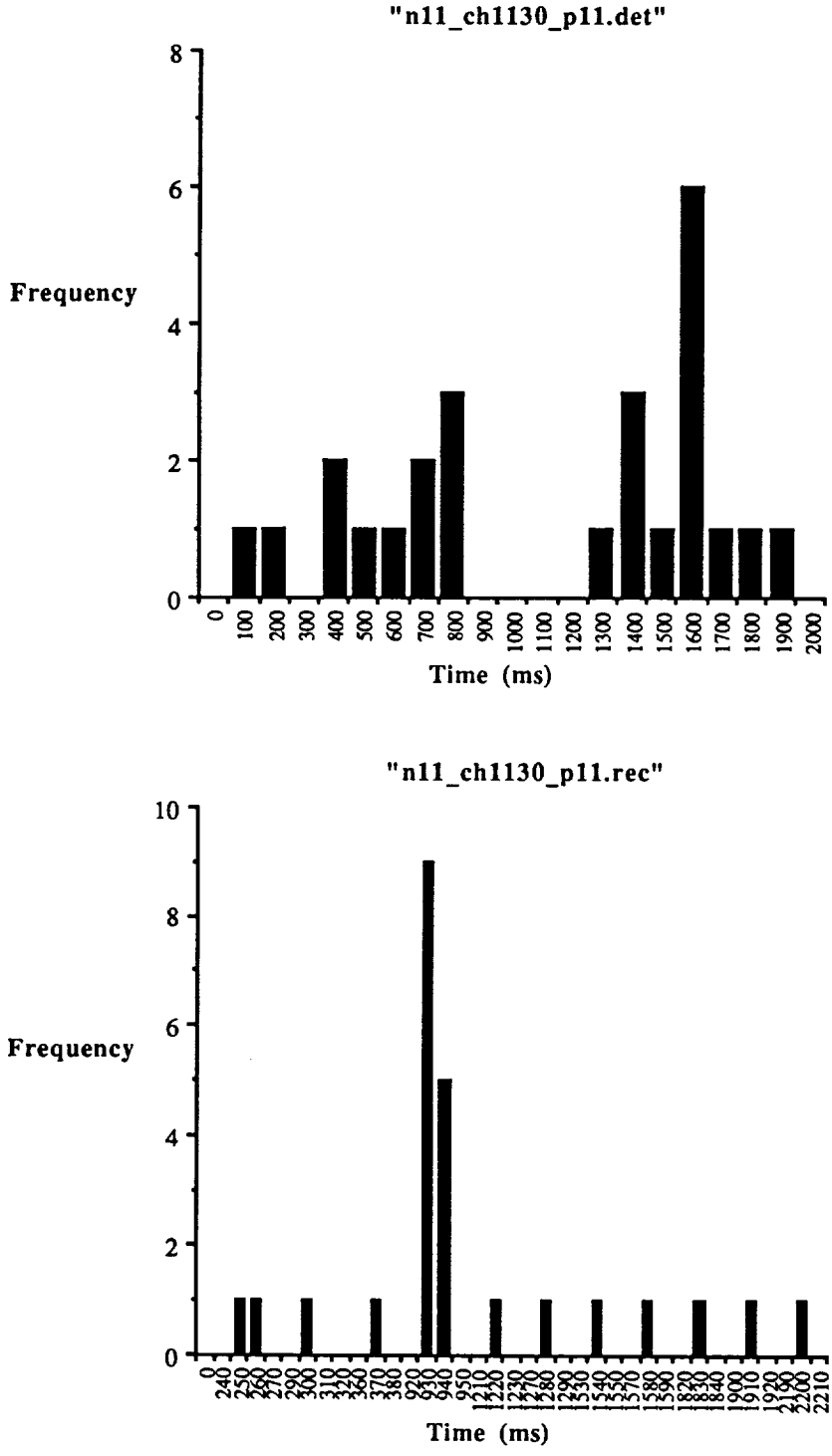

Figure 3-35. Test D.2.b

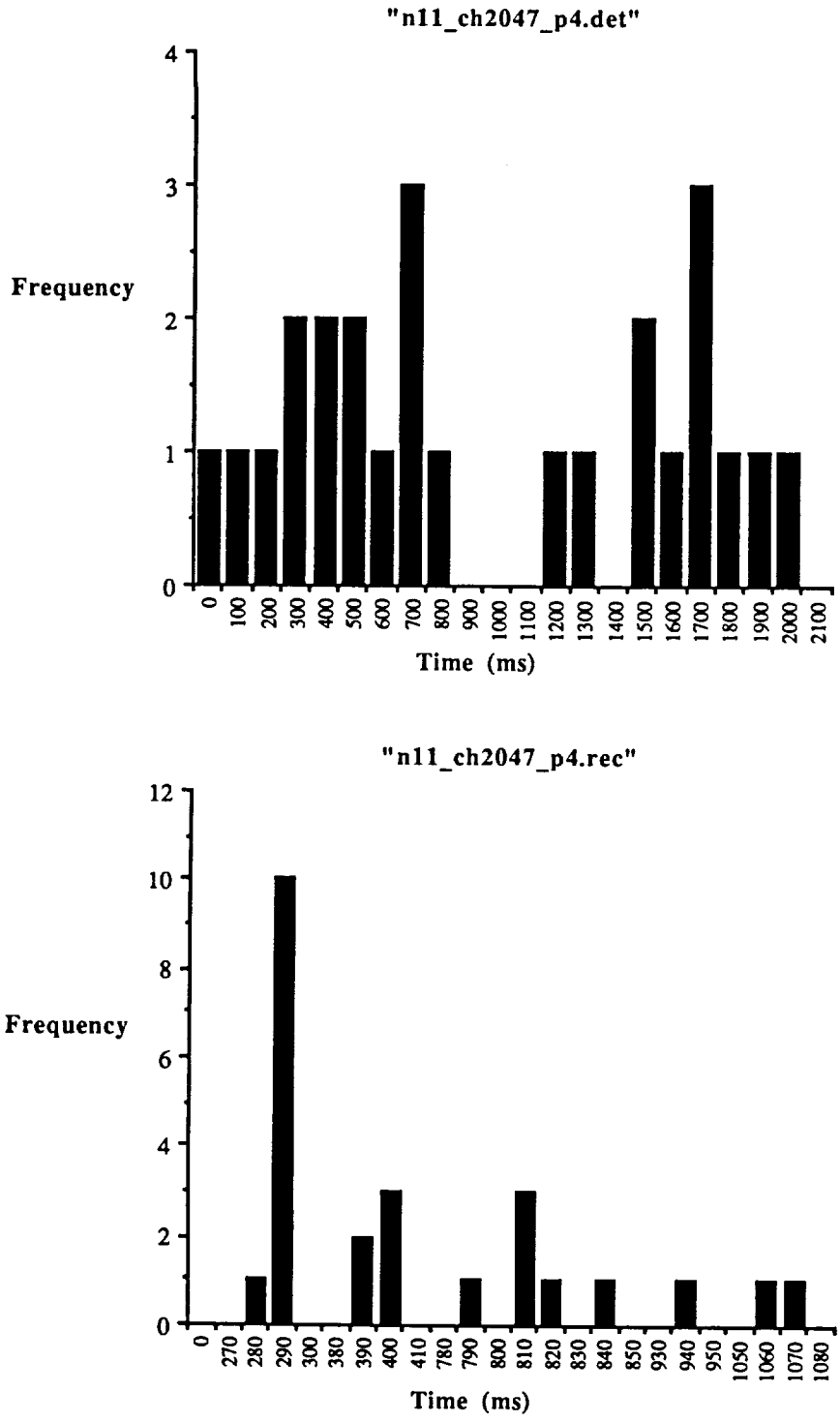

Figure 3-36. Test D.2.c

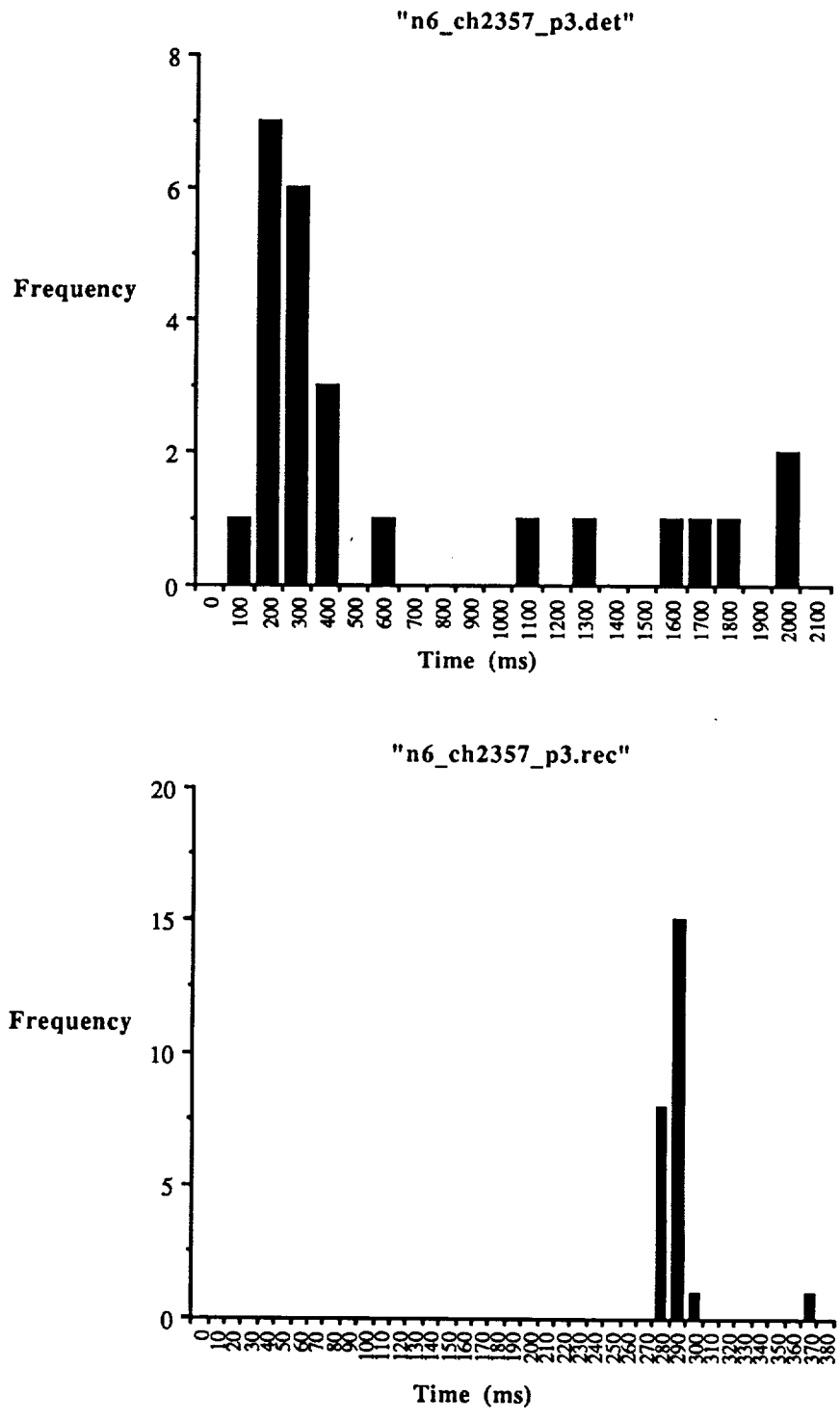

Figure 3-37. Test D.3.a

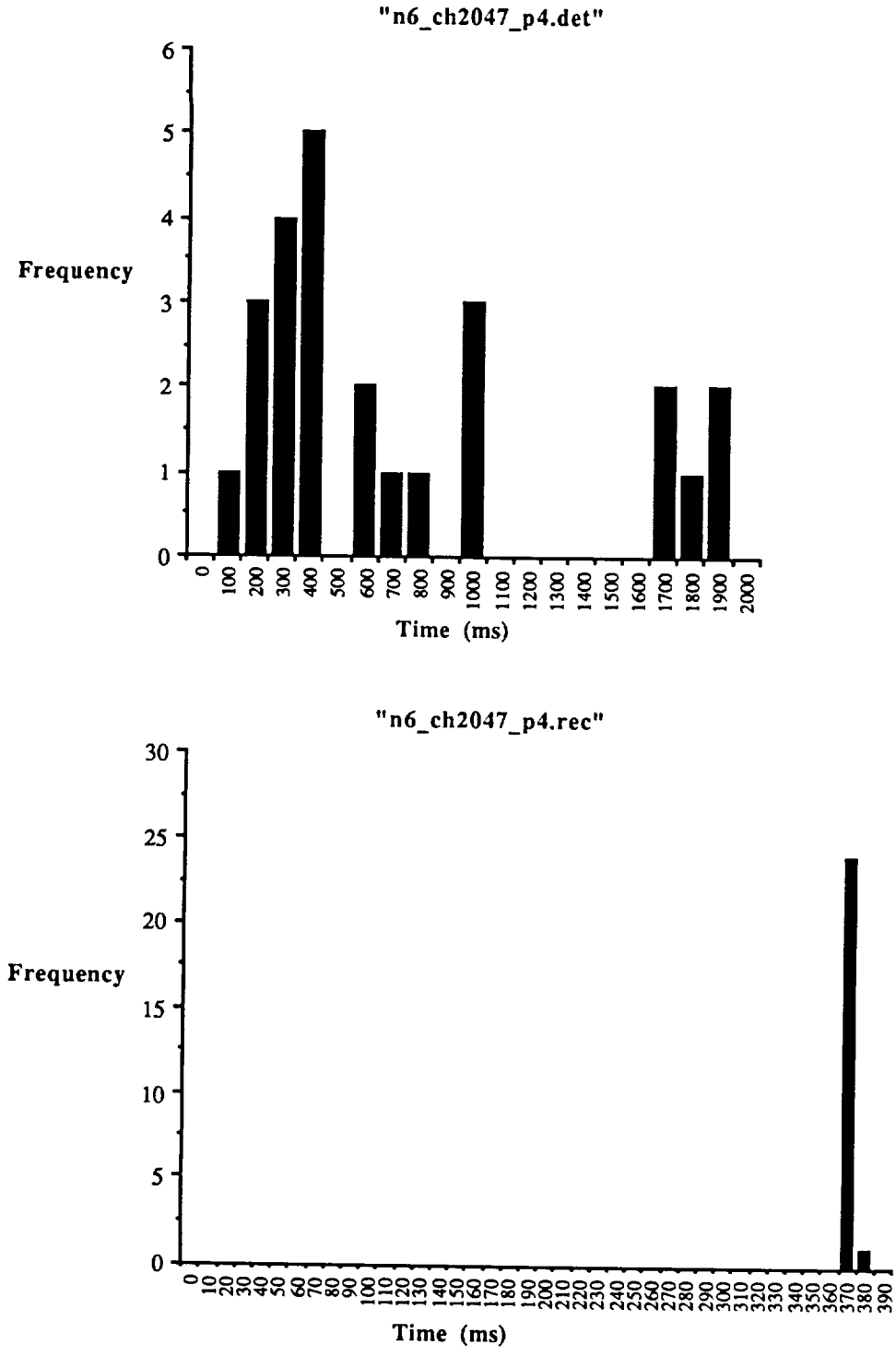

Figure 3-38. Test D.3.b

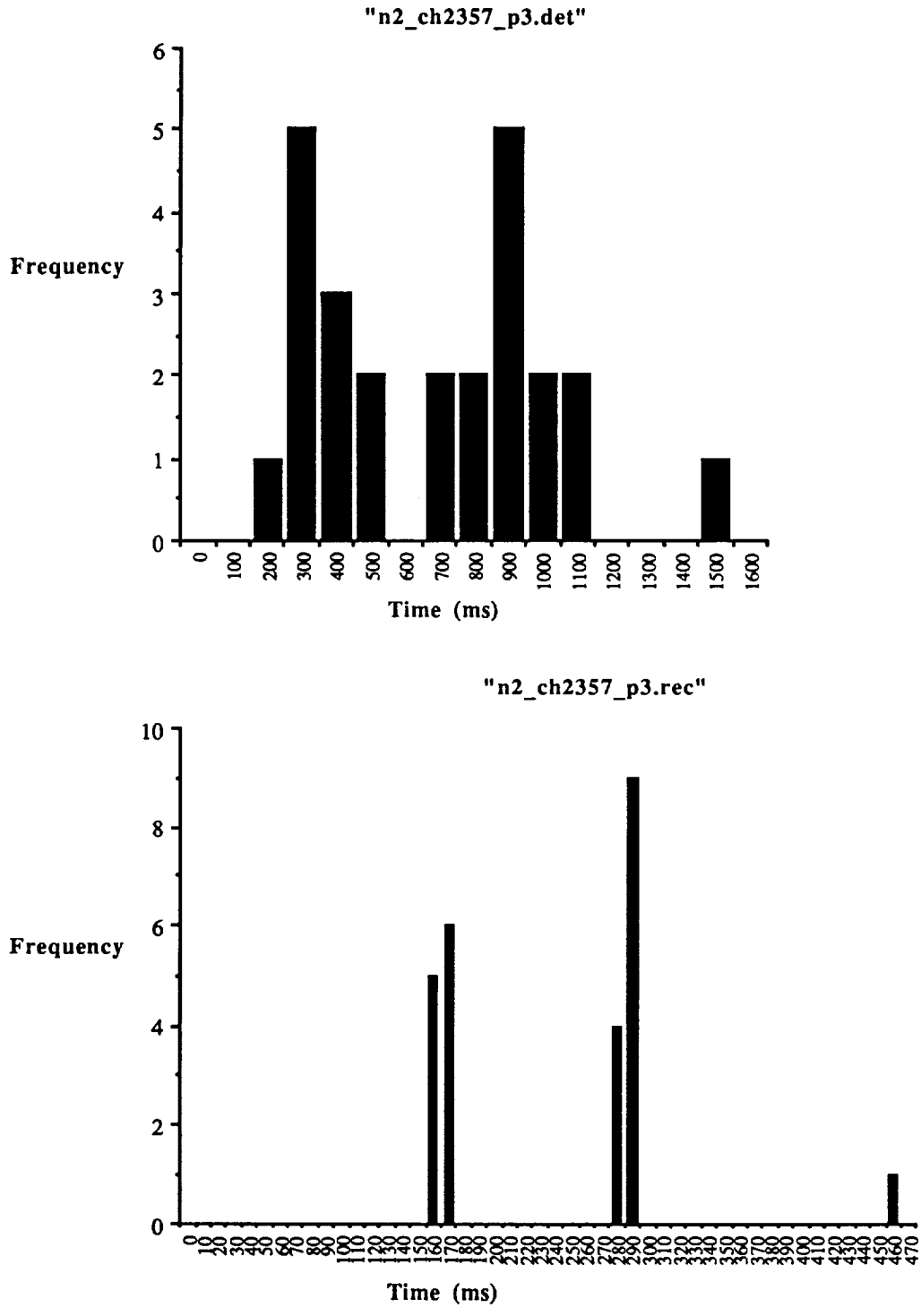

Figure 3-39. Test D.4.a

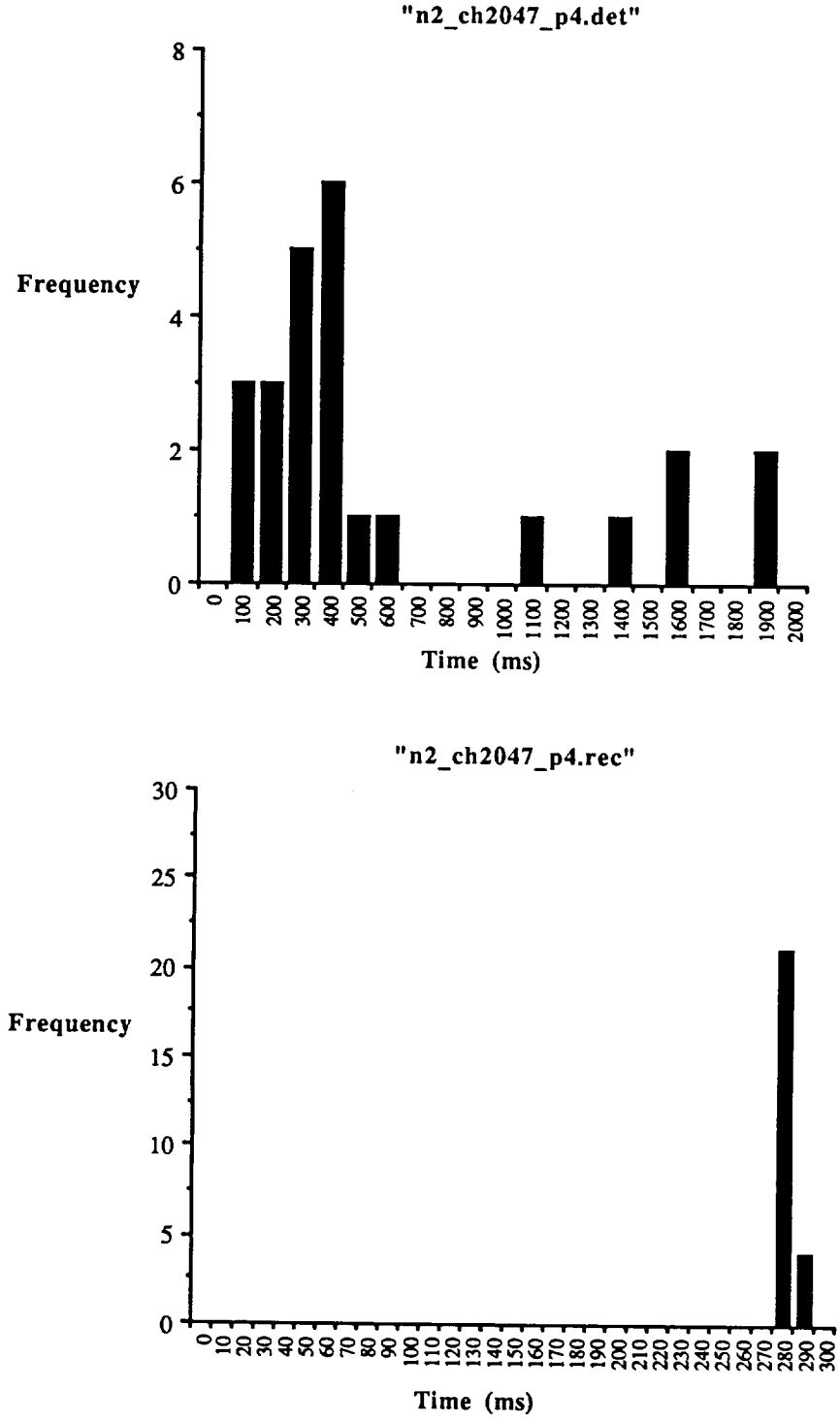

Figure 3-40. Test D.4.b

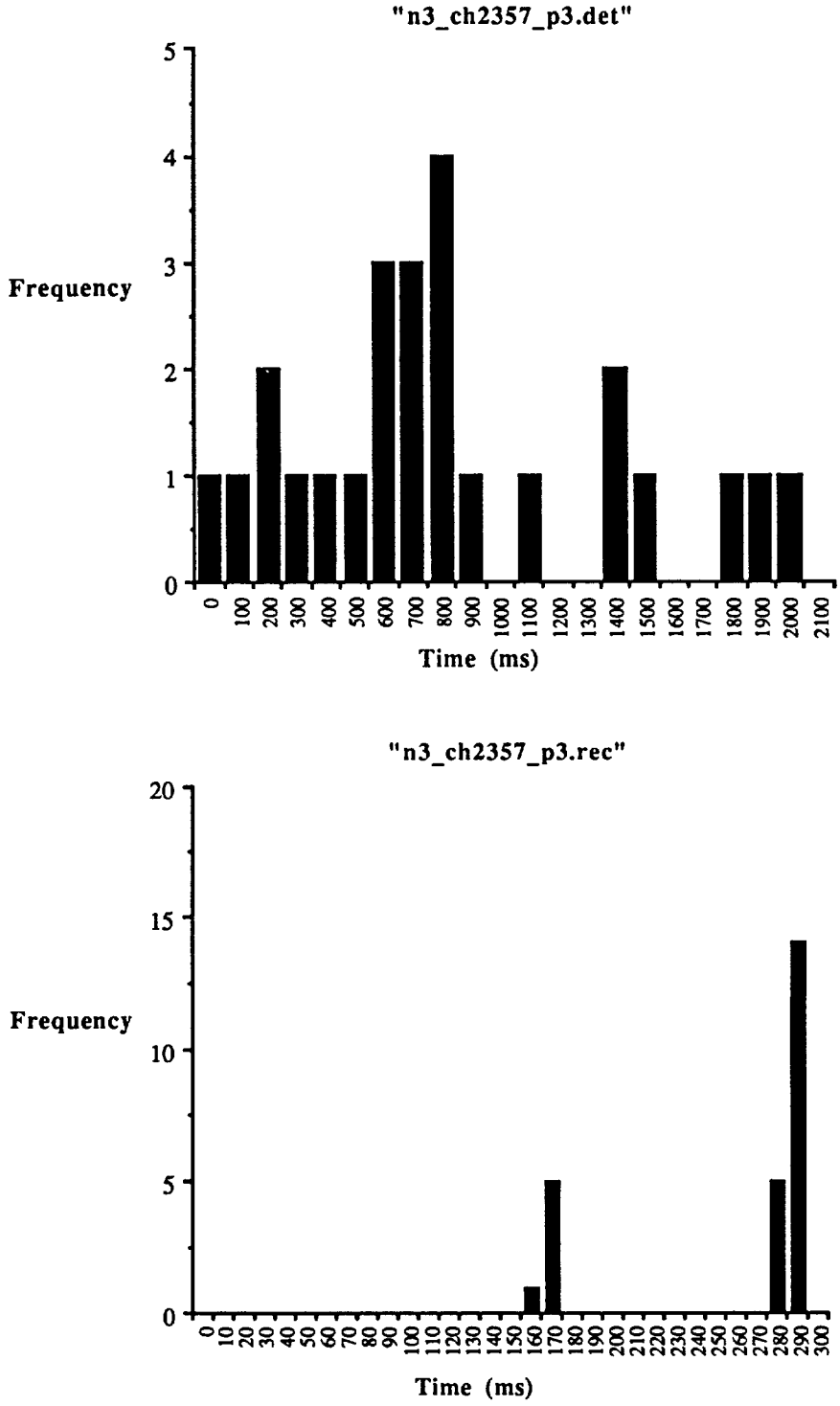

Figure 3-41. Test D.5.a

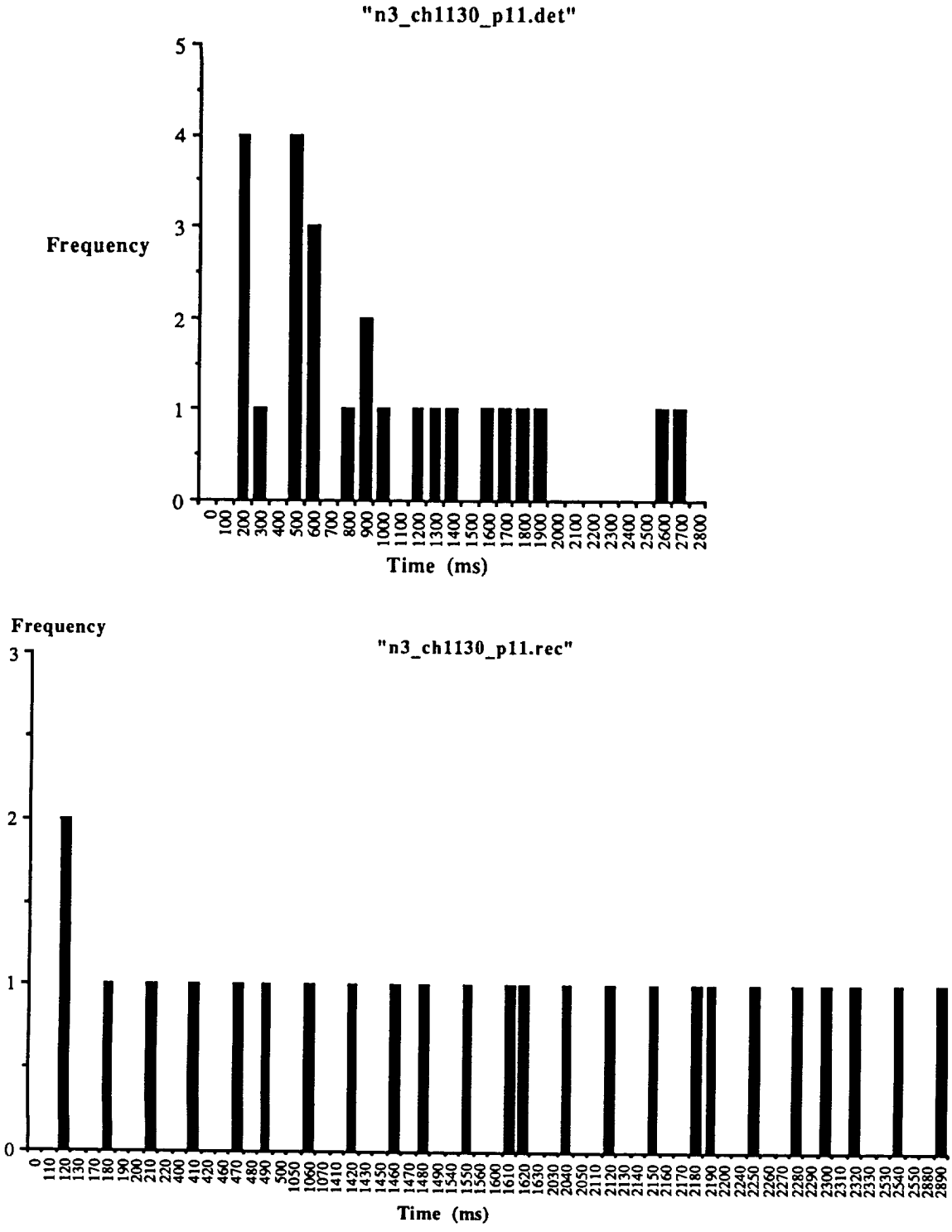

Figure 3-42. Test D.5.b

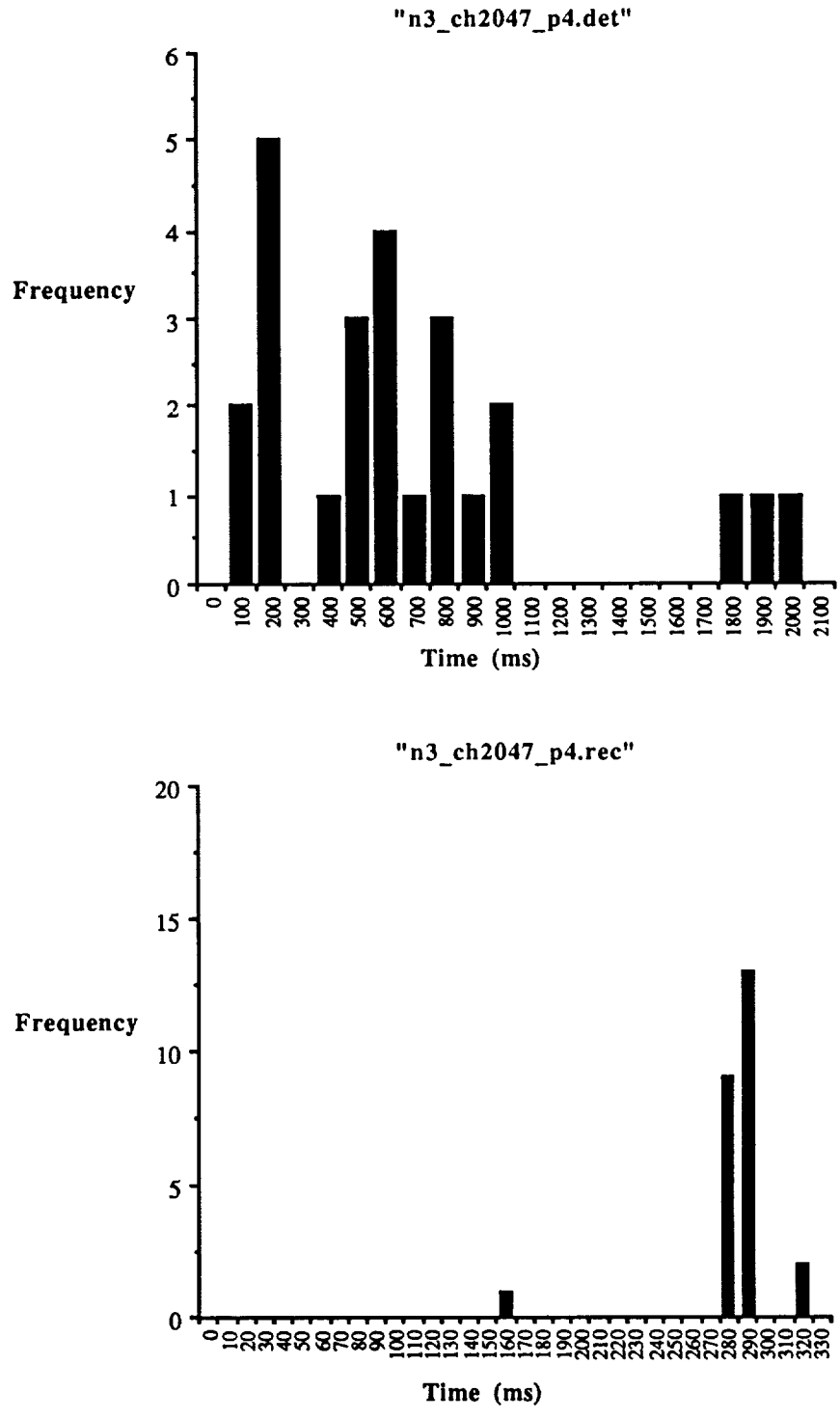

Figure 3-43. Test D.5.c

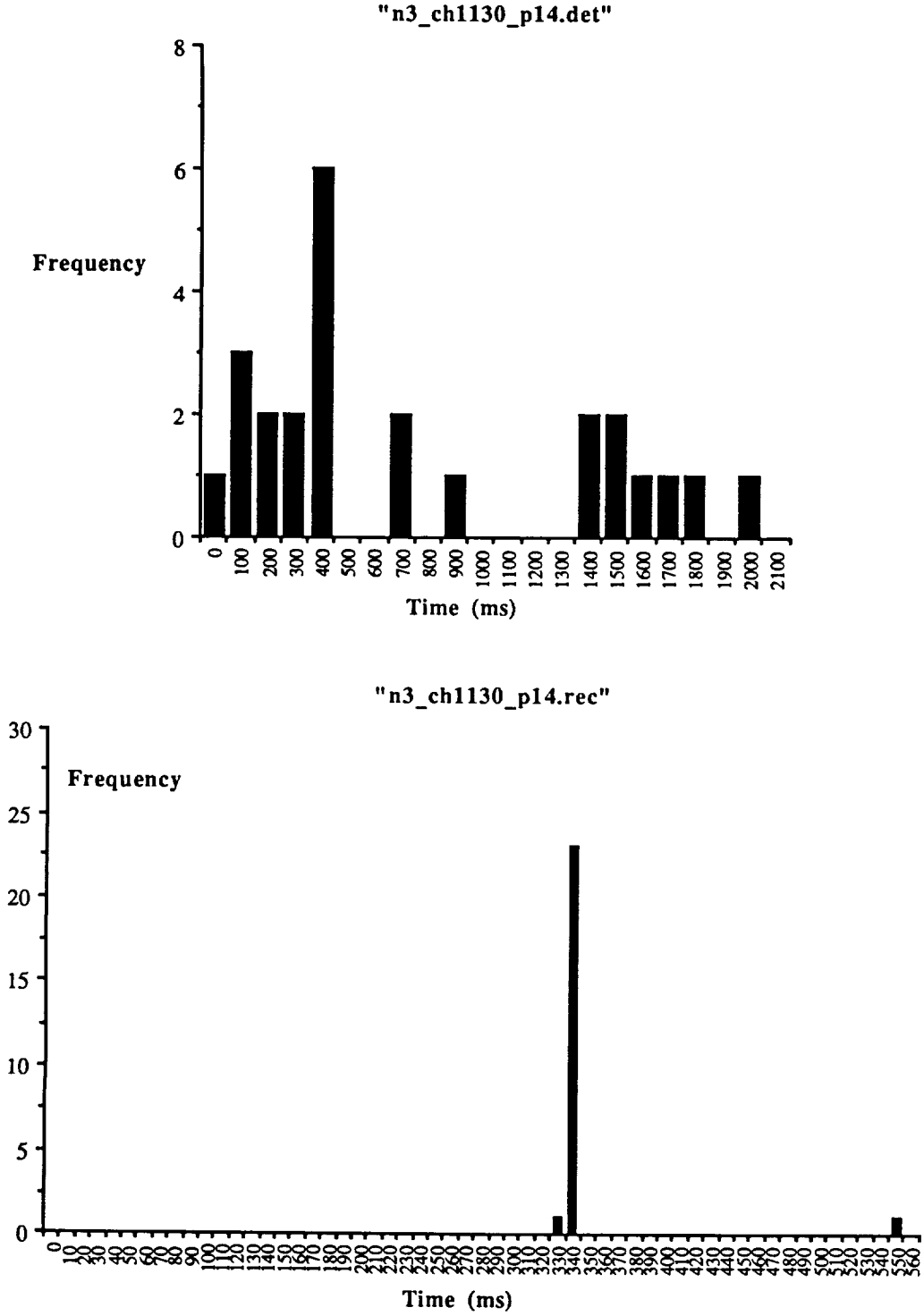

Figure 3-44. Test D.5.d

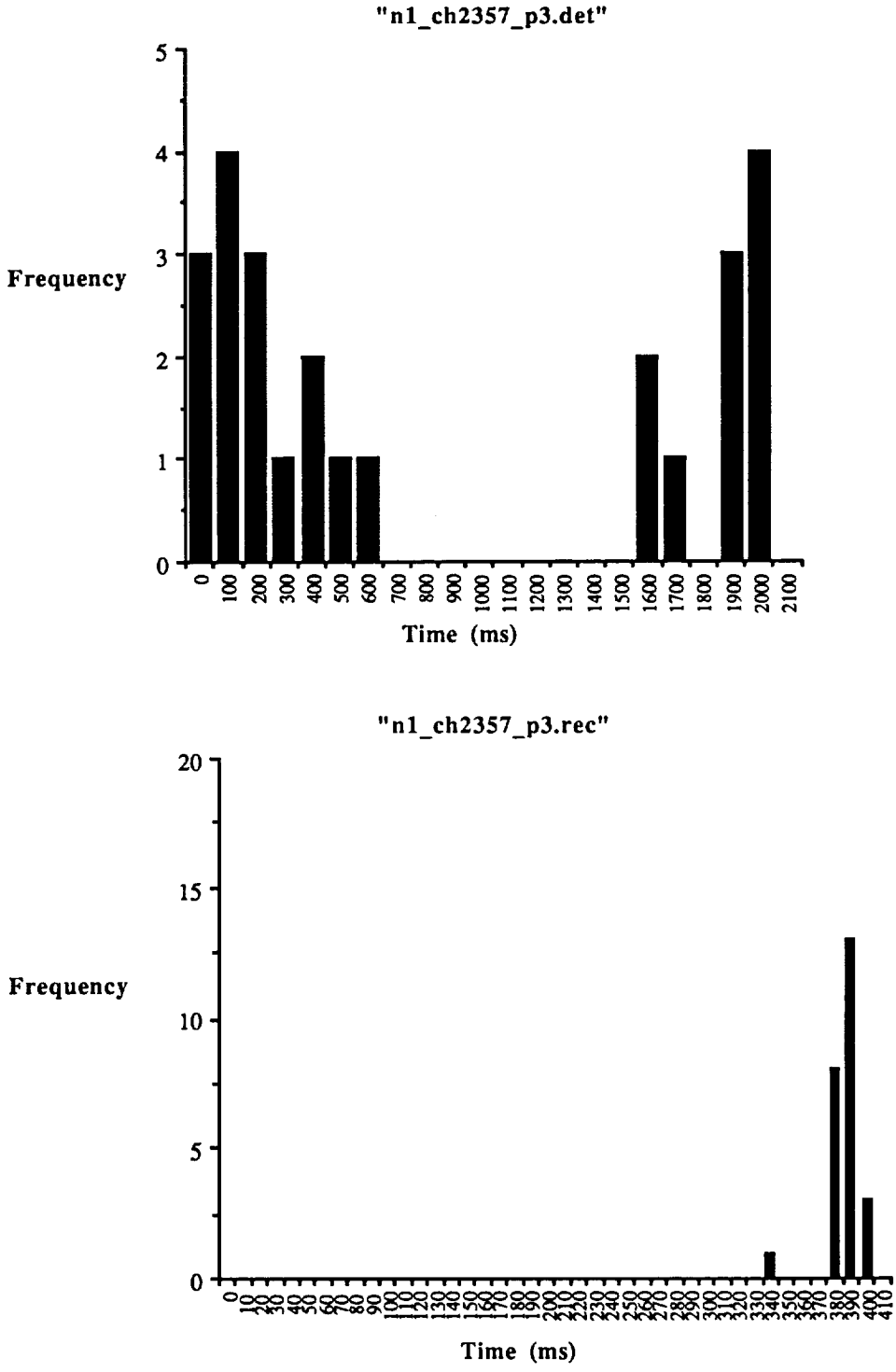

Figure 3-45. Test E.1.a

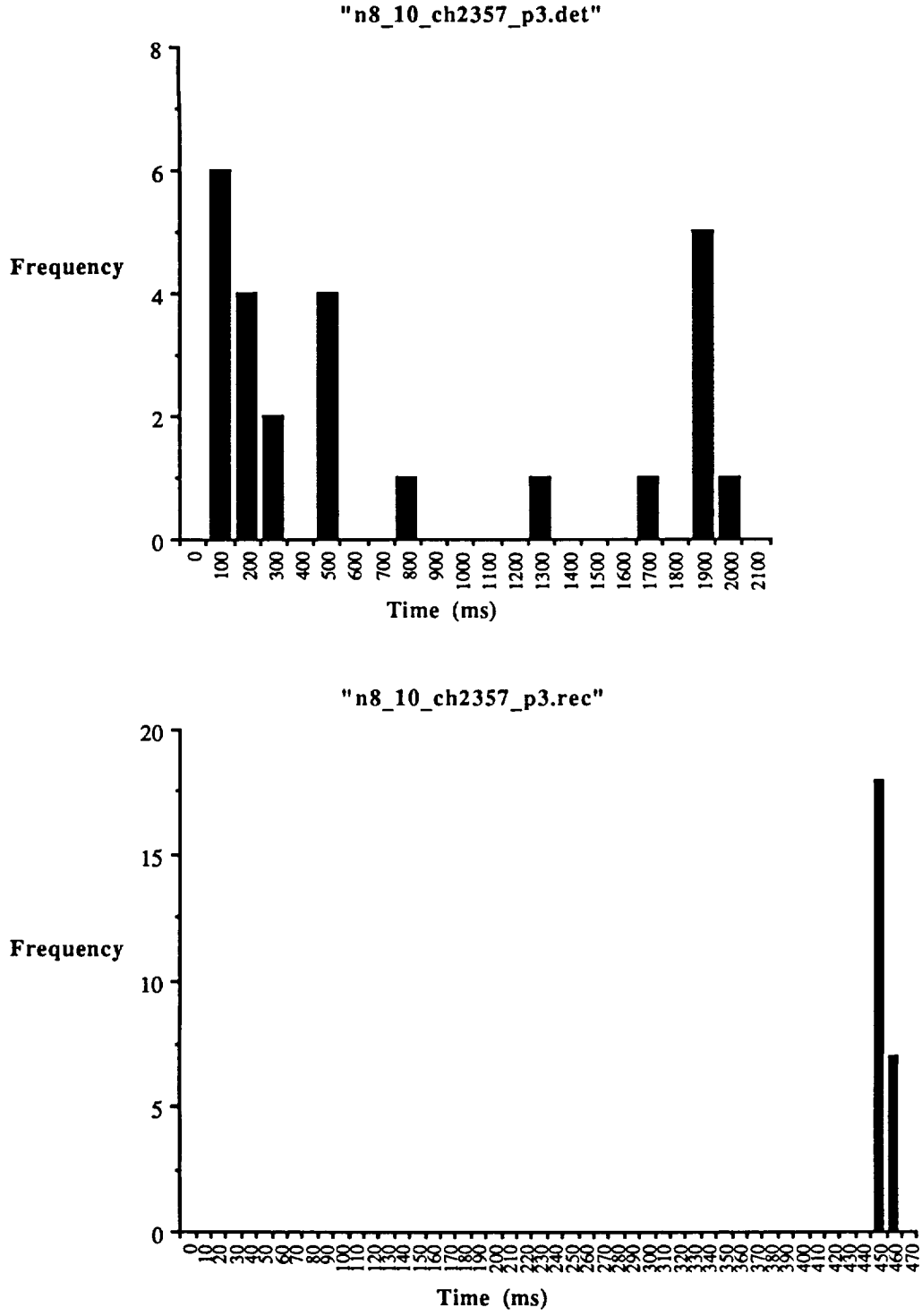

Figure 3-46. Test F.1.a

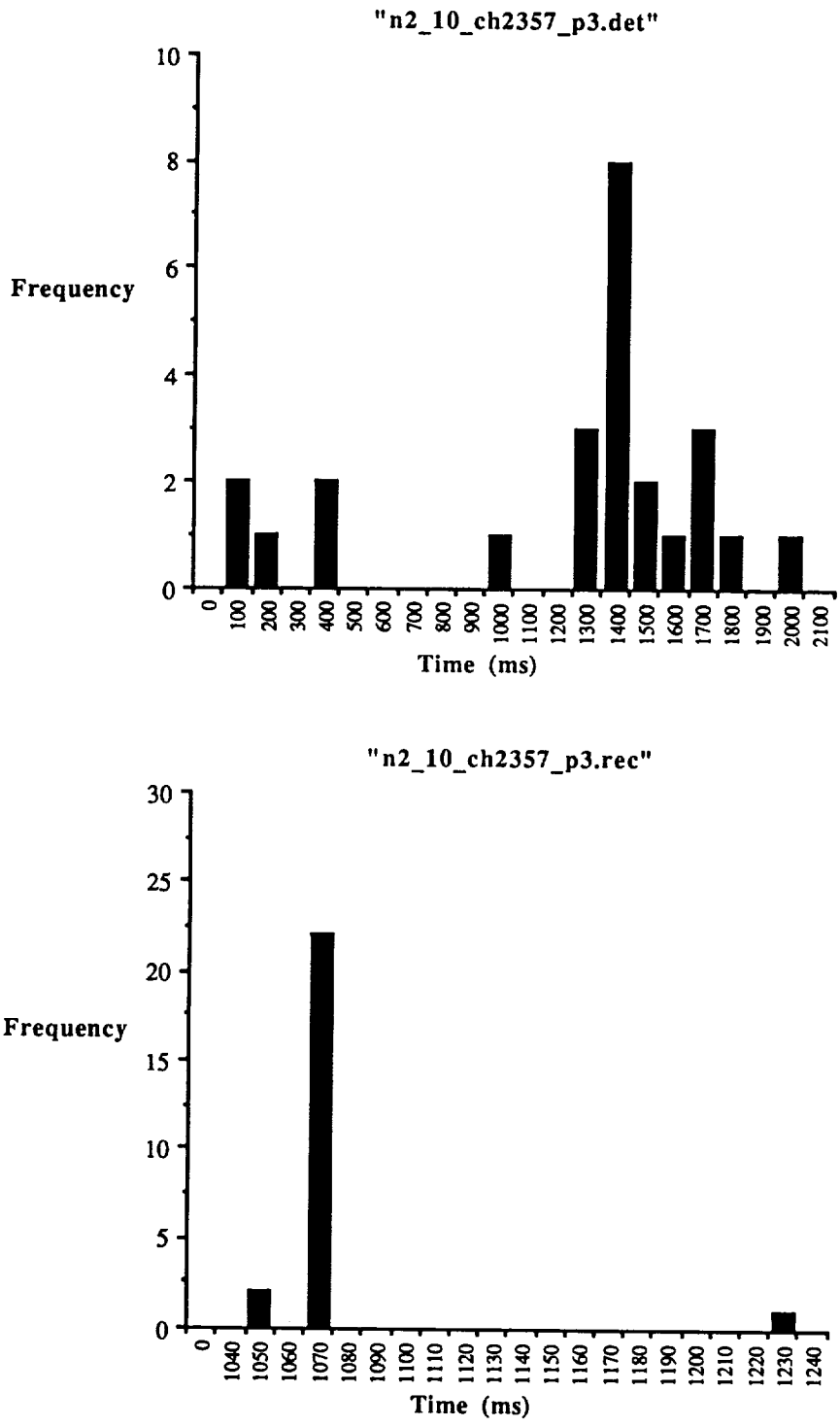

Figure 3-47. Test F.2.a

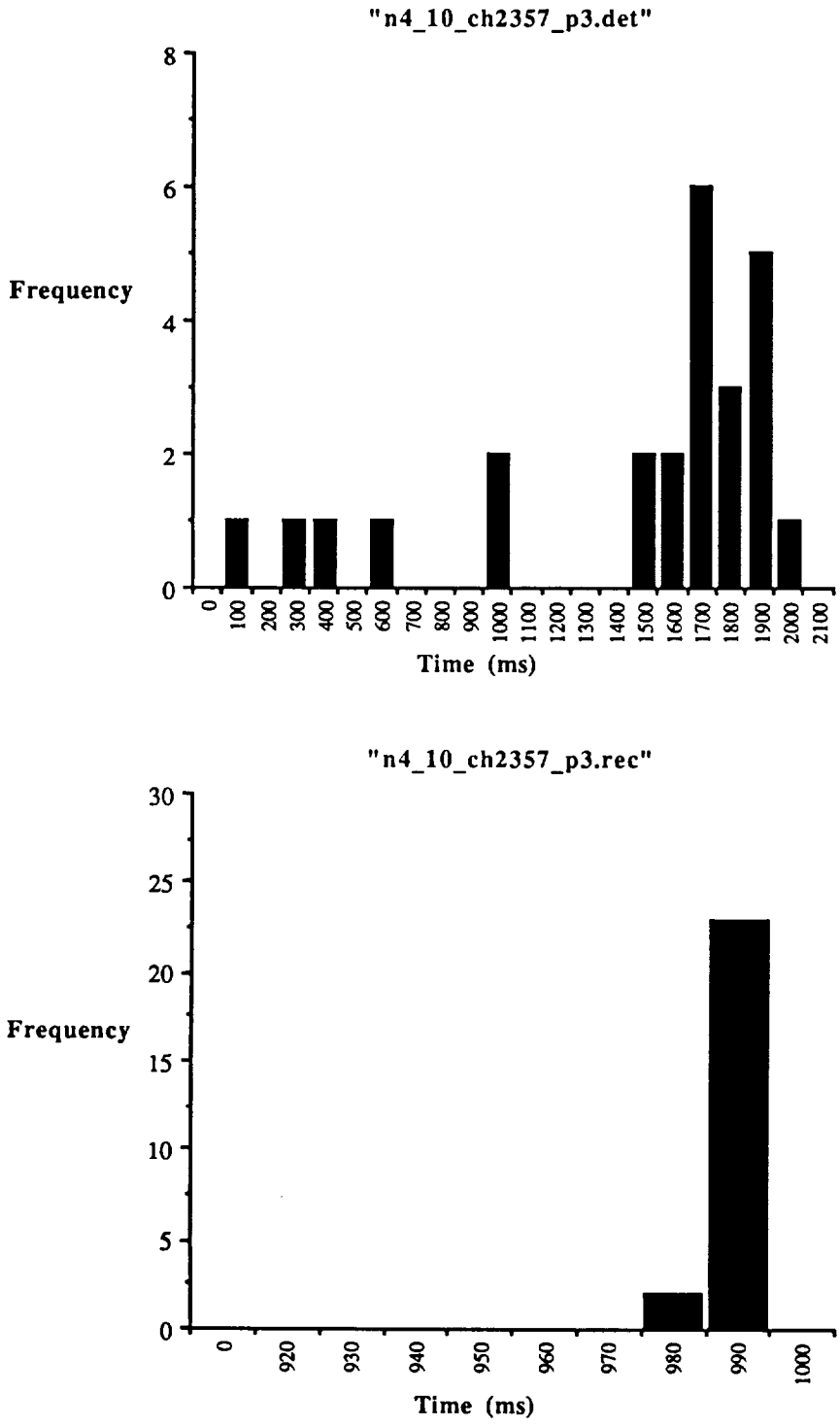

Figure 3-48. Test F.3.a

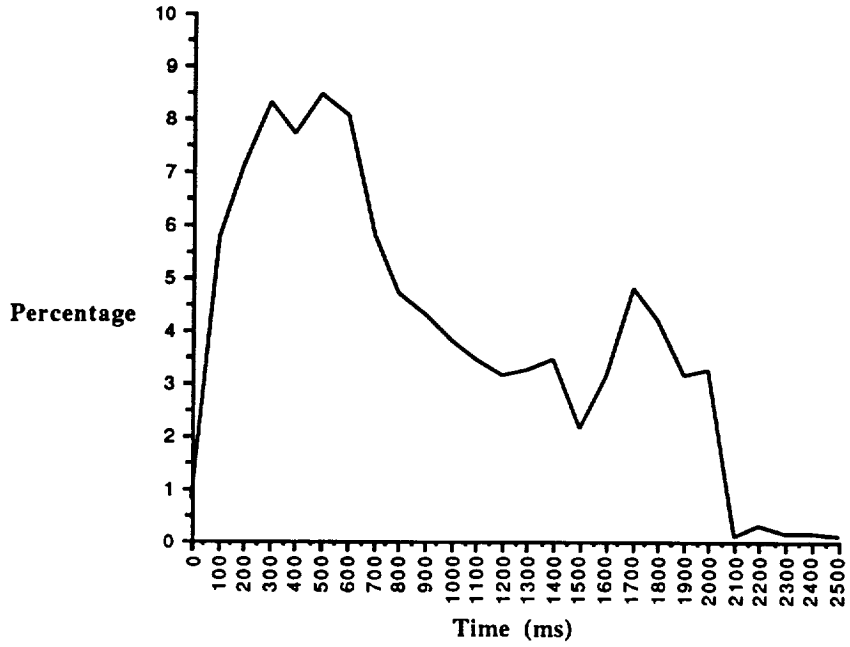

Figure 3-49. The Probability Density Function for the Detection Times

3.3.3 Probability and Cumulative Density Functions

The I/O Fault Insertion Plan is comprised of 47 tests. Each test consists of 25 iterations. For each iteration, the fault detection and reconfiguration times were recorded by the Fault Insertion Software. Accordingly, a total of 1175 (47 times 25) data sets were anticipated. As described in Section 3.3.1, however, five of the 1175 fault applications generated "don't care" error symptoms. As a result, 1170 sets of data, or 99.6 percent of the applied faults, were recorded.

