CONVEX Multibus HYPERchannel Controller (*dev4510*) Diagnostics Manual

Document No. 760-002630-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX Multibus HYPERchannel Controller (dev4510) Diagnostics Manual
Order No. DHW-238
First Edition

Printed in the United States of America

Revision Sheet

CONVEX Multibus HYPERchannel Controller (dev4510) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-002630-000	May 1991	First release. Contains the *dev4510* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.*

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

Appendixes

A Reporting Problems

List of Tables

List of Figures

Preface

Purpose and Intended Audience

This manual explains how to run the *dev4500* diagnostic, which verifies the functionality of an IKON 10077-NSC Hyperchannel controller board. This document is not a tutorial, but rather a reference for the users of the *dev4510* diagnostics, including field service and manufacturing test personnel, as well as the diagnostics sustaining staff. In addition, CONVEX customers can use this manual to execute the *dev4510* diagnostic.

Scope

This manual applies to all CONVEX computers.

Organization

This document consists of the following:

- **Chapter 1. Diagnostics Environment**—Introduces the theories and concepts that underlie I/O diagnostics on CONVEX machines as well as the basic overview, philosophy, and structure of I/O diagnostics.
- **Chapter 2. EGOS Overview**—Provides a brief overview of the Event Governed Operating System (EGOS) and how it relates to device and peripheral diagnostics testing.
- **Chapter 3. Dshell Overview**—Provides a brief overview of and a general introduction to the *dshell* utility.
- **Chapter 4. Multibus HYPERchannel Controller Test (*dev4510*)**—Describes how to operate the diagnostic, including prerequisites, test invocation, hardware initialization sequence, and class descriptions. It also describes error messages produced by the diagnostic.
- **Appendix A. Reporting Problems**—Provides an example of the CONVEX *contact* utility for reporting minor software and hardware problems.

Notational Conventions

The notational conventions used in this text are listed below:

- Bit numbering is left to right, `N-1` through `0`. The most significant numerical bit is `N-1`, the least significant `0`. The bit numbering represents the binary weight of a position.
- Bit fields are specified using the following convention: *name*<*x..y*> where the bit field is *name* from bits *x* through *y*.
- Individual bit positions within a register are denoted by specific positions separated by commas. For example, REG<15,4,0> denotes bits 15, 4, and 0 of REG.
- Byte numbering is from left to right
- A *bit* is a single binary value or entity
- A *nibble* is 4 bits
- A *byte* is 8 bits
- A *halfword* is 16 bits
- A *word* is 32 bits
- A *longword* is 64 bits
- *Single precision* is a 32-bit floating point word
- *Double precision* is a 64-bit floating point longword
- An *instruction* is a multihalfword operand
- A bit is *set* when it contains a binary value of `1`.
- A bit is *clear* when it contains a binary value of `0`.
- All memory and I/O addresses are written in hexadecimal notation unless explicitly stated otherwise.
- All register contents are written in hexadecimal notation unless explicitly stated otherwise.
- A *register* is a programmer-visible hardware storage element internal to the processor
- *Physical memory* is the physical storage installed in the processor
- *Virtual memory* is the perceived amount of physical memory as seen by the application programmer
- The symbol *K* is an abbreviation for *kilo* or 1,024
- The symbol *M* is an abbreviation for *mega* or 1,048,576
- The symbol *G* is an abbreviation for *giga* or 1,073,741,824
- A *stack* is a linked-list group of words useful for dynamic allocation and deallocation of memory
- A *return block* is a collection of registers that is pushed or popped from a context stack in response to an instruction or other event
- *Reserved* or *undefined* convey what to expect, if anything, from unused fields in registers, reserved memory, or reserved I/O space. Algorithm implementation based on the use of undefined or reserved fields is not recommended.

Warnings

The following are examples of warnings, cautions, and notes and their typical content as used in CONVEX documents:

WARNING

Warnings highlight procedures or information necessary to avoid injury to personnel. A warning immediately precedes the critical information and includes a description of the hazard.

CAUTION

Cautions highlight procedures or information necessary to avoid damage to equipment, loss of data, or invalid test results. A caution immediately precedes the critical information and includes a description of the possible damage.