Probability and cumulative density functions for these data sets were generated to complete the I/O Network Fault Insertion Analysis; these functions are illustrated in Figures 3-49 through 3-52. The probability density function for the I/O fault detection times depicts the wide variance in the observed times (discussed in Section 3.3.2.1). In the 0 to 2000 ms. range, which is the detection cycle for the AIPS Distributed Engineering Model, the percentage of events in each bucket is relatively consistent, ranging from 2.5 to 8.5 percent. In the 2000 ms. and over range, this percentage decreases to less than one-half percent. As a result, as shown by the cumulative density function for the detection times, approximately 97 percent of all faults were detected in 2000 ms. or less.

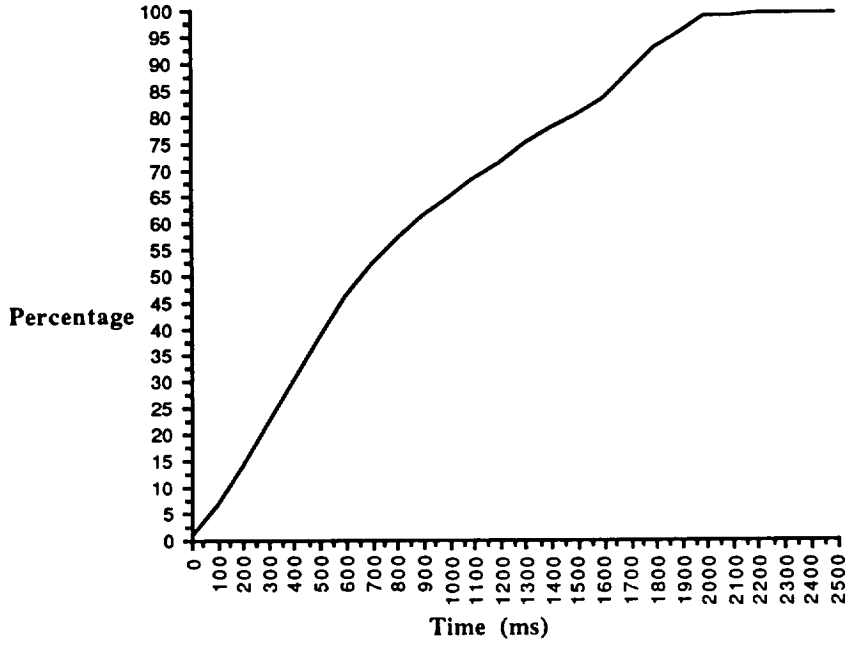

Figure 3-50. The Cumulative Density Function for the Detection Times

The probability density function for the I/O fault reconfiguration times indicate that a significant percentage of the faults were bypassed in 100 to 500 ms. while a substantial but smaller percentage were reconfigured around in 900 to 1100 ms. This latter range represents the percentage of times that the reconfiguration process employed the network regrowth algorithm that uses "low-level diagnostics" (several types of growth algorithms were designed for the AIPS I/O network). Furthermore, as depicted by the cumulative density function in Figure 3-52, approximately 85 percent of all faults were isolated and bypassed in 500 ms. or less.

3.4 I/O Network Fault Insertion: Conclusions

To conclude the discussion of the I/O Fault Insertion Plan, the Fault Insertion results are presented with respect to the goals of the Fault Injection Study. In brief, the objectives of the I/O analysis were:

1. to test the design specification for fault tolerance,
2. to obtain feedback for fault removal from the design implementation,
3. to obtain statistical data regarding fault detection, isolation, and reconfiguration (FDIR) responses, and
4. to obtain data regarding the effects of faults on system performance.

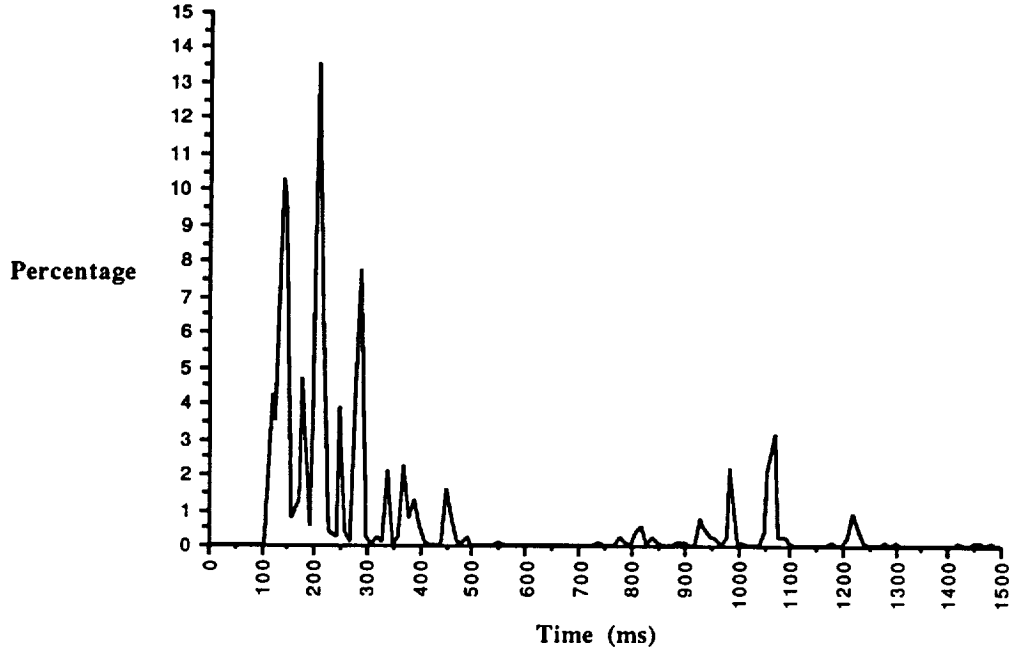

Figure 3-51. The Probability Density Function for the Reconfiguration Times

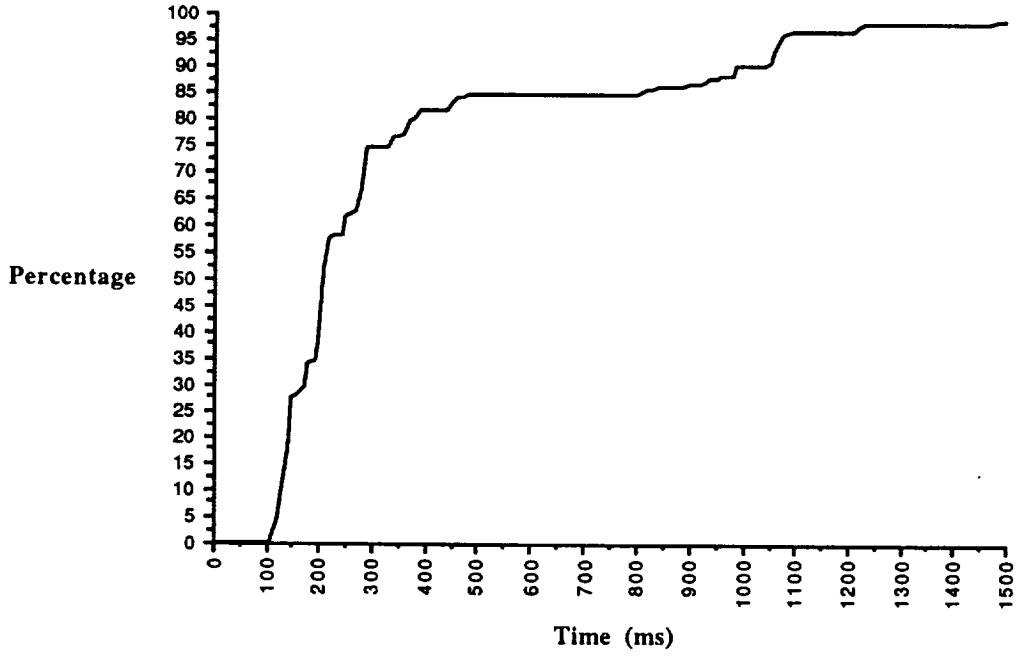

Figure 3-52. The Cumulative Density Function for the Reconfiguration Times

The process of validating the fault tolerant specifications and obtaining fault removal feedback is concerned with the presence of design flaws and the fault coverage. As detailed in Section 3.3, the limited I/O Fault Insertion process that was performed did not find any design errors. This does not necessarily prove that the design of the network is completely error free; however, the level of confidence in the design is greater now than it was before the fault injection experiments. The fault detection coverage was 99.6%; 0.4% of the faults did not produce any detectable error symptoms. The reconfiguration coverage for detected faults was 100%.