NOTE

Notes highlight useful information that is supplemental in nature. A note may immediately precede or follow the information that is being highlighted.

Associated Documents

The following is a partial list of other manuals or books that may provide more detailed information on the topics presented in this manual:

- *CONVEX Processor Diagnostics Manual (C1, C120)*, Order No. DHW-071
- *CONVEX Processor Diagnostics Manual (C200 Series)*, Order No. DHW-081
- *CONVEX Architecture Reference*, Order No. DHW-005
- *CONVEX SPU UNIX Utilities Manual*, Order No. DHW-021
- *CONVEX Processor Operation Guide (C100 Series, C200 Series)*, Order No. DHW-015
- *CONVEX Diagnostic Utilities Manual (C1, C120)*, Order No. DHW-072
- *CONVEX Diagnostic Utilities Manual (C200 Series)*, Order No. DHW-082
- *CONVEX UNIX Tutorial Papers*, Order No. DSW-002
- *The C Programming Language*, Kernighan & Ritchie, Order No. DSW-046

Ordering Documentation

To order the most current version of this or any other CONVEX document, use the product number. If the product number is not known, order by the exact title. In some situations, the most current version may not be desired. To receive a specific version of a manual, order the manual by its document number, or part number, which can be obtained by contacting the local CONVEX office or by calling the Technical Assistance Center.

The product number for this manual is DHW-238.
The document number for this manual is 760-002630-000.

CONVEX documents can be ordered by mail by sending a request to:

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851 USA

Technical Assistance

Hardware and software support can be obtained through the CONVEX Technical Assistance Center (TAC):

- From all locations in the continental United States, call 1(800)952-0379.
- From locations in Alaska, Hawaii, and Canada, call 1(214)497-4379.
- From all other locations, contact the nearest CONVEX office.

Reader's Forum

If you wish to mail your comments to us, please use the form at the end of this manual and list the document page number with your questions and comments. Thank you.

Chapter 1
Diagnostics Environment

1.1 Overview

CONVEX system diagnostics consist of a suite of test programs designed (except where noted) to execute under the Service Processor operating system, SPU UNIX. These programs utilize the capabilities of the Service Processor to test the operation of one or more of the functions of the system and report any errors detected. All of the diagnostics in this manual are intended to be executed "off-line"; that is, while CONVEX UNIX is not being executed by any of the Central Processing Units (CPUs) in the system.

The Service Processor, together with SPU UNIX, various diagnostic utilities, and the test programs, themselves, comprise the CONVEX diagnostic environment. This chapter describes the hardware and software components of this environment and is intended to provide the background necessary to fully utilize the capabilities of the CONVEX processor diagnostics.

For more information about the diagnostic environment refer to the Diagnostic Environment chapter in the *CONVEX Processor Diagnostics Manual (C200 Series)* or the *CONVEX Processor Diagnostics Manual (C1, C120)* depending on the architecture of the machine under test.

1.2 Test Program Naming Conventions

Test program names are in the form *cattypedevnn.suffix* where:

- *cat* is the subsystem being tested
- *type* is the type of test being performed, e.g., standalone, self-test, or offline functional test
- *dev* is the device being tested, e.g., disk, tape, or printer. This segment of the test program name is used *only* if the category is a device.
- *nn* is a CONVEX code used for distinguishing between test programs
- *suffix* is one of three program identifiers:
  - *.t* are programs that execute on SP2
  - *.x00* and *.rnn* are object files for different target processors other than the SP2. The target processor depends on the subject of the test. The test program name must have the test program category (*cat*) at the beginning of the name to determine the target processor.

1.2.1 Test Program Categories

Test program categories include those tests for the CPU, peripheral devices, I/O system, memory system, SP2, and entire system. For example, *cpu4041* is a CPU vector instruction test while *mem4000* is a memory system functional test. The following table lists test program categories:

Table 1–1, Test Program Categories

TEST PROGRAM CATEGORIES	
Test Category (*cat*)	**Description**
cpu	CPU subsystem related test
dev	Peripheral device test
io, idc, tli	I/O subsystem related test
mem	Memory subsystem related test
spu	SP2 subsystem related test

1.2.2 Test Program Types

A test program type describes whether a test is a standalone test, self-test, kernel hardware test, or an offline or online functional test. See the following table for the numbering system and description of test program types:

Table 1–2, Test Program Types

TEST PROGRAM TYPES	
Number (*type*)	**Description**
0	Standalone test
1	Self-test
2	Kernel hardware test
4, 5	Offline functional test

1.2.3 Test Program Device Types

Test programs will test disks, tapes, terminals, printers, and networks. See the following table for the numbering scheme and a description of the test program device types:

Table 1–3, Test Program Device Types

TEST PROGRAM DEVICE TYPES	
Number (*dev*)	**Description**
1	Disk
2	Tape
3	Terminal
4	Printer
5	Network

1.2.4 Examples of Test Program Names

The following table presents some examples using the naming conventions outlined above:

> **NOTE**
>
> In the following table, SOFF stands for Standard Object File Format.

Table 1–4, Example Test Program Names

EXAMPLE TEST PROGRAM NAMES	
Test Program Name	**Description**
cpu4041.t	SP2 object code in *b.out* format for *cpu4041*
cpu4041.rnn	C210 or C220 machine object code in SOFF format (relocatable)
cpu4041.x00	C210 or C220 machine object code in SOFF format (linked to run in segment 0)
mem4000.t	SP2 object code in *b.out* format for *mem4000*
mem4000.x00	C210 or C220 machine object code in SOFF format (linked to run in segment 0)
dev4100.t	SP2 object code in *b.out* format for *dev4100*
dev4100.x00	IOP object code in *b.out* format

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2
EGOS Overview

2.1 Overview

This chapter provides an overview of the Event Governed Operating System (EGOS) and how it relates to device and peripheral diagnostics testing. There are three basic types of EGOS systems, one for each type of CCU. There is one for the Multibus interface, one for the VME interface, and one for the HIA interface. This chapter will explain the three types of EGOS systems and how EGOS is positioned within the overall operating system environment.

2.2 Purpose of EGOS for Diagnostic Testing

EGOS is basically a simple operating system that the device tests use to handle interrupts, schedule processes, and generally allocate and control IOP/VIOP resources. The diagnostics code uses both EGOS and the Message Based System (MBS) to manipulate test program control over to the CCU side of the test program. MBS is not a part of EGOS but rather a system that allows a common section of memory to be used as a message area between multiple processors. For more information on MBS, refer to the *CONVEX Guide to Writing Device Drivers.*

EGOS initially sets up interrupt tables, determines how many chassis there are, and initializes its windows and resource allocation tables.

2.3 EGOS for the Multibus Interface

EGOS for the Multibus interface supports event driven device drivers. The Multibus version of EGOS takes interrupts that are local to a CCU and channels those errors to the proper piece of code to handle the error. It basically supplies the error interrupt handlers for the CCU error interrupts. It also contains support routines to control allocation of the various CCU-related resources.

2.4 EGOS for HSP Interface, HSP EGOS

EGOS for the HSP interface supports event driven device drivers. The HSP version of EGOS is like the Multibus version. It takes interrupts that are local to a CCU and channels those errors to the proper piece of code to handle the error. It basically supplies the error interrupt handlers for the CCU error interrupts. It also contains support routines to control allocation of the various CCU-related resources.