To confirm the coverage of the I/O faults, several devices were utilized. The detection of the fault was verified using the Fault Insertion Software (FIS) and light emitting diodes (LEDs). The FIS program records whether or not a fault was detected. Additionally, the manifestation of a fault is indicated by the I/O network LEDs, which are diodes that depict the configuration of the I/O network. The FIS logs and the network LEDs were examined by the I/O fault insertion supervisor to verify that each fault was detected.

The reconfiguration of the network was also validated using the FIS program and the I/O network LEDs. In addition, the I/O Network Redundancy Management logs were employed. Similar to the fault detection process, the FIS program was analyzed to determine if the applied fault was bypassed. Further, since the LEDs indicate the configuration of the network and the I/O reconfiguration strategy is deterministic, the expected reconfiguration was calculated by the fault injection supervisor and verified by examining the LEDs. Moreover, the I/O Redundancy Management logs, which identify the fault and summarize the reconfiguration process, were periodically examined by the fault injection supervisor to confirm the correctness of the I/O Network Management algorithm.

The third and fourth objectives of the I/O Fault Insertion study are concerned with recording data to measure the performance of the FDIR process and quantifying how the I/O faults affect system performance. As presented in Sections 3.3.1 through 3.3.3, the I/O Fault Insertion process recorded 1170 sets of data. The maximum and average detection and reconfiguration times, the corresponding frequency histograms, and the probability and cumulative distribution functions were calculated. This information accurately characterizes the performance of the I/O FDIR process on the AIPS Distributed Engineering Model. Furthermore, the data illustrates the effect that various I/O faults have on the overall performance of the Model.

As illustrated in Section 3.3.1, the I/O Fault Insertion results typically conformed to the expected maximum and average times. The anticipated maximum detection time was about 2075 ms. plus the error latency. Accordingly, the expected average time was approximately 1040 ms. plus the error latency. Due to the number of faults and the complexity of their error symptoms, the maximum and average reconfiguration times were not explicitly calculated for each fault (calculations would be numerous and gross approximations). The worst case reconfiguration time, however, was determined to be

approximately 3500 ms. - fault diagnostics plus the worst case regrowth scenario. As estimated, all reconfiguration times were less than the worst case. Since only the worst case reconfiguration time was calculated, the I/O reconfiguration times were primarily used to characterize the performance of the I/O FDIR process rather than to compare the measured data to their corresponding expectations.

The performance of the FDIR process significantly improves if the AIPS/ALS technology projections are considered. The throughput performance of the projected AIPS/ALS Fault Tolerant Processor is approximately *50* times the current capabilities of the AIPS Distributed Engineering Model, with respect to the I/O Redundancy Management code. Consequently, the maximum I/O fault detection time will be reduced from 2075 ms. to approximately 42 ms., and the corresponding average time will be about 21 ms. (plus error latency). The reconfiguration times will also be reduced by a factor of 50. For example, a failed link that causes a disjoint node, which is typically bypassed in about 150 ms. by the AIPS Engineering Model, can be reconfigured around in 3 ms. with the AIPS/ALS system. Since the AIPS/ALS Fault Tolerant Processor will significantly decrease the fault detection and reconfiguration times (with respect to the Engineering Model), the effect that I/O faults have on the system performance will also be greatly reduced.

4.0 APPLYING FAULTS TO THE CORE FTP

This chapter discusses the Core FTP Fault Injection Plan, which is a set of faults that were injected into the core of an AIPS Fault Tolerant Processor. Section 4.1 describes the AIPS Fault Tolerant Processor while Section 4.2 summarizes the Fault Detection, Identification and Reconfiguration algorithm. The test cases comprising the Core FTP Fault Injection Plan are specified in Section 4.3, and the results of the test cases are presented in Section 4.4. Section 4.5 contains conclusions and summary remarks about the test cases.

4.1 Overview of the AIPS Fault Tolerant Processor

The Fault Tolerant Processor (FTP) consists of a variable number of redundant processing channels depending on the reliability requirements of the application. The AIPS Engineering Model FTP is intended to be operated primarily as a triplex, but it provides fail-stop capability when operated as a duplex. A single channel can also be used for non-critical operations as a simplex computer.

Each channel of an FTP consists of three sections: a computational section, an input/output section, and shared resources. The first section contains a Computational Processor (CP), memory, timers and clocks. The second section contains an Input/Output Processor (IOP), memory, timers, and clocks. The shared resources include shared memory, data exchange hardware, timers, and external interface hardware. The redundant processors are tightly synchronized using a fault-tolerant clock. Data is exchanged among redundant channels on point-to-point links. The data exchange hardware also performs the bit-for-bit voting, fault detection and masking functions in a manner that satisfies all the requirements to protect the FTP from Byzantine failures. Apart from redundancy, there are other features that provide hardware and software fault tolerance. These include watchdog timers, processor interlocks, a privileged operating mode, handlers for hardware and software exceptions, and self tests.

A functional view of one channel of an AIPS FTP is shown in Figure 4-1. The CP and IOP are identical, conventional processor architectures, and each processor refers to the other as its companion. Interval timers are used for scheduling tasks and maintaining time-out limits on applications tasks (task watchdog timers). A hardware watchdog timer is provided to increase fault coverage and to cause a processor to fail-safe in case of hardware or software malfunctions. This timer resets the processor and disables all of its outputs, if it is not reset periodically. The watchdog timer is implemented independently of the basic processor timing circuitry. A monitor and interlock circuit in each channel provides the capability to disable the outputs of faulty processors. Any two correctly operating processors in a triplex FTP can disable the outputs of the third failed processor through this interlock mechanism. A processor that is failed active is thus prevented from transmitting erroneous data or commands on I/O networks, IC networks, and local I/O devices.

The CP and IOP share resources through a bus that can be accessed by either processor. These shared resources include memory; a system timer; the interchannel circuitry for the data exchange, fault-tolerant clock and monitor interlock; and interfaces to one or more I/O networks, memory mapped I/O devices, and the IC network.

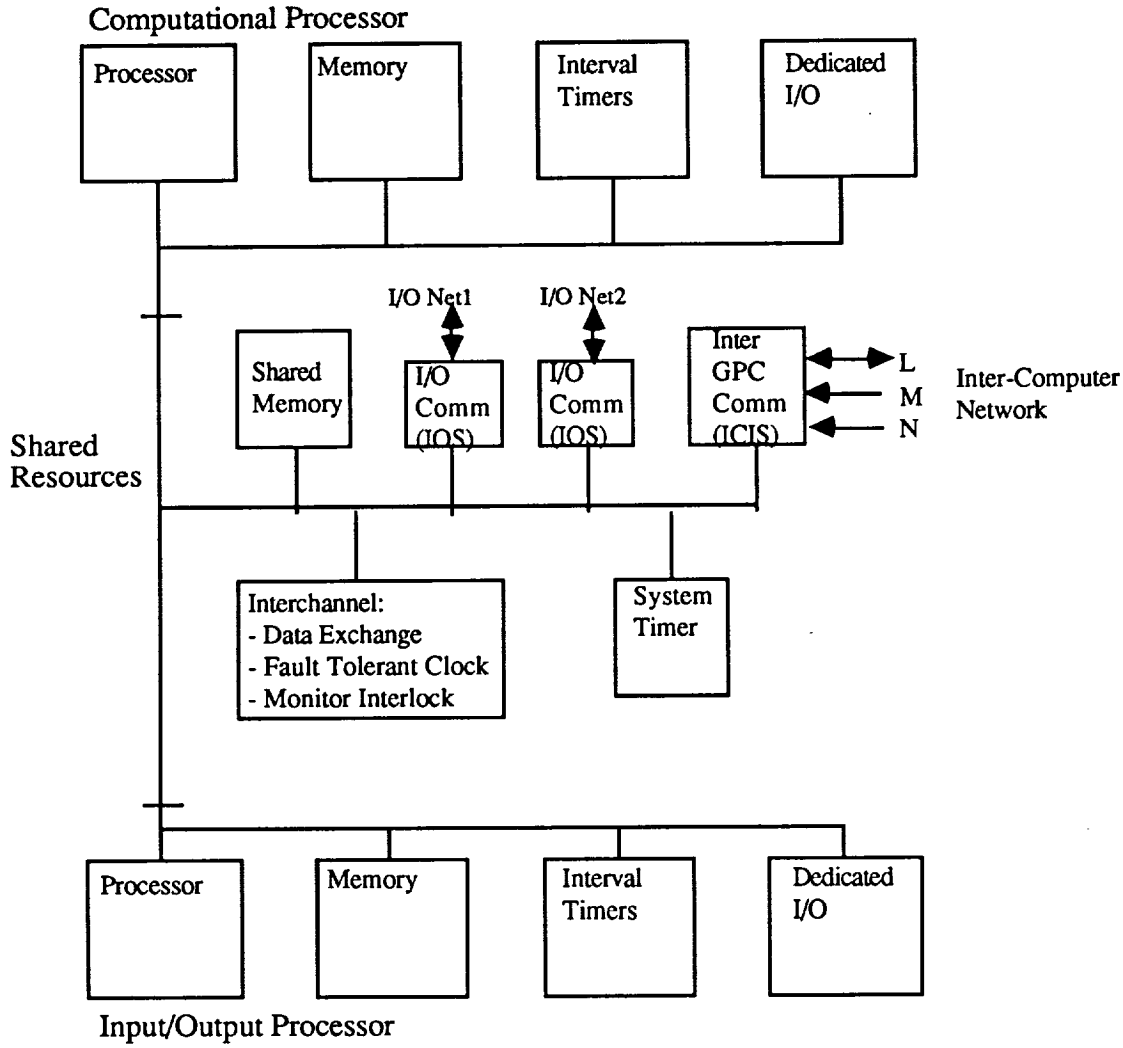

Figure 4-1. Fault Tolerant Processor: Functional View (One Channel)

One very important aspect of the FTP architecture is the interconnection hardware between redundant channels. The interchannel data exchange and voting hardware serves three purposes: it provides a path for distributing data in one channel to all other channels; it provides a mechanism for comparing results of the redundant channels; and it provides a path for distributing and comparing timing and control signals such as the fault tolerant clock and external interrupts.

The same software executes on a redundant FTP as on a simplex channel and application code is written as if it were to operate on a simplex computer. All redundant processors have identical software and execute identical instructions at exactly the same time. This feature of the architecture is carried out in the data exchange hardware and software as well. The data exchange hardware is designed such that all redundant processors execute identical instructions when exchanging data whether it is redundant data to be voted or simplex data being transmitted from one channel to others. Thus, for example, if a simplex exchange is to be made from channel A, all three channels write to their FROM_A register. While the contents of the FROM_A register are transmitted from A, voted, and deposited in the receive registers of all three processors, the contents of the FROM_A registers in channels B and C, which are meaningless, are ignored.

On a routine basis, the internally produced data that needs to be exchanged consists of error information and cross channel comparisons of results for fault detection. These operations can be easily confined to the program responsible for Fault Detection, Identification, and Reconfiguration (FDIR). Therefore, the remaining pieces of the Operating System software and the applications programs need not be aware of the existence of the data exchange registers.

4.2 FTP Fault Detection, Identification and Reconfiguration

The AIPS FTP uses hardware redundancy with fault detection and masking capabilities to provide fault tolerance. The fault tolerance provided by the hardware is greatly enhanced by the Fault Detection, Identification and Reconfiguration (FDIR) functions which are part of the FTP local operating system. While the hardware alone in a triplex FTP could sustain one fault, the FDIR software allows it to sustain multiple successive faults and identifies the fault(s) for an operator, thus making the FTP much more robust and serviceable.

The FDIR software in AIPS has two main functions:

- identifying a failed channel, i.e., detecting a fault, isolating it to a single channel, masking the faulty channel's inputs, and disabling its outputs.

- recovering a failed channel, i.e., determining that the fault no longer exists, bringing the channel into line with the two synchronized channels, accepting the channel's inputs, and enabling its outputs.

These functions are described in more detail below.

4.2.1 Fault Detection and Identification

Fault detection mechanisms are implemented in both hardware and software, while the identification mechanisms are implemented solely in software. Instruction-level synch-

ronization together with bit-for-bit comparison of redundant data makes it possible to isolate a fault to a single channel.

There are four main processes which detect and identify faults:

- *Fast FDIR.* A periodic, high-priority task which checks for failure of a companion, an unsynchronized channel, a fault in the data exchange hardware, and a fault in the fault-tolerant clock;

- *Watchdog Timer Reset.* A periodic, high-priority task which taps the watchdog timer within the given time bounds so that the timer does not overrun and cause a hardware reset;

- *Background Selftests.* A low-priority task which does tests to uncover latent faults in memory, voting circuitry and error latches, and the real-time clock; and

- *Hardware Exception Handler.* A procedure for handling M68010 hardware exceptions such as an illegal instruction or addressing error.

After a channel is identified as being faulty, the FTP is reconfigured so that the faulty channel does not affect FTP operation. The errors generated by the channel are masked, and its outputs are stopped. This is done by a procedure, *Reconfigure,* that sets a software variable identifying the channel as failed, disengages the monitor interlock so that outputs from the faulty channel are disabled, and logs the fault and the reconfiguration for later examination by an operator.

4.2.1.1 Fast FDIR

Fast FDIR is one of four tasks which detect and isolate errors. It is a high-priority task which runs every 40 ms on both the CP and IOP. It checks for:

- a fault reported by its companion processor
- an unsynchronized channel
- a fault detected but not reconfigured around by the selftests
- a data exchange fault, i.e., either a faulty in the interstage or in any link in the data exchange hardware
- a fault in the fault-tolerant clock
- a missing companion processor (i.e., a companion that is not executing FDIR at the prescribed rate).

Error detection is done in the order given above. If any particular test uncovers a fault, the remaining tests are not done during that iteration of Fast FDIR. When an error is detected,

the error is logged, the FTP is reconfigured to exclude the faulty channel, and the reconfiguration is logged.

4.2.1.2 Watchdog Timer Reset

The second fault detection process is the Watchdog Timer Reset process. This process does not perform fault detection functions in quite the same way, however, as other processes in this category, i.e., by responding to a specific fault. Rather, the failure of this task to execute at its scheduled period would indicate a critical fault in either hardware or software and would cause a hardware reset.

The watchdog timer is a hardware component whose purpose is to prevent infinite software loops or hardware faults from hanging up the system. After it has been started, the watchdog must be cleared periodically within a set time window; if it is not cleared within this window (i.e., either too early or too late) a hardware reset occurs. On the AIPS FTP this window is 60-120 milliseconds plus or minus 10%. The Watchdog Timer Reset process performs the function of clearing the watchdog timer.

4.2.1.3 Background Self Test

The third fault detection process is the Selftest task, a task of the lowest priority that runs when there are no higher priority tasks to be executed. Its job is to uncover latent faults, that is, faults which exist but which have not yet caused data exchange errors or desynchronization of a channel. This task tests memory, the voting circuitry and error latches, and the real-time clock.

The following memory tests are performed:

- *PROM sum check.* This test verifies that all channels have identical values in ROM by doing a sum check.
- *RAM scrub.* This test checks each memory location to ensure that the values are identical among the three channels.
- *RAM pattern test.* This test checks the functionality of each location. It tests each bit's ability to hold both a 1 and a 0 by writing specific patterns to each word.
- *Shared memory scrub.* This test checks each memory location in shared memory to ensure that the values are identical among the three channels.

The voter circuitry and error latches are tested by writing normal and faulty patterns of data to the voters. After these votes, both the resulting values and the error latches are checked to confirm that all errors were properly latched and corrected.

The real-time clock is tested by reading the current value and ensuring that it is identical among the three channels.

4.2.1.4 Hardware Exception Handler

The exception handler is the fourth fault detection process. It is invoked when there is a hardware exception such as an illegal instruction or an address error. A presence test is done to determine if the exception was caused by a hardware error or a generic software error. If the results of the presence test show that the processor is alone, this implies a hardware error. If the presence test shows that the processor is with others, this implies a generic software error. In either case, the exception and the results of the presence tests are logged in the non-congruent log, and the processor(s) are restarted.

4.2.2 Channel Recovery

The reliability of the FTP is greatly enhanced if channels previously diagnosed as faulty but currently operating without faults can be brought back into the FTP configuration. How a channel is recovered depends on the type of failure, i.e., whether or not the fault has caused the channel to fall out of sync. A failure in the data exchange hardware does not desynchronize a channel, while other kinds of failures do. An FTP has recovered from a fault, therefore, when

- the failed channel can be resynchronized, or
- the failed channel no longer shows errors in the data exchange hardware.

When a channel has been recovered, the FTP must be reconfigured so that the recovered channel participates in the FTP operation and another fault can be tolerated.

There are three main processes involved in channel recovery. Transient FDIR distinguishes between transient and hard faults when a channel recovery is being attempted in order to balance competing system needs. Lost Soul Sync is responsible for resynchronizing an unsynchronized channel, i.e., synchronizing it to the instruction level and making its internal state the same as the duplex processors. Finally, the Restart process is invoked when a second fault or a common-mode failure occurs. These faults result in a fail-safe condition, which the AIPS FTP responds to with a system restart.

4.2.2.1 Transient FDIR

When recovering a failed channel, system resources are used most efficiently if a distinction is made between transient faults and hard faults. Transient faults are assumed to be caused by some temporary environmental condition (e.g., a power surge). By

definition, they are expected to disappear with time. Hard faults, on the other hand, are caused by breakdowns of the FTP hardware that must be physically repaired.

The attempt to recover a failed channel could be made automatically (i.e., the software periodically tests the channel to determine its current state) or it could be made solely under operator direction (i.e., the operator enters a command indicating the channel has been repaired). The first method satisfies the need to recover the channel as quickly as possible while the second method satisfies the need to not waste system resources by repeatedly testing a channel with a hard fault. Transient FDIR strikes a balance between these two needs by initially assuming that any particular fault is transient (it has been observed that 50 to 80 percent of all faults in computer systems are transient) and automatically attempting a recovery. As time passes without the channel being recovered, it becomes more likely that the fault is a hard fault rather than a transient, and Transient FDIR makes the recovery attempt less often. After a certain period it can reasonably be assumed that the fault is a hard fault. Then Transient FDIR either waits for an operator signal or, in the case where there is no operator, tests the channel only at some infrequent interval such as its mean time to repair.

Additionally, it has been noted that hard faults tend to manifest themselves sporadically. A channel may be recovered according to the above criteria, but may immediately fail again. Transient FDIR attempts to prevent this situation by regarding a recovered channel as recovered only on a trial basis. If the channel passes its trial period without further errors, it is regarded as fully recovered and can be added back into the FTP configuration. Intermittent faults which occur at infrequent intervals (i.e., after the trial period has passed) will not be handled by this scheme, however, but will be regarded as new faults.

This distinction between transient and hard faults thus defines the two functions of Transient FDIR:

- It decides when it is appropriate to attempt to recover a failed channel.
- Once a channel is seen as fault-free, it monitors its health for a brief probation period before declaring it fully recovered.