2.5 EGOS for VME Interface, VIOP EGOS

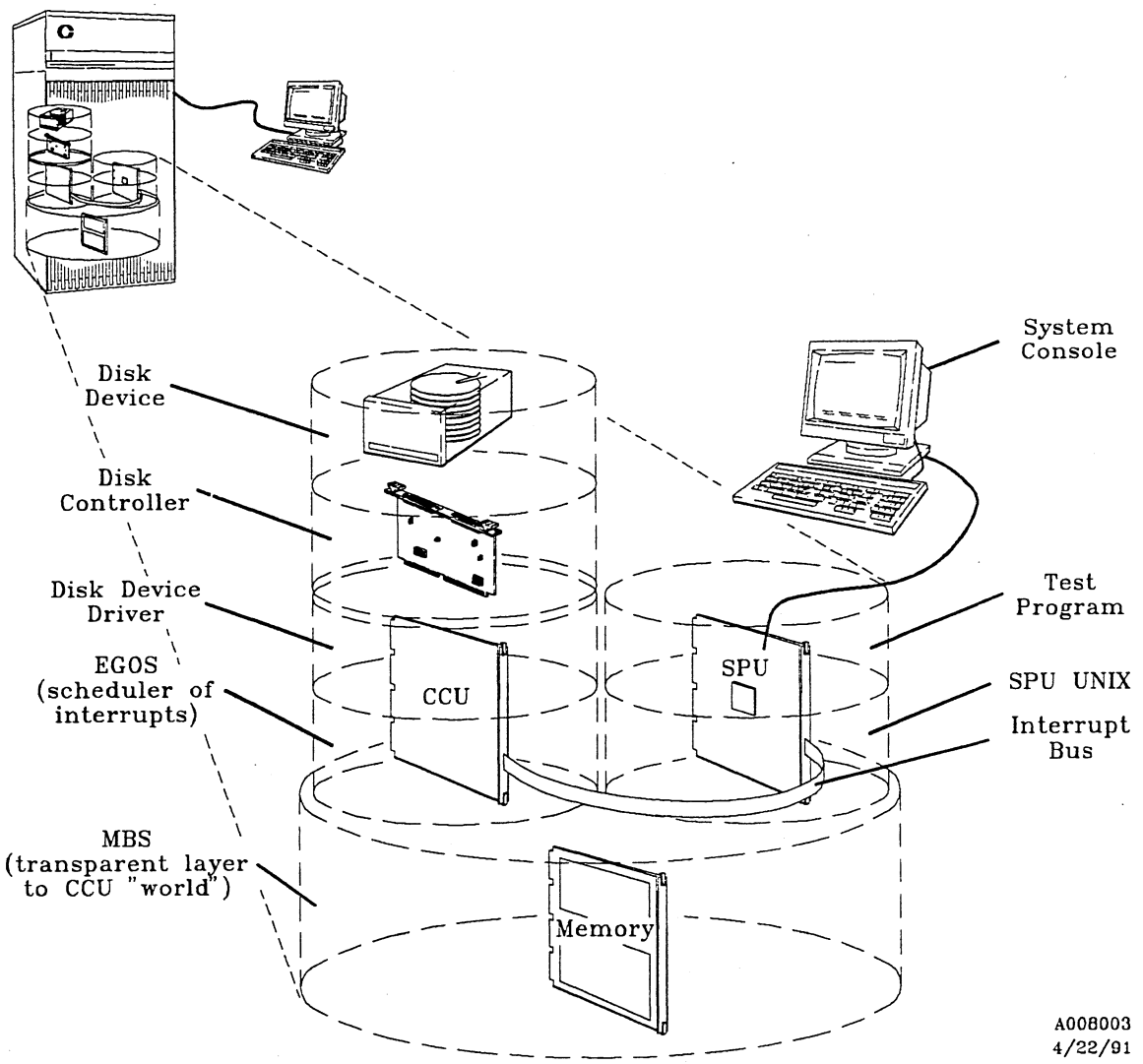
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes it less complicated to create user written device drivers. Each process device driver has a priority level that can be defined relative to other processes. The scheduler supports 32 process priorities and is preemptive for higher priority processes. The VIOP hardware supports 14 device events for event driven device drivers. The 14 levels actually share 2 68020 interrupt levels. Therefore, two is the maximum number of processes at any given time.

2.6 EGOS Position in the Environment

EGOS is positioned in the operating environment between the actual device driver and MBS. MBS is a transparent layer that bridges the CCU and its resources to SPU UNIX. SPU UNIX handles many of the message manipulations that occur during testing. Many error messages that occur during diagnostics testing come from the device driver. When the device driver detects an error from the controller, it calls a routine in EGOS that places a message in the MBS system. This causes SPU UNIX to be interrupted and it retrieves the message from MBS. SPU UNIX then passes a signal to the test program. The test program then prints an error message to the console based on the code that it received.

The following figure illustrates the position of EGOS in the operating system environment.

Figure 2–1, EGOS' Position in the Environment



THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3
Dshell Overview

3.1 Overview

This chapter provides a brief overview of the *dshell* utility. Included in this overview is an overall explanation of the utility and a list of the utility's commands. For a complete description of this utility, refer to the Dshell chapter of the *CONVEX Diagnostic Utilities Manual (C200 Series)* or the *CONVEX Diagnostic Utilities Manual (C1, C120)* depending on the architecture of the machine under test.

3.2 Diagnostic Shell (*dshell*) Overview

The Diagnostic Shell (*dshell*) is a command interface program that runs on the Service Processor. Most of the diagnostics available for the CONVEX machines are interfaced through the *dshell*. Certain peripheral diagnostics are run as standalone tests. To determine whether a test can be run under the *dshell*, consult the appropriate chapter in this manual.

The *dshell* has two basic functions:

- Selecting diagnostics for execution
- Selecting test options
  - Pause on a failure or at the beginning or end of any specific subtest
  - Loop on a specific type of subtest or on a given set of subtests
  - Select subtest execution order
  - Direct test output to a file or to the screen (or both) to monitor the test as it runs or to analyze test results later
  - Select long or short error messages, or turn messages off
  - Execute either user-created or predefined command scripts

The following table list the various *dshell* commands and their functions.

Table 3–1, *dshell* Commands

COMMAND	FUNCTION
! [command]	This command is used to access, or *fork* a UNIX shell to execute the command that follows *!*.
exit	The *exit* command causes immediate termination of the *dshell* process and any test processes that may have been forked.
quit	The *quit* command causes immediate termination of the *dshell* process and any test processes that may have been forked.
^C	Returns user to the *dshell* command level if no subtest is running.
^B	Immediately terminate the *dshell* and any associated active processes. Core is dumped.
help	The *help* command causes a standard *help* menu to be displayed. The menu describes the correct command syntax for each *dshell* command and gives a terse description of what each command does.
status	The *status* command generates a report on the current state of the *dshell* command options. This report gives the name of each flag, its current value, and an explanation of its current effect.
log [options]	The *log* command provides a mechanism for specifying the number of failures that will be allowed to occur before a test or subtest terminates execution.
loop [options]	The *loop* command causes the *dshell* to repeat the execution of a test or subtest.
msgs [options]	The *msgs* command enables or disables different levels of test, class, and subtest result messages.
pause [options]	The *pause* command returns program control to the *dshell* to the beginning, end, or failure of all or specific subtests.
test [options]	The *test* executes specific tests, and displays test, class, and subtest menus.

3.3 Syntax Help for *dshell* Commands

The syntax for each *dshell* command can be obtained by typing the command with no options and pressing <**CR**>. For example, by entering **loop** and pressing <**CR**>, the syntax help in the following figure will be displayed on the screen:

Figure 3–1, Syntax Help for the *loop* Command

```
: loop
Proper syntax is:

loop off (-s) (-t)                          :disables loop modes
loop -s nnn                                 :loop on subtest nnn
loop -t                                     :loop on test
```

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4
Multibus HYPERchannel Controller Test (*dev4510*)

4.1 Overview

The *dev4510* test verifies the functionality of an IKON 10077-NSC Hyperchannel controller board. This board interfaces to the Network Systems Corporation (NSC) A400 Hyperchannel Adapter which, in turn, connects to the actual Hyperchannel bus.

4.2 Prerequisites and Required Equipment

The following table lists the required hardware depending on the type of machine under test.

Table 4–1, Hardware Requirements

C1, C120	C200 Series
MCU	Memory System[1]
MAU	CPX
SPU	SP2
IOP	IOP
MBCU	MBCU
	PIA

[1] Memory System consists of a minimum of one pair of memory boards (one odd and one even).

NOTE

A loopback cable, CONVEX cable 601-330001-200, is required for Class 2 subtests.

NOTE

The controller board must be detached from the A400 during this test. The voltages must be within margin and the clock must be between 5Mhz and 11Mhz.

This test is *not* designed to verify the IOP or Multibus Control Unit (MBCU). However, the test is coded to protect itself against Multibus or IOP failures if such are encountered.

4.3 Test Invocation

The *dev4510* test executes under the Diagnostic Shell (*dshell*) and supports all the features of the *dshell*. The *dshell* permits tests to be initiated in any order.

To invoke the *dev4510* test, use the procedure shown in the following figure. All responses in **boldface** are entered by the user. The prompts and responses appear sequentially on the screen, one line at a time. All prompts and responses are shown in one figure for convenience.