Attempting Channel Recovery

The initial response to all detected faults is to mask the fault and disable all outputs from the faulty channel. Thereafter, the status of the failed channel is periodically "sampled" to determine if the fault is transient. Immediately after a failure, a recovery attempt is made and a sampling of the channel's health is taken. If the attempt fails (i.e., the unsynchronized channel cannot be found or the data exchange latches still show errors), the

time between successive attempts is doubled, until Mean Time To Repair (MTTR) is reached. This time delay between successive recovery tries and the samplings of the status of a failed channel is a function of state variables representing the "health" of the channel. The "health" variable, in turn, is a function of the error history of the particular channel with many recent fault observations for the channel indicating "poor" health and declining fault observations representing "good" health. The time between recovery attempts is doubled following each status sampling which indicates the fault is still present. This sampling sequence is repeated until either the fault status changes to indicated the fault is no longer present or an upper threshold on the retry time is crossed at which point the fault is deemed "hard". From this point on recovery will be attempted only when an operator has signaled that the channel has been repaired or, in the case where there is no operator, when another MTTR period has passed.

Probation Monitoring

After a channel has been recovered, it must undergo a trial period before being declared fully recovered and functional. The length of this period is a function of the "health" of the channel and depends on the number and type of faults. A channel with multiple faults in quick succession (i.e., the channel fails before it has passed its trial period) will have a longer trial period than if it only had a single fault. Faults that desynchronize a channel require a longer trial period than data exchange errors.

4.2.2.2 Lost Soul Sync

Lost Soul Sync is the process of attempting to resynchronize a previously failed channel (a "lost soul") and, if successful, bringing it to the same state as the two good channels. This process has two main steps:

- resynchronizing the channel, i.e., synchronizing it to the instruction level with the other two channels, and

- aligning the channel, i.e., making its volatile memory and registers the same as those of the other two channels. This ensures that after the code execution is synchronized, only a fault could cause a channel to lose synchronization, rather than, for example, a memory location that contained an incorrect value.

This task is described in detail in [4], for the reader who is unfamiliar with the operation of Lost Soul Sync.

4.2.2.3 System Restart

Certain faults which are detected may be of such a magnitude that they are unsustainable and may be recovered from only by restarting the system. Examples of such faults are a second fault detected by Fast FDIR and common-mode faults. The restart process accomplishes the system restart without requiring operator intervention.

4.2.3 Reconfiguration

The reconfiguration process is invoked by the fault detection and identification tasks when a fault has been identified and by the recovery tasks when a channel has been repaired. During a reconfiguration a channel is either removed from the configuration because it was found to be faulty, or added in because the fault no longer exists.

When a channel is identified as being faulty, the errors generated by that channel must be masked and its outputs must be stopped. This is done by (1) setting a software variable which identifies the channel as having failed, and (2) disengaging the monitor interlock so the channel's outputs are disabled.

When a channel has recovered from a failure, its inputs must be accepted and its outputs enabled. This is done by the reverse process, i.e., (1) setting the software variable to say that the channel is now functional, and (2) engaging the monitor interlock so the channel's outputs are enabled.

When a channel has recovered from a failure, it is considered part of the configuration only by FDIR until its probation period has expired. This is done by setting a software variable to say that the channel is enabled on a trial basis.

4.3 Specification of Core FTP Faults

Two methods were used to apply faults to the core FTP. The first method used the Fault Injector Software described in Chapter 2 and simulated memory faults by altering selected portions of one channel's memory. It is referred to in the following sections as Software Fault Injection.The second method used both the Fault Injector Hardware and Software described in Chapter 2. This method, referred to in the following sections as Hardware Fault Injection, created faults by altering the signals provided to a selected pin.

4.3.1 The Software Fault Injection Plan

The Software Fault Injection Plan inserts defects into a triplex FTP. A simulated memory fault is inserted by corrupting the memory of one channel of the FTP. The faults applied are

divided into three categories: data, constants, and code. This classification is based on the type of memory that is altered.

- *Data.* The data section used by a software module is corrupted. The data section has three segments: initialized, uninitialized, and debug. Initialized data are objects that are assigned values when they are declared. In contrast, uninitialized data are objects that are declared but not assigned; they are initialized by the program at a later time. Last, debug data is information used by the AIPS system for debugging.

- *Constants.* The constants section of a software module is corrupted.

- *Code.* The instructions in a particular software module are corrupted.

These simulated memory faults will typically cause one or more of the following symptoms:

- *Unsynchronized Channel.* The corrupted memory causes the channel to go out of synchronization (because data utilized by the program or the program itself is altered by the applied defect). This condition is identified by a Presence Test. The presence test detects an unsynchronized channel by sending a unique pattern from each channel through the data exchange. If the result read from the data exchange receiver is not the expected pattern, the channel originating the exchange is judged not present and therefore out of sync.
- *Inconsistent RAM.* The corrupted memory differs from its analogous values in the other channels (the data should be the same). This condition is identified by the RAM Scrub process. This test checks each memory location to ensure that the values are identical among the three channels.
- *Incorrect PROM sum.* The altered memory changes program instructions. This error symptom is determined by the PROM Sum Check. This test verifies that all channels have identical values in ROM by doing a sum check. (In the AIPS Engineering model, code actually resided in RAM but was treated as if it was in PROM.)
- *Unknown DX.* The faulty channel temporarily goes out of sync (in particular, during a data exchange) but is forced back into sync prior to the subsequent presence tests. This sequence of events is detected when the faulty channel's data exchange (DX) latches are compared to other channels (to determine if the DX latches agree).

The Software Fault Injection Plan is comprised of 178 tests; 117 tests involve the CP while 61 affect the IOP. Each test was deterministically selected; that is, the software module to be corrupted was chosen by the fault injector supervisor and an address range that would disrupt the module's execution was determined. (If random fault selection was performed, the address range would be arbitrarily selected.)

The Core FTP Software Fault Injection Plan is detailed in the following table. The table presents five sets of information.

- *Test Numbers.* The tests are divided into two sections: CP and IOP. Each section is segmented into three subcategories: data, constants, and code. The tests are identified by the section and subcategory.

    - CP_1 to CP_99 affect the CP's data memory;
    - CP_100 to CP_199 involve the CP's constant sections;
    - CP_200 to CP_299 corrupt the CP's program space;
    - IOP_1 to IOP_99 affect the IOP's data memory;
    - IOP_100 to IOP_199 involve the IOP's constant sections; and
    - IOP_200 to IOP_299 corrupt the IOP's program space;

- *Module Name.* The name of the software module that is affected by the application of the defect. The modules selected were chosen from the entire range of AIPS system software components, including the Ada Run-Time System, FDIR, the CRT Display tasks, Inter-Computer Communication Services, and application tasks.
- *Data Type.* The type of information that is corrupted by the fault: BSS (uninitialized data), DEBUG (debug log), DATA (initialized data), CONST (constants region), and CODE (program section).
- *Addresses Corrupted.* This is the address range that is altered by the Fault Injection Software.
- *Faulty Data.* The value of the faulty data written into the address range. This is typically 0 or FFFF.

The Software Fault Injection Plan only applied faults to a subset of the AIPS FTP software, because it was more concerned with a general characterization of the FTP Redundancy Management process' performance rather than a comprehensive one. Given enough resources, an extensive Core FTP Fault Injection Plan could be developed and applied to the AIPS Engineering Model, but this was not done for the present project.

Test No.	Module Name	Data Type	Addresses Corrupted	Faulty Data
CP_1	CALENDAR_B_K	BSS	1E4F14 - 1E4F22	FFFF
CP_2	CALENDAR_K	BSS	1E4058 - 1E4062	FFFF
CP_3	CONFIG_B	BSS	1E8F74 - 1E8F76	FFFF
CP_4	DEBUG_TRACE_B	DEBUG	7100 - 7D00	FFFF
CP_5	LSS_CONFIG	BSS	1E42AC - 1E42F2	FFFF
CP_6	LSS_CONFIG_B	BSS	1E5070 - 1E5082	FFFF
CP_7	LSS_CLOCK_ERR	BSS	1E4E84 - 1E4EB6	FFFF
CP_8	LSS_CLOCK_ERR_B	BSS	1E4EB8 - 1E4EE2	FFFF
CP_9	LSS_DX_ERR	BSS	1E4AD8 - 1E4B26	FFFF
CP_10	LSS_DX_ERR_B	DATA	1DE280 - 1DE2CE	FFFF
CP_11	LSS_EVENT_CNTL	DATA	1DE010 - 1DE01E	FFFF
CP_12	LSS_EVENT_CNTL_B_K	BSS	1E4DB8 - 1E4DCA	FFFF
CP_13	LSS_FFDI_B	BSS	1E8AD8 - 1E8B22	FFFF
CP_14	LSS_FFDI	BSS	1E41D4 - 1E41EA	FFFF
CP_15	LSS_FDIR_GLOBALS	BSS	1E4D60 - 1E4D82	FFFF
CP_16	LSS_GLBOAL_MEM	DATA	1DE31C - 1DE362	FFFF
CP_17	LSS_MEMORY	BSS	1E4138 - 1E416E	FFFF
CP_18	LSS_NON_CONGRUENT_DATA	BSS	1E42F4 - 1E43F2	FFFF
CP_19	STATUS_DATABASE_MGR_K	BSS	1E43F4 - 1E4426	FFFF
CP_20	STATUS_DATABASE_MSG_B_K	BSS	1E4428 - 1E4AAE	FFFF
CP_21	LSS_SYNC	BSS	1E4D84 - 1E4D9E	FFFF
CP_22	LSS_SYNC_B	BSS	1E8A08 - 1E8A16	FFFF
CP_23	LSS_TEST	BSS	1E5084 - 1E50C2	FFFF
CP_24	LSS_TEST_B	BSS	1E8A90 - 1E8AD2	FFFF
CP_25	LSS_TEST2	BSS	1E51C8 - 1E51FE	FFFF
CP_26	LSS_TEST2_B	DATA	1DE6BC - 1DE88A	FFFF
CP_27	LINK_BLOCK	BSS	1E41A4 - 1E41AE	FFFF
CP_28	LINK_BLOCK_B	BSS	1E4238 - 1E425E	FFFF
CP_28A	OS_B	DATA	1DEE88 - 1DEF3E	FFFF
CP_29	TIMER_SUP	BSS	1E914C - 1E915E	FFFF
CP_30	TIMER_SUP_B	DATA	1DF134 - 1DF146	FFFF

Test No.	Module Name	Data Type	Addresses Corrupted	Faulty Data
CP_31	TS_MD	BSS	1E91B0 - 1E91C6	FFFF
CP_32	ICCS_CP_IOP_COMMON	DATA	1DE364 - 1DE63E	FFFF
CP_33	TS_SIGNAL_B	DATA	1DEC58 - 1DEC86	FFFF
CP_34	TS_MD_CLK	DATA	1DF168 - 1DF172	FFFF
CP_35	SYS_TABLE	DATA	1DF1EC - 1DF252	FFFF
CP_36	LSS_TIME_MGR	BSS	1E4EF8 - 1E4F12	FFFF
CP_37	ICSS_USER_SERVICES	BSS	1E5200 - 1E5216	FFFF
CP_38	ICCS_CP_IOP_COMMON	BSS	1E5218 - 1E5500	FFFF
CP_39	LSS_TIME_MGR_B	BSS	1E8A18 - 1E8A3A	FFFF
CP_40	ICDEMO_STATUS_INFO	BSS	1E50C4 - 1E51C6	FFFF
CP_41	ICCS_USER_SERVICES_B	BSS	1E8AD4 - 1E8AD6	FFFF
CP_42	ICDEMO_ST_BRCAST	BSS	1E8B54 - 1E8B6A	FFFF
CP_43	ICDEMO_ST_BRCAST	BSS	1E8C04 - 1E8C1E	FFFF
CP_44	ICCS_DISP_MAIN_CP_B	BSS	1E8D34 - 1E8D3A	FFFF
CP_45	ICCS_TFDI_CP_B	BSS	1E8A3C - 1E8A7A	FFFF
CP_46	ICCS_FDIR_TIME_CP_B	BSS	1E8B44 - 1E8B46	FFFF
CP_47	TS_MD_CLK_B	BSS	1E9184 - 1E9192	FFFF
CP_100	DEBUG_TRACE_B	CONST	1B2B80 - 1B2BB2	FFFF
CP_101	ICCS_ICIO_MAIN_PROG_CP	CONST	1C0458 - 1C047A	FFFF
CP_102	CALENDAR_K	CONST	1B1574 - 1B15A2	FFFF
CP_103	MACHINE_CODE	CONST	1B1610 - 1B1632	FFFF
CP_104	MACHINE_CODE	CONST	1B1F1C - 1B1F3E	FFFF
CP_105	SYSTEM	CONST	1B1634 - 1B1672	FFFF
CP_106	TEXT_IO_K	CONST	1B1674 - 1B16A6	FFFF
CP_107	SYSTEM_B	CONST	1B1F40 - 1B1FE2	FFFF
CP_108	LSS_TASK_IDS	CONST	1B202C - 1B106E	FFFF
CP_109	LSS_MEMORY	CONST	1B1628 - 1B270E	FFFF
CP_110	LSS_TASK_IDS_B_K	CONST	1B1734 - 1B2866	FFFF
CP_111	ICCS_FDIR_TIME_CP	CONST	1B28E0 - 1B290A	FFFF
CP_112	LSS_EXCHANGE	CONST	1B2A18 - 1B2A66	FFFF
CP_113	LSS_EVENT_CNTL	CONST	1B2AE4 - 1B2B36	FFFF

Test No.	Module Name	Data Type	Addresses Corrupted	Faulty Data
CP_114	LSS_SCHEDULER	CONST	1B35A8 - 1B370E	FFFF
CP_115	LSS_SCHEDULER_B_K	CONST	1B3710 - 1B37E2	FFFF
CP_116	LSS_CONFIG	CONST	1B3AF4 - 1B3D76	FFFF
CP_117	LSS_NON_CONGRUENT_DATA	CONST	1B3D78 - 1B3FB2	FFFF
CP_118	STATUS_DATABASE_MGR..._K	CONST	1B3FD8 - 1B41DA	FFFF
CP_119	STATUS_DATABASE_MGR..._B_K	CONST	1B44DC - 1B45DE	FFFF
CP_120	LSS_FDIR_GLOBALS	CONST	1B679C - 1B6846	FFFF
CP_121	LSS_GLOBAL_MEM_UTIL	CONST	1B6B7C - 1B6BBE	FFFF
CP_122	LSS_EVENT_CNTL_B_K	CONST	1B6BC0 - 1B6D12	FFFF
CP_123	LSS_GLOBAL_MEM_UTIL_B_K	CONST	1B6D5C - 1B6D9E	FFFF
CP_124	CALENDAR_B_K	CONST	1B7B54 - 1B7F26	FFFF
CP_125	LSS_SYNC	CONST	1B6A44 - 1B6B56	FFFF
CP_126	LSS_TIME_UTIL_B	CONST	1B7F80 - 1B8366	FFFF
CP_127	LSS_GLOBAL_MEM	CONST	1B8728 - 1B874A	FFFF
CP_128	LSS_CONFIG_B	CONST	1B8924 - 1B8A6E	FFFF
CP_129	LSS_TIME_MGR_B	CONST	1BB634 - 1BB802	FFFF
CP_130	LSS_EXCHANGE_B	CONST	1BBE14 - 1BBED6	FFFF
CP_131	LSS_DX_ERR	CONST	1B4ADC - 1B4B12	FFFF
CP_132	LSS_DX_ERR_B	CONST	1B6848 - 1B6A1E	FFFF
CP_133	LSS_CLOCK_ERR	CONST	1B7740 - 1B77D6	FFFF
CP_134	LSS_CLOCK_ERR_B	CONST	1B77D8 - 1B788A	FFFF
CP_135	LSS_SYNC_B	CONST	1BB29C - 1BB60E	FFFF
CP_136	ICCS_TFDI_CP_B	CONST	1BBCCC - 1BBD76	FFFF
CP_137	LSS_FFDI_B	CONST	1BCA4B - 1BCD32	FFFF
CP_138	TS_MD_INT_B	CONST	1C16F8 - 1C184E	FFFF
CP_139	CONFIG_B	CONST	1C1850 - 1C18D6	FFFF
CP_140	OS_SUP_B	CONST	1C1D60 - 1C1E46	FFFF
CP_141	OS_B	CONST	1C1E48 - 1C21BA	FFFF
CP_142	TIMER_SUP	CONST	1C2334 - 1C239E	FFFF
CP_143	TIMER_SUP_B	CONST	1C24A0 - 1C265E	FFFF

Test No.	Module Name	Data Type	Addresses Corrupted	Faulty Data
CP_200	CALENDAR_B_K	CODE	10D800 - 10DCFA	FFFF
CP_201	CONFIG_B	CODE	14191C - 141B08	FFFF
CP_202	DEBUG_TRACE_B	CODE	101330 - 1013E2	FFFF
CP_203	LSS_CONFIG_B	CODE	110326 - 110700	FFFF
CP_204	LSS_CLOCK_ERR_B	CODE	10CA00 - 10CAAA	FFFF
CP_205	LSS_DX_ERR_B	CODE	108EF4 - 109200	FFFF
CP_206	LSS_EVENT_CNTL_B_K	CODE	109C0C - 109F00	FFFF
CP_207	LSS_FFDI_B	CODE	122524 - 122724	FFFF
CP_208	STATUS_DATABASE_MGR_B_K	CODE	1045C4 - 104A00	FFFF
CP_209	LSS_SYNC_B	CODE	1189EC - 118AEC	FFFF
CP_210	LSS_TEST_B	CODE	11E800 - 11F000	FFFF
CP_211	LSS_TEST2_B	CODE	11FE10 - 120318	FFFF
CP_212	LINK_BLOCK_B	CODE	10126C - 1012F0	FFFF
CP_213	TIMER_SUP_B	CODE	146E64 - 14711A	FFFF
CP_214	ICCS_FDIR_TIME_CP_B	CODE	124754 - 124A84	FFFF
CP_215	SYSTEM_B	CODE	1003DC - 100534	FFFF
CP_216	TS_SIGNAL_B	CODE	141F64 - 142230	FFFF
CP_217	TS_MD_CLK_B	CODE	1476F0 - 1478B2	FFFF
CP_218	LSS_TIME_MGR_B	CODE	11AD96 - 11B152	FFFF
CP_219	LSS_USER_SERVICES_B	CODE	121972 - 121C72	FFFF
CP_220	ICDEMO_ST_BRCAST	CODE	12679C - 126DB0	FFFF
CP_221	ICCS_DISP_MAIN_B	CODE	134EF4 - 1351F4	FFFF
CP_222	ICCS_TFDI_CP_B	CODE	11CEFE - 11CFFE	FFFF
CP_223	ICDEMO_CCP_APPLIC	CODE	126E98 - 127100	FFFF
CP_224	LSS_WATCHDOG_TIMER_B	CODE	101510 - 1015E8	FFFF
CP_225	LSS_SERIAL_IO_B_K	CODE	105E34 - 106100	FFFF
CP_226	TEXT_IO_B_K	CODE	106C76 - 106DCE	FFFF
CP_227	LSS_DISP_ROUT_B	CODE	111BA0 - 111D0A	FFFF

Test No.	Module Name	Data Type	Addresses Corrupted	Faulty Data
IOP_1	DEBUG_TRACE_B	DEBUG	7100 - 7D00	FFFF
IOP_2	ICCS_CP_IOP_COMMON	DATA	1DE318 - 1DE5F2	FFFF
IOP_3	LSS_TEST2_B	DATA	1DE708 - 1DE8D6	FFFF
IOP_4	ICCS_MSG_SEND_RCV_B (OT)	DATA	1DF0E4 - 1DF13A	FFFF
IOP_5	TS_EVENT_CONTROL_B	DATA	1DF348 - 1DF35A	FFFF
IOP_6	SYS_TABLE	DATA	1DF868 - 1DF8CE	FFFF
IOP_7	ICCS_DATA_TYPES	BSS	1E42B0 - 1E42BA	FFFF
IOP_8	STATUS_DB_MGR_B_K	BSS	1E44E4 - 1E4B6A	FFFF
IOP_9	LSS_FDIR_GLOBALS	BSS	1E4E1C - 1E4E3E	FFFF
IOP_10	ICCS_MSG_SEND_RCV	BSS	1E4EAC - 1E5FF2	FFFF
IOP_11	ICCS_ERROR_LOG_B	BSS	1E60B8 - 1E61B2	FFFF
IOP_12	LSS_TEST	BSS	1E620C - 1E624A	FFFF
IOP_13	LSS_TEST2	BSS	1E6350 - 1E6386	FFFF
IOP_14	ICCS_CP_IOP_COMMON	BSS	1E63B8 - 1E884E	FFFF
IOP_15	LSS_TIME_MGR_B	BSS	1E9E20 - 1E9E42	FFFF
IOP_16	LSS_TEST_B	BSS	1E9E6C - 1E9EAE	FFFF
IOP_17	LSS_FFDI_B	BSS	1E9EB4 - 1E9EFE	FFFF
IOP_18	ICCS_MSG_SEND_RCV_B (OT)	BSS	1EA908 - 1EAB5E	FFFF
IOP_19	TS_KRN_B	BSS	1EB08C - 1EB14A	FFFF
IOP_20	TS_MD_CLK_B	BSS	1EB184 - 1EB192	FFFF
IOP_101	DEBUG_TRACE_B	CONST	1B35A0 - 1B35D2	FFFF
IOP_102	STATUS_DB_MGR_B_K	CONST	1B4FD4 - 1B51D6	FFFF
IOP_103	LSS_SERIAL_IO_B_K	CONST	1B578C - 1B583E	FFFF
IOP_104	LSS_EVENT_CNTL_B_K	CONST	1B777C - 1B78CE	FFFF
IOP_105	ICCS_MSG_SEND_RCV	CONST	1B795C - 1B79EA	FFFF
IOP_106	CALENDAR_B_K	CONST	1B853C - 1B890E	FFFF
IOP_107	ICDEMO_STATUS_INFO	CONST	1BA344 - 1BA386	FFFF
IOP_108	LSS_TEST2	CONST	1B94C4 - 1B94E6	FFFF
IOP_109	ICCS_CP_IOP_COMMON	CONST	1B9554 - 1B96AE	FFFF
IOP_110	LSS_TIME_MGR_B	CONST	1BE918 - 1BEAE6	FFFF
IOP_111	LSS_EXCHANGE	CONST	1BF080 - 1BF142	FFFF

Test No.	Module Name	Data Type	Addresses Corrupted	Faulty Data
IOP_112	LSS_TEST_B	CONST	1BF7A0 - 1BFA3A	FFFF
IOP_113	LSS_TEST2_B	CONST	1BFA3C - 1BFEC6	FFFF
IOP_114	ICCS_USER_SERVICES_B	CONST	1BFEC8 - 1BFF5A	FFFF
IOP_115	LSS_FFDI_B	CONST	1BFF5C - 1C0246	FFFF
IOP_116	ICDEMO_ST_BRCAST_IOP	CONST	1C19B0 - 1C1A52	FFFF
IOP_117	ICCS_MESSAGE_SEND_RCV_B (OT)	CONST	1C71C0 - 1C738A	FFFF
IOP_118	ICCS_DISP_MAIN_IOP_B	CONST	1C8188 - 1C82F2	FFFF
IOP_119	TIMER_SUP_B	CONST	1CF444 - 1CF602	FFFF
IOP_201	SYSTEM_B	CODE	100418 - 10057A	0000
IOP_202	LSS_WATCHDOG_TIMER_B	CODE	1016B8 - 10180A	0000
IOP_203	LSS_SERIAL_IO_B_K	CODE	106180 - 1061EA	FFFF
IOP_204	ICCS_ERROR_LOG	CODE	10E034 - 10E0D2	FFFF
IOP_205	LSS_TIME_MGR_B	CODE	123B50 - 123BFE	FFFF
IOP_206	ICCS_CP_IOP_COMMON_B	CODE	125688 - 1256CE	FFFF
IOP_207	LSS_EXCHANGE_B	CODE	125FE0 - 1260C6	FFFF
IOP_208	LSS_TEST_B	CODE	1270CC - 1270E2	0000
IOP_209	LSS_TEST2_B	CODE	1286E4 - 128792	0000
IOP_210	ICCS_USER_SERVICES_B	CODE	12A294 - 12A59A	0000
IOP_211	ICDEMO_ST_BRCAST_IOP	CODE	12E230 - 12E2D2	0000
IOP_212	ICCS_MSG_SEND_RCV_B (OT)	CODE	156708 - 156906	0000
IOP_213	ICCS_MSG_SEND_RCV_S3	CODE	173874 - 173B94	0000
IOP_214	ICCS_MSG_SEND_RCV_S2	CODE	173DB0 - 173FAC	0000
IOP_215	ICCS_MSG_SEND_RCV_S2	CODE	173FAC - 17429C	0000
IOP_216	ICCS_MSG_SEND_RCV_S1	CODE	18BC40 - 18BD70	0000
IOP_217	ICCS_MSG_SEND_RCV_S1	CODE	18D180 - 18D46C	0000
IOP_218	IOSS_NET_MGR_CONFIG_B	CODE	188714 - 188914	0000
IOP_219	IOSS_NET_MGR_COLLECT_B	CODE	192898 - 192A98	0000
IOP_220	IOSS_UTLS_B	CODE	100FA4 - 101043	0000
IOP_221	IOSS_GPC_DATA_B	CODE	101300-10138B	0000
IOP_222	IOSS_NET_STATUS_B	CODE	11F534 - 11F734	0000

Table 4-1. Software Fault Injection Plan

4.3.2 The Hardware Fault Injection Plan

The Hardware Fault Injection Plan involved applying faults to links in the data exchange network and the fault-tolerant clock network. Overviews of these two networks are shown in Figures 4-2 and 4-3, respectively. Details about the data exchange and fault-tolerant clock may be obtained from [4] and [5] by the reader who is unfamiliar with their operation. The fault-tolerance specific hardware of the AIPS FTP was chosen as the target of hardware fault injection since that is the unique part of the FTP.

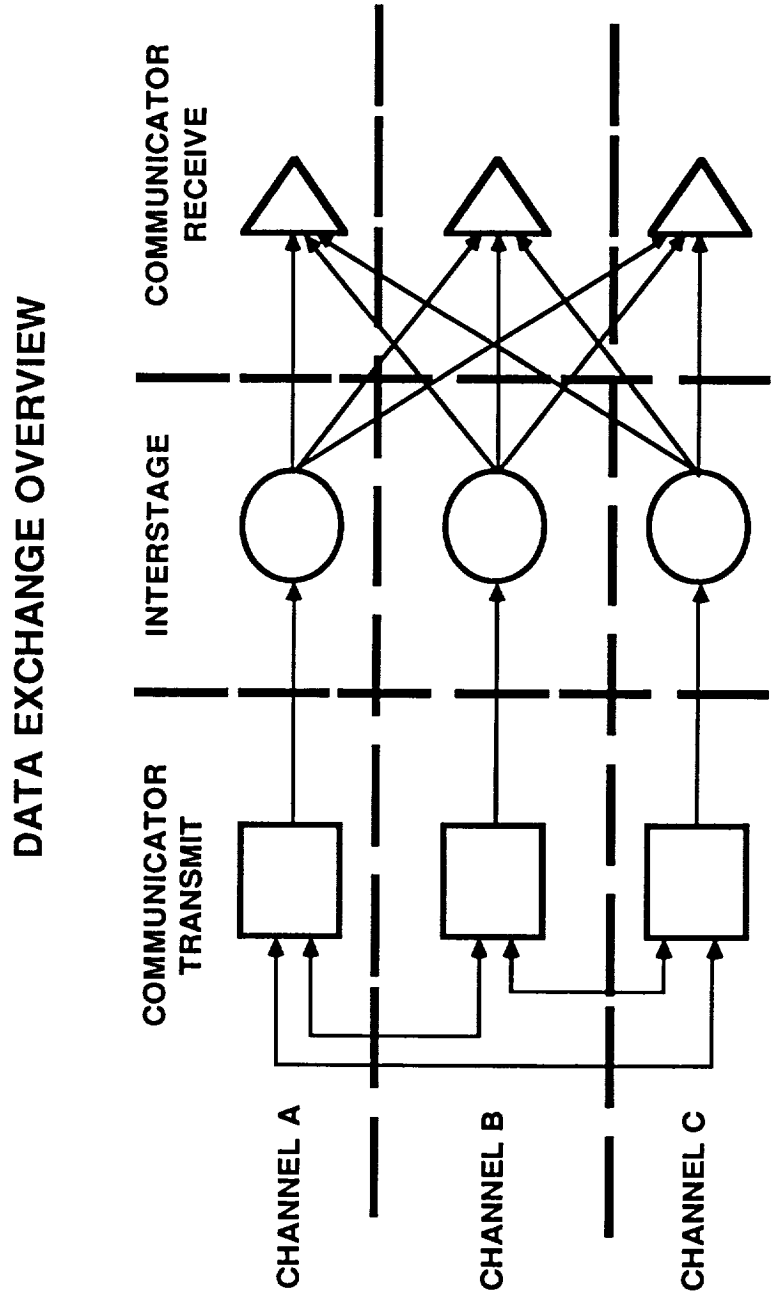

Figure 4-2. Data Exchange Network

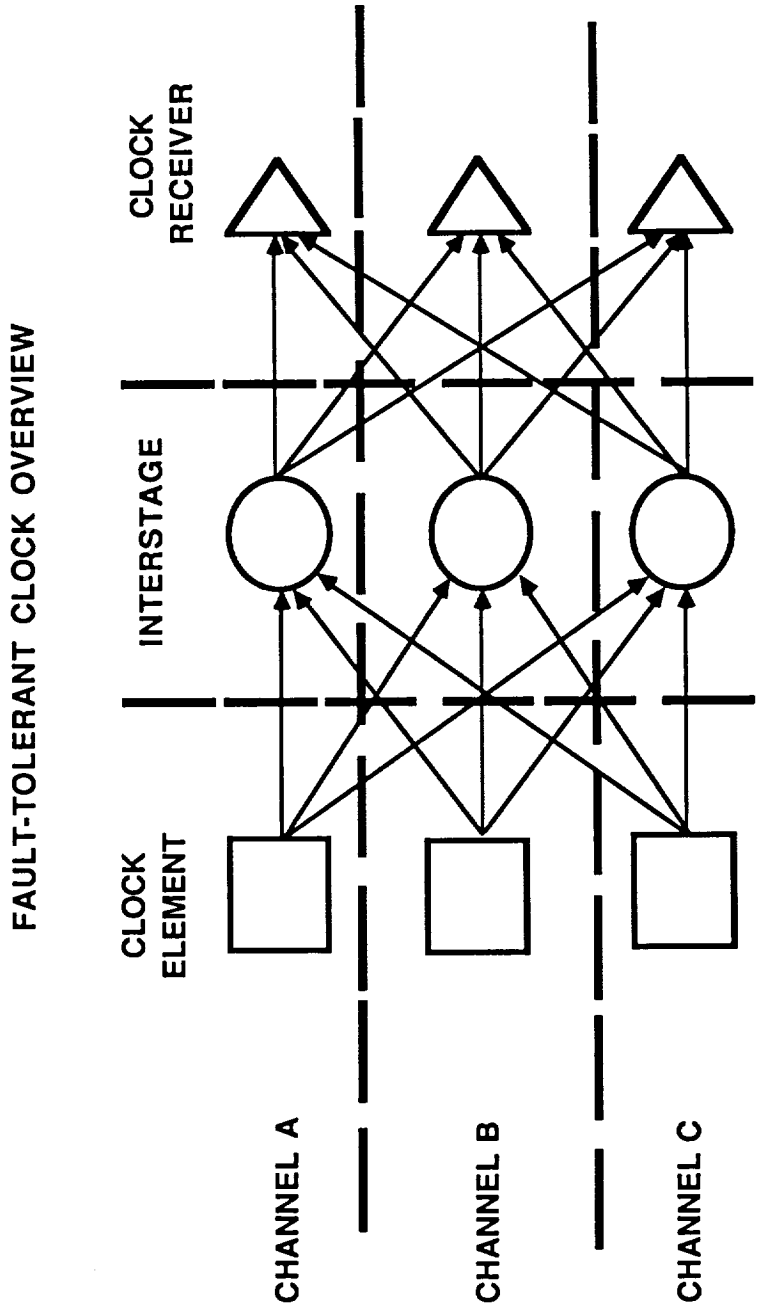

Figure 4.3. Fault Tolerant Clock Network

The Hardware Fault Injection Plan consisted of twelve tests, which were deterministically selected; that is, the particular pins whose signals were altered were chosen by the fault injector supervisor to produce a desired fault. Details of these tests are given in Table 4-2.

Test ID	Component To Be Faulted	IC Location	Pin #	Logic Level	Schematic Page
COM2_1356_6	DX: C-to-A transmitter link	1356	6	0	COM, Pg. 4
COM_1356_11	DX: C-to-B transmitter link	1356	11	0	COM Pg. 4
COM_1756_6	DX: B to C's Receiver	1756	6	0	COM, Pg. 5, 1&2
COM_0748_8_0	DX: C's Bypass Receiver	0748	8	0	COM Pg. 4, 1&2
COM_0748_8_1	DX: C's Bypass Receiver	0748	8	1	COM Pg. 4, 1&2
COM_0510_2_0	FTC: C's clock element to A's interstage	0510	2	0	COM Pg. 6
COM_0510_2_1	FTC: C's clock element to A's interstage	0510	2	1	COM Pg. 6
COM_0510_4_0	FTC: C's clock element to B's interstage	0510	4	0	COM Pg. 6
COM_0510_6_0	FTC:C's clock element to C's interstage	0510	6	0	COM Pg. 6
COM_0510_6_1	FTC: C's clock element to C's interstage	0510	6	1	COM Pg. 6
COM_0510_7_0	FTC: C's clock element to C's interstage	0510	7	0	COM Pg. 6
COM_0510_7_1	FTC: C's clock element to C's interstage	0510	7	1	COM Pg. 6

Table 4-2. Hardware Fault Injection Plan

4.4 Core FTP Fault Injection Test Results

4.4.1 Software Fault Injection Test Results

As discussed in Section 4.3.1, the Core FTP Software Fault Injection Plan is comprised of 178 tests. Each test is repeated 5 times. Each iteration of a test involves the application of a fault, its detection by the FDIR software, the appropriate FTP reconfiguration, and finally recovery of the faulty channel by the Lost Soul Sync process. For each iteration, the Fault Injector Software recorded the fault detection and reconfiguration times. It also verified that the channel had been recovered before injecting the fault again.

The FTP Software Fault Injection results are presented in two sections. First, Section 4.4.1.1 presents the maximum and average times for each test. Next, Section 4.4.1.2 illustrates and discusses the Software Fault Injection probability and cumulative density functions.

4.4.1.1 Maximum and Average Times

Table 4-3 provides the maximum and average detection and reconfiguration times for each test. As explained in Section 2.1, detection time is defined as the time elapsed between insertion of the fault and detection of the fault by the FDIR process and reconfiguration time is defined as the time elapsed between detection of the fault and removal of the failed channel from the active configuration. The identifiers in the "Test Numbers" column in Table 4-3 correspond to those given in Table 4-1. The "Test Results" column indicates whether or not a fault was detected. If the fault was detected, the process by which it was found is given. In some cases the fault was detected, but the faulty channel could not be recovered, i.e., it could not be resynchronized. These cases are so noted and are explained in following sections. In these cases, however, the FTP as a whole did properly sustain the fault and continue to operate as a duplex.

Test No.	Test Results	Max Detect (ms)	Max Reconfig (ms)	Average Detect (ms)	Average Reconfig (ms)
CP_1	Fault Detected;Chan Not Recovered			191.2	1.7
CP_2	Detected: RAM Scrub	95536.5	31.8	90936.7	19.3
CP_3	Detected: RAM Scrub	95273.0	39.4	81129.8	24.0
CP_4	Detected: Presence Test	29.8	1.9	16.7	1.8
CP_5	Detected: Presence Test	19.9	1.8	12.2	1.8
CP_6	Detected: RAM Scrub	96927.8	27.0	90608.0	22.8
CP_7	Detected: RAM Scrub	95594.2	35.4	93376.7	24.8
CP_8	Detected: Presence Test	58.7	1.8	39.7	1.8
CP_9	Detected: RAM Scrub	96584.0	34.5	89280.0	18.4
CP_10	Detected: RAM Scrub	96250.3	126.4	85340.9	47.8
CP_11	Detected: RAM Scrub	106816.6	38.4	95036.5	33.1
CP_12	Detected: RAM Scrub	146641.1	127.1	119054.9	43.6
CP_13	Detected: Presence Test	510.2	1.7	185.7	1.7
CP_14	Detected: Presence Test	28.7	1.7	17.2	1.7
CP_15	Detected: RAM Scrub	96583.7	27.9	77436.6	20.3
CP_16	Fault Detected;Chan Not Recovered			12.8	1.8
CP_17	Fault Detected;Chan Not Recovered			58130.1	28.4
CP_18	Fault Detected;Chan Not Recovered			8.7	1.7
CP_19	Detected: Presence Test	358.2	1.8	201.6	1.8
CP_20	Detected: RAM Scrub	146297.6	35.9	118256.2	22.9
CP_21	Detected: RAM Scrub	96081.5	44.7	81916.6	30.7
CP_22	Fault Detected;Chan Not Recovered			98451.3	35.6
CP_23	Detected: RAM Scrub	96156.9	12.0	53652.6	3.8
CP_24	Detected: Presence Test	61.1	1.8	32.0	1.8
CP_25	Detected: RAM Scrub	96855.2	87.5	83947.3	38.6
CP_26	Detected: Presence Test	53.8	1.9	28.1	1.8
CP_27	Detected: RAM Scrub	96084.1	37.0	90501.5	22.3
CP_28	Detected: RAM Scrub	97651.6	88.0	93740.5	32.4

Test No.	Test Results	Max Detect (ms)	Max Reconfig (ms)	Average Detect (ms)	Average Reconfig (ms)
CP_29	Detected: Presence Test	32.6	1.8	24.4	1.8
CP_30	Detected: RAM Scrub	147435.0	124.4	140214.0	37.8
CP_31	Detected: RAM Scrub	146690.3	34.9	124763.0	23.1
CP_32	Detected: Presence Test	181.2	1.8	120.9	1.8
CP_33	Detected: RAM Scrub	145787.1	124.2	141029.4	53.3
CP_34	Detected: RAM Scrub	145449.8	171.3	141481.8	56.5
CP_35	Detected: Presence Test	2559.1	1.8	2154.2	1.8
CP_36	Detected: Unknown DX, (2) RAM Scrub, (2) Presence Test	93983.9	16.3	26167.1	6.7
CP_37	Detected: RAM Scrub	96451.0	32.6	95110.9	23.5
CP_38	Detected: Presence Test	419.6	1.8	257.3	1.8
CP_39	Detected: RAM Scrub	145522.8	31.3	131949.5	18.8
CP_40	Detected: Unknown DX, (4) Presence Test	866.8	1.8	462.2	1.8
CP_41	Detected: RAM Scrub	96339.4	36.6	87893.3	22.6
CP_42	Detected: RAM Scrub	96473.5	31.6	93824.6	20.3
CP_43	Detected: RAM Scrub	153216.9	34.0	146463.7	22.8
CP_44	Detected: Presence Test	226.2	1.9	101.2	1.8
CP_45	Detected: RAM Scrub	96657.1	38.7	89099.0	27.7
CP_46	Detected: Presence Test	207.5	1.8	121.9	1.8
CP_47	Detected: (4) RAM Scrub, Presence Test	98961.1	86.6	66234.2	31.7
CP_100	Detected: PROM Sum	149025.6	34.2	126143.8	23.1
CP_101	Detected: PROM Sum	146475.7	126.0	139417.8	59.2
CP_102	Detected: PROM Sum	96552.2	86.5	79653.5	32.6
CP_103	Detected: PROM Sum	95355.4	39.8	88859.0	22.4
CP_104	Detected: PROM Sum	105570.4	85.8	94836.4	36.6
CP_105	Detected: PROM Sum	147230.7	126.3	128520.9	56.4
CP_106	Detected: PROM Sum	146347.9	165.6	138544.8	49.0
CP_107	Detected: PROM Sum	145507.7	36.7	140070.1	25.4

Test No.	Test Results	Max Detect (ms)	Max Reconfig (ms)	Average Detect (ms)	Average Reconfig (ms)
CP_108	Detected: PROM Sum	146181.2	87.0	138558.3	33.7
CP_109	Detected: Presence Test	37.4	1.7	24.0	1.7
CP_110	Detected: PROM Sum	144039.0	207.1	119955.2	65.2
CP_111	Detected: PROM Sum	146472.8	127.0	139991.4	44.6
CP_112	Detected: PROM Sum	95114.8	86.4	78788.5	36.2
CP_113	Detected: PROM Sum	95282.8	126.7	88119.7	57.8
CP_114	Detected: PROM Sum	91444.8	34.7	81038.1	23.0
CP_115	Detected: PROM Sum	103959.8	29.0	93040.1	20.5
CP_116	Detected: PROM Sum	149127.6	47.3	140330.2	28.9
CP_117	Detected: PROM Sum	112119.6	47.6	94477.6	25.1
CP_118	Detected: PROM Sum	95999.6	39.6	79645.4	24.2
CP_119	Detected: PROM Sum	96148.4	33.9	86219.3	22.8
CP_120	Detected: PROM Sum	146386.3	32.6	130109.0	27.0
CP_121	Detected: PROM Sum	95199.9	33.7	92444.6	22.1
CP_122	Detected: PROM Sum	91539.1	32.3	84941.6	28.9
CP_123	Detected: PROM Sum	101278.8	90.1	93879.4	36.5
CP_124	Detected: PROM Sum	214498.1	34.5	136828.9	28.1
CP_125	Detected: PROM Sum	148181.3	85.6	143241.3	33.3
CP_126	Detected: PROM Sum	150162.1	126.2	144700.0	38.9
CP_127	Detected: PROM Sum	95944.0	34.1	93968.3	27.2
CP_128	Fault Detected;Chan Not Recovered			44058.4	12.8
CP_129	Detected: PROM Sum	111365.6	29.3	96410.6	23.4
CP_130	Detected: PROM Sum	95094.7	39.1	89605.8	33.6
CP_131	Detected: PROM Sum	92340.8	38.8	86177.6	29.7
CP_132	Detected: PROM Sum	95953.4	33.2	88120.0	18.6
CP_133	Detected: PROM Sum	146889.4	33.6	134713.3	26.2
CP_134	Detected: PROM Sum	144691.0	89.1	142409.8	30.9
CP_135	Detected: PROM Sum	146346.8	26.9	143411.4	19.6
CP_136	Detected: PROM Sum	147351.0	89.5	142848.3	39.4

Test No.	Test Results	Max Detect (ms)	Max Reconfig (ms)	Average Detect (ms)	Average Reconfig (ms)
CP_137	Detected: PROM Sum	148606.3	38.9	96564.1	28.3
CP_138	Detected: PROM Sum	96890.8	37.2	93503.4	24.0
CP_139	Detected: PROM Sum	96786.7	48.2	93405.1	27.5
CP_140	Detected: PROM Sum	145792.4	31.6	107682.4	21.6
CP_141	Detected: PROM Sum	94577.3	89.3	88625.5	39.9
CP_142	Detected: Presence Test	644.0	1.8	437.0	1.8
CP_143	Detected: PROM Sum	96056.9	20.0	82852.3	16.9
CP_200	Detected: Presence Test	168677.5	86.5	106666.7	42.9
CP_201	Detected: Presence Test	28.1	1.7	21.5	1.7
CP_202	Fault Detected;Chan Not Recovered			36.1	1.7
CP_203	Detected: PROM Sum	95505.6	34.0	80481.9	30.6
CP_204	Detected: Presence Test	51.5	1.8	34.8	1.8
CP_205	Detected: Presence Test	23.9	1.7	14.0	1.7
CP_206	Detected: Presence Test	477.6	1.8	265.4	1.8
CP_207	Detected: Presence	37.9	1.8	26.5	1.8
CP_208	Detected: PROM Sum	155942.4	85.2	95294.7	31.6
CP_209	Fault Detected;Chan Not Recovered			60.1	1.8
CP_210	Detected: Presence Test	87.7	1.8	46.6	1.8
CP_211	Detected: Presence Test	61.5	1.8	36.3	1.8
CP_212	Detected: PROM Sum	191510.9	127.6	145278.9	43.8
CP_213	Fault Detected;Chan Not Recovered			46.6	1.8
CP_214	Detected: Presence Test	40.0	1.8	31.2	1.8
CP_215	Detected: PROM Sum	195341.8	126.7	178607.7	46.7
CP_216	Detected: PROM Sum	192959.1	38.6	189657.3	20.6
CP_217	Detected: Presence Test	25.1	1.7	15.2	1.7
CP_218	Detected: Presence Test	211.7	1.8	113.3	1.8
CP_219	Detected: Presence Test	446.5	1.8	244.4	1.8

Test No.	Test Results	Max Detect (ms)	Max Reconfig (ms)	Average Detect (ms)	Average Reconfig (ms)
CP_220	Detected: Presence Test	431.7	1.8	275.2	1.8
CP_221	Detected: Presence Test	178.7	1.8	134.6	1.8
CP_222	Detected: Presence Test	198.3	1.8	129.2	1.8
CP_223	Detected: Presence Test	203.5	1.8	127.7	1.8
CP_224	Detected: Presence Test	54.5	1.8	39.7	1.8
CP_225	Detected: Presence Test	70.3	1.8	53.7	1.8
CP_226	Detected: PROM Sum	81303.0	71.7	65041.4	33.4
CP_227	Detected: PROM Sum	95554.4	36.2	76895.2	26.6

Table 4-3. Software Fault Injection Results (continued)

Test No.	Test Results	Max Detect (ms)	Max Reconfig (ms)	Average Detect (ms)	Average Reconfig (ms)
IOP_1	Detected: Presence Test	51.7	1.8	17.3	1.8
IOP_2	Detected: Presence Test	213.5	1.8	73.8	1.8
IOP_3	Detected: Presence Test	82.4	1.8	44.1	1.8
IOP_4	Detected: RAM Scrub	81359.2	28.6	72260.6	22.3
IOP_5	Detected: RAM Scrub	123713.3	207.0	111310.0	59.4
IOP_6	Detected: Presence Test	4351.0	1.8	2915.1	1.8
IOP_7	Detected: Presence Test	453.9	1.8	210.7	1.8
IOP_8	Detected: RAM Scrub	124721.8	30.8	107873.3	22.8
IOP_9	Detected: RAM Scrub	125274.2	126.3	107683.2	38.2
IOP_10	Detected: RAM Scrub	122840.8	34.4	114341.6	24.4
IOP_11	Detected: RAM Scrub	123551.2	34.1	119467.1	23.7
IOP_12	Detected: (2) RAM Scrub, (3) Presence Test	103016.8	206.0	62119.4	47.3
IOP_13	Detected: RAM Scrub	97933.4	35.6	60132.0	22.9
IOP_14	Detected: RAM Scrub	124456.5	37.5	112652.0	26.3
IOP_15	Detected: Presence Test	81504.3	86.4	77742.9	33.0
IOP_16	Detected: RAM Scrub	43.8	1.8	24.6	1.8
IOP_17	Detected: (4) Presence Test, RAM Scrub	34048.2	27.8	14011.5	7.0
IOP_18	Detected: Presence Test	451.4	1.9	260.3	1.8
IOP_19	Detected: RAM Scrub	80649.0	30.4	64473.7	21.5
IOP_20	Detected: RAM Scrub	82006.5	21.1	78393.8	16.5
IOP_101	Detected: PROM Sum	112417.8	126.9	84467.5	42.6
IOP_102	Detected: PROM Sum	81264.9	86.6	70073.0	34.9
IOP_103	Detected: PROM Sum	81583.4	28.9	67491.2	22.3
IOP_104	Detected: PROM Sum	124832.0	30.1	99660.3	22.3
IOP_105	Detected: PROM Sum	124163.1	38.2	90309.8	33.8
IOP_106	Detected: PROM Sum	81008.4	25.9	66098.0	20.4
IOP_107	Detected: PROM Sum	81412.8	34.7	77279.2	25.5
IOP_108	Detected: PROM Sum	81336.4	47.8	79185.6	29.3

Test No.	Test Results	Max Detect (ms)	Max Reconfig (ms)	Average Detect (ms)	Average Reconfig (ms)
IOP_109	Detected: PROM Sum	81871.0	126.2	77719.7	39.8
IOP_110	Detected: PROM Sum	81761.3	39.6	78144.3	20.4
IOP_111	Detected: PROM Sum	125472.4	129.0	101481.2	48.7
IOP_112	Detected: PROM Sum	123597.6	32.2	120999.4	18.8
IOP_113	Detected: PROM Sum	76507.7	40.1	52846.2	30.8
IOP_114	Detected: PROM Sum	87151.9	35.0	58393.2	20.8
IOP_115	Detected: PROM Sum	79690.6	88.8	51821.7	41.8
IOP_116	Detected: PROM Sum	71076.4	35.9	48508.4	26.8
IOP_117	Detected: PROM Sum	81476.4	37.4	77877.6	23.0
IOP_118	Detected: PROM Sum	124452.0	32.2	86398.3	23.2
IOP_119	Detected: PROM Sum	94316.9	126.0	71361.5	40.2
IOP_201	Detected: PROM Sum	80541.2	37.1	60870.8	18.3
IOP_202	Fault Detected;Chan Not Recovered			34.7	1.7
IOP_203	Detected: (4) PROM Sum, Presence Test	81364.0	35.2	67697.8	19.0
IOP_204	Detected: PROM Sum	122004.9	20.3	114485.8	14.4
IOP_205	Detected: PROM Sum	81567.6	28.1	72695.6	16.9
IOP_206	Detected: PROM Sum	82127.9	39.0	81483.0	21.0
IOP_207	Fault Detected;Chan Not Recovered			20.3	1.8
IOP_208	Detected: Presence Test	47.7	1.8	31.0	1.8
IOP_209	Detected: Presence Test	146.6	1.8	63.0	1.8
IOP_210	Detected: PROM Sum	125624.5	26.7	122849.6	19.3
IOP_211	Detected: PROM Sum	138244.3	127.2	123907.0	48.1
IOP_212	Detected: PROM Sum	124774.5	32.4	120438.3	21.2
IOP_213	Detected: PROM Sum	124400.9	32.0	105208.4	23.9
IOP_214	Detected: PROM Sum	81767.7	29.7	78645.9	21.1
IOP_215	Detected: Presence Test	419.6	1.8	269.9	1.8
IOP_216	Detected: Presence Test	377.6	1.8	144.6	1.8
IOP_217	Detected: PROM Sum	84940.3	38.8	82048.9	35.0

Test No.	Test Results	Max Detect (ms)	Max Reconfig (ms)	Average Detect (ms)	Average Reconfig (ms)
IOP_218	Detected: PROM Sum	124793.0	35.0	100550.4	23.0
IOP_219	Detected: PROM Sum	126107.3	38.5	121129.2	25.5
IOP_220	Detected: PROM Sum	123171.1	88.4	116307.4	39.7
IOP_221	Detected: PROM Sum	123045.8	20.9	107199.0	16.6
IOP_222	Detected; PROM Sum	122310.4	33.2	106561.6	23.1

Table 4-3. Software Fault Injection Results (concluded)

Detection Times

Detection times vary because for each test the corrupted area of memory is referenced with a different frequency. If the corrupted area is commonly used, the fault will most likely be manifested by the next iteration of Fast FDIR. If the corrupted area is infrequently referenced, it may not cause a problem until several iterations of Fast have occurred. If the area is never referenced, the fault will only be detected by the Prom Sum or Ram Scrub tests in the Background Selftest task, which could take several minutes.

None of the detection times were unexpected. All faults detected by Fast FDIR (Presence Test) were detected within 40 ms after being injected. Faults detected by the Background Selftests (RAM Scrub, PROM Sum) took a maximum of 214 seconds (approximately 3-1/2 minutes). This is consistent with the amount of time it takes to do a complete iteration of the Background Selftests, which is about 4 minutes. The wide range of detection times for faults detected by the Background Selftests is a result of the position of the tests at the time when the memory was corrupted. If the corrupted locations had just been examined by the selftests, the fault would take much longer to detect than if the corrupted locations were just about to be examined.

Reconfiguration Times

Reconfiguration times vary depending on the process that detected the fault. When a fault is detected by Fast FDIR, the reconfiguration takes place immediately (less than 2 ms). Faults detected by the Background Selftests (RAM Scrub, PROM Sum) can be expected to have a wide range of reconfiguration times, which is explained by the fact that the Background Selftests does not do the reconfiguration itself, but passes the information to Fast FDIR to act upon. If Fast FDIR has just finished prior to the Selftests detecting a fault, the reconfiguration will not take place for at least 40 ms. Additional delays can occur because the Selftests may be interrupted by a higher priority task between the time that it detects the fault and the time that it has finished creating the reconfiguration information for Fast and is ready for Fast to act upon it.

None of the reconfiguration times were unexpected. All faults detected by Fast FDIR (Presence Test) were reconfigured immediately (less than 2 ms) after being detected. The highest reconfiguration times for Selftest-detected faults were 160-170 ms (only 2 cases), which indicates that the Selftests were suspended for four iterations of Fast. This is not unreasonable, considering the number of tasks operating in the system.

Faults Detected but Channel Not Recovered

In several cases, the fault was detected but the channel was not recovered, i.e., it could not be resynchronized. These situations are discussed below.

- *CP_1*: The corrupted memory in this case caused the faulty processor to get into an infinite loop with interrupts off. The other channels saw the faulty channel as failing the presence test and disengaged its Monitor Interlock. This prevented the faulty processor's Watchdog Timers from going off and getting the processor out of its infinite loop.

- *CP_16*: This test wipes out a section of memory used in doing the CP-IOP handshake before a lone channel attempts to be picked up. The CP is in an infinite loop because it has written the handshake word into the wrong location and is now waiting for the IOP to respond in that location. The fault has crippled the channel to such an extent that it cannot even attempt to be picked up.

- *CP_17*: This test wipes out a section of memory used by the Lost Soul Sync task when a lone channel attempts to be picked up by good channels. The lone channel cannot function well enough to even attempt to be picked up.

- *CP_18*: This test wipes out a section of memory used in doing the CP-IOP handshake before a lone channel attempts to be picked up. The CP is in an infinite loop attempting to complete the handshake. The fault has crippled the channel to such an extent that it cannot even attempt to be picked up.

- *CP_128*: This test wipes out a section of memory used in doing the CP-IOP handshake before a lone channel attempts to be picked up. The CP is in an infinite loop attempting to complete the handshake. The fault has crippled the channel to such an extent that it cannot even attempt to be picked up.

- *CP_202*: This test wipes out a section of code used by the Lost Soul Sync task when a lone channel attempts to be picked up by good channels. The CP in the lone channel is getting repeated hardware exceptions. The fault has crippled the channel to such an extent that it cannot even attempt to be picked up.

- *CP_209*: This test wipes out a section of memory used in doing the CP-IOP handshake before a lone channel attempts to be picked up. The CP is in an

infinite loop attempting to complete the handshake. The fault has crippled the channel to such an extent that it cannot even attempt to be picked up.

- *CP_213*: This test wipes out a section of memory used in doing the CP-IOP handshake before a lone channel attempts to be picked up. The CP is in an infinite loop attempting to complete the handshake. The fault has crippled the channel to such an extent that it cannot even attempt to be picked up.

- *IOP_207*: This test wipes out a section of memory used in doing the CP-IOP handshake before a lone channel attempts to be picked up. The CP is in an infinite loop attempting to complete the handshake. The fault has crippled the channel to such an extent that it cannot even attempt to be picked up.

4.4.1.2 Probability and Cumulative Density Functions

The Core FTP Fault Insertion Plan is comprised of 178 tests. Each test consists of five iterations. For each iteration, the fault detection and reconfiguration times are recorded by the Fault Insertion Software. Accordingly, a total of 890 data sets were anticipated. However, as described in Section 4.4.1.1.4, in ten tests the fault was detected but the faulty channel could not be recovered (implying that only one data set was recorded per test rather than five). As a result, 850 sets of data, or 95.5 percent of the applied faults, were posted.

Probability and cumulative density functions for these data sets were generated to complete the Core FTP Fault Insertion Analysis; these functions are illustrated in Figures 4-4 through 4-9.

The probability density function for the FTP fault detection times, shown in Figure 4-4, depicts the wide variance in the observed times. As discussed in Section 4.4.2.1, this variance occurred because some faults were detected by the high priority Fast FDIR task while other faults were detected by the low priority background Selftest process.

Several distinguishable peaks can be observed (illustrated in Figures 4-4 and 4-5):

- *0 - 40 ms.* Approximately 25 percent of the faults were detected in this range.
- *78,000 - 82,000 ms.* Ranges from a minimum of 0 to a maximum of 2 percent of the faults.
- *92,000 - 97,000 ms.* Ranges from a minimum of 0 to a maximum of 2 percent of the faults.
- *120,000 - 125,000 ms.* Ranges from a minimum of 0 to a maximum of 1 percent of the faults.

- *140,000 - 147,000 ms.* Ranges from a minimum of 0 to a maximum of 1 percent of the faults.

As shown by the cumulative density function for the detection times (Figure 4-6), approximately 25 percent of the software injected faults were detected in 40 milliseconds or less, 50 percent in 80 seconds or less, and 75 percent in 100 seconds or less.

The probability density function for the Core FTP fault reconfiguration times (Figures 4-7 and 4-8) indicates that a significant percentage of the faults were bypassed in 0 to 220 ms. while a substantial but smaller percentage were reconfigured around in 220 to 1950 ms. This variance occurred because some faults were detected by the high priority Fast FDIR task and therefore reconfigured immediately, while other faults that were detected by the low priority Selftest task took longer to reconfigure.

As depicted by the cumulative density function in Figure 4-9, approximately 94 percent of all the simulated memory faults were isolated and bypassed in 2000 ms. or less.

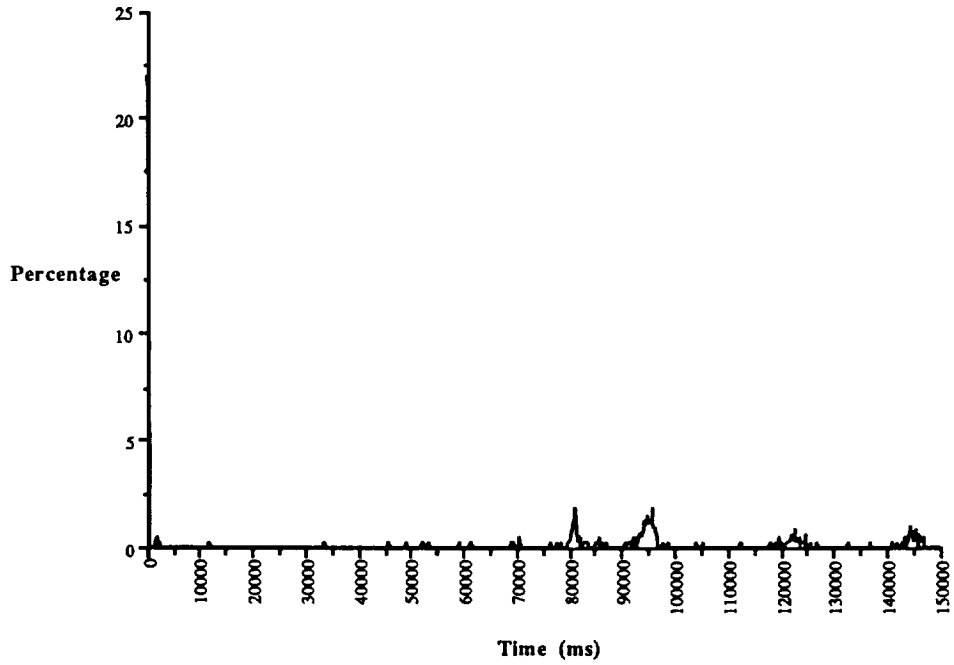

Figure 4-4. The Probability Density Function for the Detection Times

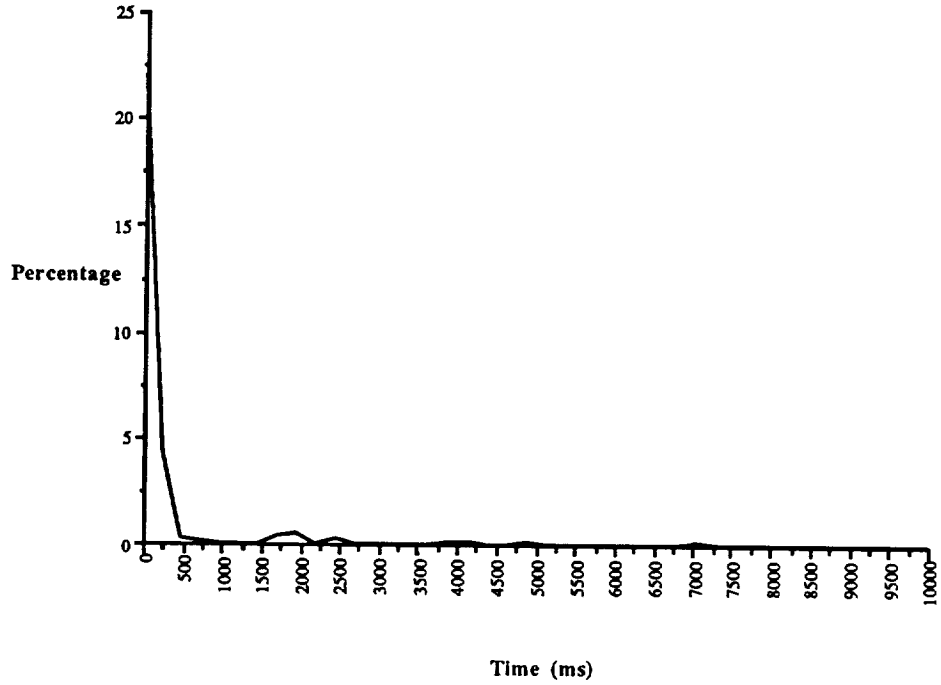

Figure 4-5. The Probability Density Function for the Detection Times: Expansion of the 0 to 10,000 ms. Region

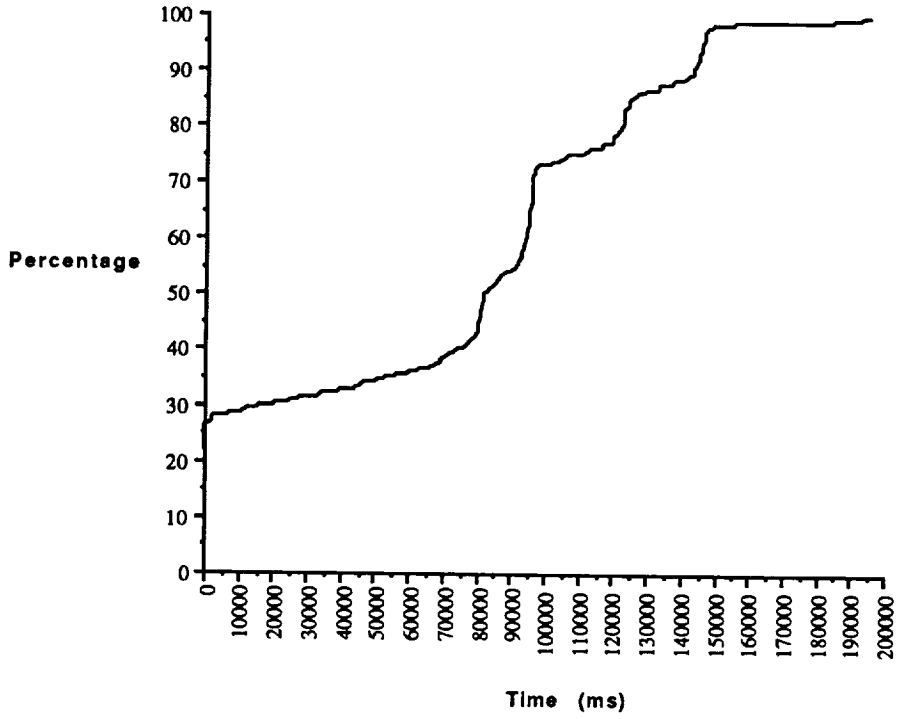

Figure 4-6. The Cumulative Density Function for the Detection Times

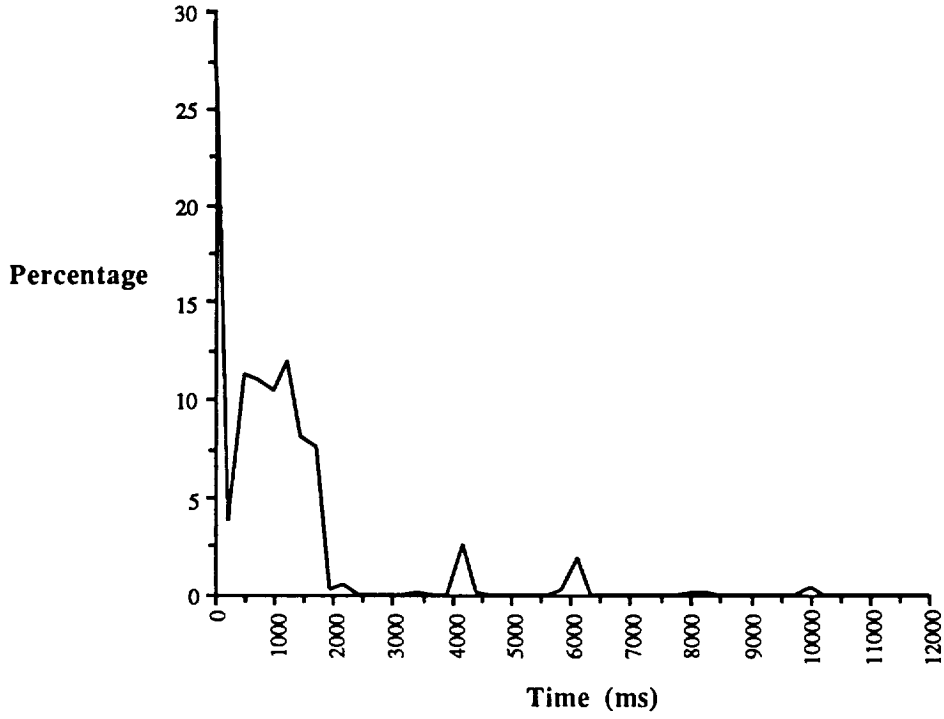

Figure 4-7. The Probability Density Function for the Reconfiguration Times

Figure 4-8. The Probability Density Function for the Reconfiguration Times: Expansion of Range 0 to 2400 ms.

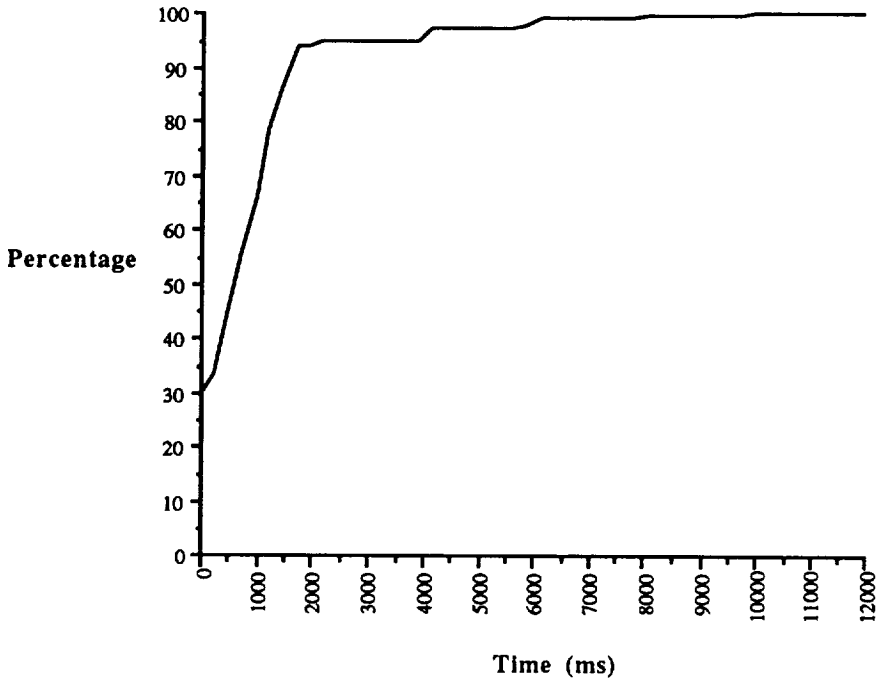

Figure 4-9. The Cumulative Density Function for the Reconfiguration Times

4.4.2 Hardware Fault Injection Test Results

As discussed in Section 4.3.2, the Hardware Fault Injection Plan consists of 12 tests. Each test is repeated 10 times. Each iteration of a test involves the application of a fault, its detection by the FDIR software, the associated FTP reconfiguration, and finally recovery of the faulty channel. For each iteration, the Fault Insertion Software recorded the fault detection and reconfiguration times. It also verified that the channel had been recovered before injecting the fault again.

The test results are shown in Table 4-4. In the first three tests, the fault was detected and correctly identified and the system was reconfigured. In the next two tests, the fault was detected and identified, but the channel could not be recovered. This is hypothesized to be due to the fact that the injected fault prevented the faulty channel from being able to be resynchronized. In the final seven tests, the fault was not detected. This was found to be due to a bug in the FDIR software, which only looks at a voted value of the FTC error latches rather than looking at each channel's latch individually. Looking at the voted value recognizes faults in the entire clock element or entire interstage, but ignores individual link faults. The particular fault created in all seven test cases was the failure of a link between a clock element and an interstage. There was no opportunity to rerun the tests with a corrected version of the software. Since the number of test cases was small, no statistical analysis was done.

Test ID	Test Results	Max Detect (ms)	Max Reconf (ms)	Average Detect (ms)	Average Reconf (ms)
COM2_1356_6	Detected: A-to-C transmitter link failure	39.9	3.4	26.3	3.4
COM_1356_11	Detected: B-to-C transmitter link failure	39.9	3.5	28.4	3.5
COM_1756_6	Detected: A-to-C transmitter link failure	41.1	3.5	24.1	3.4
COM_0748_8_0	Detected: Presence; Channel Not Recovered			13.6	1.8
COM_0748_9_0	Detected: Presence; Channel Not Recovered			14.8	1.8
COM_0510_2_0	Not Detected				
COM_0510_2_1	Not Detected				
COM_0510_4_0	Not Detected				
COM_0510_6_0	Not Detected				
COM_0510_6_1	Not Detected				
COM_0510_7_0	Not Detected				
COM_0510_7_1	Not Detected				

Table 4-4. Hardware Fault Injection Results

4.4.3 Design Flaws Uncovered by the Fault Injection Tests

During execution of both the Hardware Fault Injection Plan and the Software Fault Injection Plan, several design flaws were uncovered. These design flaws included both hardware and software. All software design faults were corrected.

Hardware Design Flaws

1. Disengaging a channel's Monitor Interlock should not disable its Watchdog Timers. Test CP_1 showed that this can prevent a channel from being

recovered if the fault resulted in a processor being in an infinite loop with interrupts off.

Software Design Flaws

1. If the faulty channel went out of sync between the presence test and the data exchange latch analysis in Fast FDIR (refer to Section 4.2.1), a data exchange fault would be identified and possibly the wrong channel identified as faulty. This was corrected by exchanging data exchange latches and FTC latches before the presence test.

2. Occasionally the faulty channel would behave in such a way that it was out of sync only temporarily (e.g., during a data exchange) but was somehow forced back into sync and passed subsequent Program Counter checks and presence tests. However, the data exchange latches had been set by the other two channels, but not by the faulty channel. As in (1), this led to an erroneous diagnosis. The solution here was to verify that any reported DX fault was reported by all channels (except for a faulty interstage-to-receiver link, which is normally reported by only one channel).

3. A failed link in the FTC network (as opposed to an entire clock element or interstage) would not be detected by Fast FDIR because it looked only at a voted value of the FTC error latches, rather than looking at each channel's latches individually. This was corrected by examining each individual channel's latches.

4.5 Core FTP Fault Injection: Conclusions

To conclude the discussion of the Core FTP Fault Injection Plan, the Fault Injection Results are considered with respect to the goals of the Fault Injection Study. In brief, the objectives were:

1. to test the design specification for fault tolerance,
2. to obtain feedback for fault removal from the design implementation,
3. to obtain statistical data regarding fault detection, isolation, and reconfiguration responses, and
4. to obtain data regarding the effects of faults on system performance.

To test the system design specification for fault tolerance (Goal 1), we relied solely on visual observation of the CRT display to determine that the system functioned correctly during and after the fault. Since the display tasks execute at the lowest priority, this

ensured that no higher priority task was monopolizing the system as a result of the fault. In all test cases, the system functioned correctly.

To determine the correctness and completeness of the fault detection and identification (Goal 2) for each test case, two methods were used. One was visual inspection of the error log that is maintained by the core FTP FDIR process. This log indicated whether a particular fault was detected and isolated correctly The second method involved verification by the FIS that the fault was detected and that reconfiguration took place. As shown in Table 4-3, all of the test cases in the Software Fault Injection Plan were correctly detected and isolated. In a small percentage of the cases, the faulty channel could not be recovered because the memory that was corrupted was memory used by the recovery process, so that the fault appeared as a hard fault rather than a transient. In one case, the faulty channel could not be recovered because of a hardware design flaw that prevented the faulty channel from detecting that it was faulty. In addition, during the initial iterations of the Software Fault Injection Plan, errors in the FDIR software were detected which were corrected for subsequent iterations; only the final iteration was presented in this report.

As shown in Table 4-4, about half of the test cases in the Hardware Fault Injection Plan were correctly detected and isolated; the other half were not detected. The undetected faults were similar in that they all consisted of the failure of a link in the fault tolerant clock network. The FDIR software was determined to contain an error that prevented it from recognizing this type of fault. There was no opportunity to rerun the tests with corrected software.

Statistical data about the fault detection and identification (Goal 3) was obtained by having the FDIR software log the detection time in the Testport Interface and by having the Fault Injector Software note the time at which fault isolation occurred. As presented in Sections 4.4.1 and 4.4.2, the Core FTP Fault Injection Plan recorded 853 sets of data. The maximum and average times and the probability and cumulative distribution functions were calculated. The results of the Core FTP Fault Injection test cases conformed to the expected maximum and average times.

The effects of faults on system performance (Goal 4) include both (1) the additional time required by the particular FDIR process when it is dealing with a fault and the subsequent scheduling delays incurred by other tasks, and (2) the effects on users of the faulty component. The additional time required by the FDIR processes has been measured at other times during the life of the AIPS project and documented in a previous report [ ]. Measurement of the scheduling delays incurred by non-FDIR tasks and the effects of a fault on users of the faulty components require additional instrumentation for obtaining a record of system performance and was not in place at the time of this study.

5.0 CONCLUSIONS

This report has described a plan for systematically injecting large numbers of faults into the AIPS building blocks and collecting data about the resulting actions of the redundancy management processes. The goals of this fault injection plan were fourfold:

1. To test the system design specification for fault tolerance.
2. To obtain feedback for fault removal from the design implementation.
3. To obtain statistical data regarding fault detection, isolation, and reconfiguration responses.
4. To obtain data regarding the effects of faults on system performance.

A comprehensive set of possible fault injection tests was developed; from this a subset of actual test cases was selected. Both pin-level hardware faults using a hardware fault injector and software-injected memory mutations were used to test the system. A Fault Injection Software program was used to facilitate automatic fault injection and collect timing information.

Two of the AIPS building blocks, the I/O Network and the Core FTP, were chosen as candidates for extensive fault injection. The I/O Network Fault Injection Plan consisted of 47 different pin-level faults, inserted using the hardware fault injector. Each of these faults was applied 25 times. The detection coverage for these faults was 99.6%; the other 0.4% of the faults did not produce any detachable error symptoms. The reconfiguration coverage for detected faults was 100%. No design errors were found by these tests. This does not necessarily prove that the design of the network is completely error free, but it does increase the level of confidence in the design.

The anticipated maximum detection time for faults injected in the I/O Network was about 2075 ms. plus the error latency, and the expected average time was approximately 1040 ms. plus the error latency. The I/O Fault Injection results typically conformed to these expected maximum and average times. For the reconfiguration time, the worst case time, i.e., one including fault diagnostics plus the worst case regrowth scenario, was determined to be 3500 ms. All of the actual reconfiguration times were less than the worst case.

One problem was encountered when using the hardware fault injector to apply faults to the I/O Network. Sometimes when a fault injector probe was attached to an I/O node, the probe caused the node to fail. The fault injector supervisors speculated that this problem was caused by impedance differences that occurred when the probe was attached. Because of this problem, some of the originally proposed I/O Network faults were not injected.

The Core FTP Fault Injection Plan consisted of 178 simulated memory faults, inserted using the Fault Injection Software program, and 12 pin-level faults, inserted using the hardware fault injector. The memory faults were each applied 5 times; the pin-level faults were applied 10 times. Four design flaws were uncovered by the Core FTP Fault Injection Plan. Three of these were software design flaws; one was a hardware design flaw. As time permitted, the software design flaws were corrected and the test cases rerun. The hardware design flaw was not corrected.

The detection coverage for the simulated memory faults was 100%; the reconfiguration coverage was also 100%. The detection and reconfiguration times from each test typically conformed to the expected maximum and average times. However, in a number of the simulated memory fault test cases, the fault prevented the channel from being recovered. Therefore multiple instances of these particular faults could not be injected without restarting the system. The detection coverage for the pin-level fault test cases was 42%; the reconfiguration coverage was also 42%. The undetected faults were all of the same type and were the result of a software error. In two of the cases where the fault was detected, the faulty channel could not be recovered. The fault injection supervisors speculate that although the hardware fault injector stopped corrupting the pin after a certain time, the effect remained, thereby creating a permanent fault rather than a transient one.

A second problem encountered when using the hardware fault injector was that simply attaching the fault injector probe to a pin caused the channel to fail. The fault injector supervisors speculate that this was caused by the added distance between the chip and the card, which could result in either changed timing, added capacitance, or added noise. Because of this problem, many of the originally proposed Core FTP faults were not injected.

This study has demonstrated the importance of fault injection in the overall validation of a system. In addition to providing data for reliability parameter estimation, it can also provide feedback for fault removal from the design implementation. This does not mean that fault injection is a substitute for the design-for-validation methodology, but it *is* a component of the methodology just as specifications, design reviews, analytical models and formal methods are. If the fault injection process does not uncover a single flaw in the system under test, this does not imply that the system is perfect, only that the system is correct with respect to the fault set to which it was subjected. And if some design flaws *are* uncovered, the fault injection process provides a deeper understanding of the fault tolerance design and a more fundamental appreciation of the cascade of events triggered by a fault, including complex interactions between hardware and software elements. The Fault Injection tests performed as part of this study successfully met three of the four originally stated goals. They helped validate the system design specification for fault tolerance; they

detected faults in the design implementation; and they provided statistical data regarding fault detection, isolation, and reconfiguration responses.

Recommendations for future work in this area include gathering data on the effects of faults on system performance, in particular, any delays in execution of time-critical tasks. The envelope of test cases could also be extended to cover a fuller spectrum of the fault injection plan that has been described in this report. Another research area is to understand why certain transient faults behave as permanent faults, i.e., how the errors produced by a transient fault permanently disable a channel preventing its reintegration in the FTP.

6.0 REFERENCES

[1] Harper, R.E., Alger, L.S., and Lala, J.H., "Advanced Information Processing System: Design and Validation Knowledgebase", NASA Contractor Report 187544, September 1991.

[2] Johnson, S.C., and Butler, R.W., "Design for Validation," 10th AIAA/IEEE Digital Avionics Systems Conference, Los Angeles, CA, October 1991, PPs. 487-492.

[3] Lala, J.H., and Smith, T.B., III, "Development and Evaluation of a Fault Tolerant Multiprocessor (FTMP) Computer, Volume III, FTMP Test and Evaluation", NASA Contractor Report 166073, May 1983.

[4] Burkhardt, L.F., L. Alger, R. Whittredge, P. Stasiowski, "Advanced Information Processing System: Local System Services," NASA Contractor Report 181767, March 1989.

[5] Gauthier, R.J., "The Airlab Fault-Tolerant Processor: Physical Implementation," Charles Stark Draper Laboratory Report, CSDL-R-1928, December 1986.

[6] Lala, J.H., Harper, R.E., and Alger, L.S., "A Design Approach for Ultrareliable Real-Time Systems," IEEE Computer Special Issue on Real-Time Systems, May 1991.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Contractor Report

4. TITLE AND SUBTITLE Advanced Information Processing System: Fault Injection Study and Results

5. FUNDING NUMBERS
WU 506-59-61-03
C NAS1-18565

6. AUTHOR(S) Laura F. Burkhardt, Thomas K. Masotto, and Jaynarayan H. Lala

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Charles Stark Draper Laboratory, Inc.
Cambridge, MA 02139

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA CR-189590

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Felix L. Pitts
Final Report - Task 12

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified - Unlimited

Subject Category 62

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words) The objective of the Advanced Information Processing (AIPS) program is to achieve a validated fault-tolerant distributed computer system. The goals of the AIPS fault injection study were:

1. To present the fault injection study components addressing the AIPS validation objective
2. To obtain feedback for fault removal from the design implementation
3. To obtain statistical data regarding fault detection, isolation, and reconfiguration responses
4. To obtain data regarding the effects of faults on system performance

The organization of this report is as follows. Section 1 describes the parameters that must be varied to create a comprehensive set of fault injection tests, the subset of test cases selected for this study, the test case measurements the test case execution. Both pin-level hardware faults using a hardware fault injector and software-injected memory mutations were used to test the system. Section 2 provides an overview of the hardware fault injector and the associa software used to carry out the experiments. Sections 3 and 4 give detailed specifications of faults and test results for t Network and the AIPS Fault Tolerant Processor, respectively. Section 5 summarizes the results and gives conclusion the study.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Fault-tolerant computing, fault injection, distributed fault-tolerant computers empirical validation	168
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABST
Unclassified	Unclassified		