Figure 4–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4510 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

NOTE

After entering **dshell**, specific *dshell* parameters may be changed. Refer to the "Dshell Overview" chapter of this manual for more information.

Entering only **test dev4510** executes all *dev4510* subtests sequentially. Execute a specific class(es) of subtest(s) or one or more individual subtests by using the **-c** or **-s** options, respectively. Detailed information for using these options can be found in the "Dshell Overview" chapter of this manual. The [+>*filename*] option allows the test results to be appended to *filename*.

The following alternate test invocation procedure may be required in some cases.

CAUTION

The user response, **initall**, is typically required if the *initall* utility has not been run since the last power up. However, if any problems have occurred subsequent to the last time *initall* was run, (i.e., system crash, hard error, or failure of previous diagnostic), it should be run again. In this case, failure to run *initall* could result in invalid test results.

NOTE

The *initall* utility requires a significant amount of time (2 to 3 minutes depending on whether the control stores have been previously loaded) to execute. If no system abnormalities have occurred subsequent to the last time the system was booted or *initall* was executed, it is not necessary to run *initall*.

Figure 4–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4510 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

4.3.1 Test Parameter Menu

Once the test is invoked, a test menu prompt is presented allowing selection of default switches. The following figure shows all prompts, their possible answers (in brackets [ ]), and their default answers (in parentheses ( )). The prompts and responses in the following figure appear sequentially on the screen, one line at a time. All the prompts and responses are shown in one figure for convenience.

For help or information during test parameter entry, enter one of the following characters followed by a RETURN:

Table 4–2, Getting Help During Test Parameter Entry

CHARACTER	DESCRIPTION
?	Provides the help information
h	Provides help for a specific prompt
i	Displays the */ioconfig* file

After the information displays, the system beeps and redisplays the last prompt.

The **Test Parameter Menu** illustrates *all* questions that can be displayed during test parameter input. However, some questions may be omitted, depending on answers to previous questions. In all cases, questions are numbered sequentially. However, the numbers displayed on the screen during testing may not correspond to those shown in the example **Test Parameter Menu**, as the questions illustrated are examples only.

Figure 4–3, Test Parameter Menu

```
                ENTER TEST PARAMETERS

     []   Encloses allowed input ranges or values
     ()   Encloses the default value
     ^    Returns to the previous prompt
     :nn  Returns to the prompt # nn
     :    Returns to the first unsatisfied prompt
     :?   Reviews previous entries
     ?    Prints an additional help menu

           PERIPHERAL CONFIGURATION DATA
      CCU    Chassis  TYPE    CSR   Int Unit Type
     ------- -------- ------- ----- --- ---- -------
1)   iop 3      0     LAN-002 0xf00  3   0   HYP-001

Enter device 0 to begin user-defined configurations

  1: Device Selection [0,1]                          (0) ->
  2: IOP [3-7][1]                                    (3) ->
  3: Multibus Chassis [0-3]                          (0) ->
  4: Controller Offset in Multibus [0x0-0xfff]
                                                  (0x5c0) ->
  5: Interrupt number [0-7]                          (5) ->
  6: Enter OK, or :NN to return to question NN [OK]
                                                    (OK) ->
```

[1] The possible selections for this prompt will change depending on machine architecture.

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parameter Menu. The following list summarizes the available options:

:nn — Returns to an earlier prompt (n is the prompt number)

: — Advances to the next unanswered prompt

:? — Displays (reviews) all responses up to the current prompt

? — Requests help for the current prompt (if available)

^ — Returns to the previous prompt

4.3.2 Prompt Explanations

A description of the meaning of each prompt follows:

```
Device Selection [0,1]                          (0) ->
```

This prompt allows selection of a controller from the system configuration. If **0** (default) is entered, then user-defined prompts are displayed to determine the configuration to be tested. Otherwise, if a device is entered, the prompts are not displayed.

```
IOP [3-7]                                          (3) ->
```

Enter the CCU slot number for the desired IOP. Then additional prompts are displayed requesting hardware configuration information.

```
Multibus Chassis [0-3]                             (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                                (0x5c0) ->
```

Enter the low-order 12 bits of the controller's address within the Multibus (this address is selected with switches on the controller).

```
Interrupt number [0-7]                             (5) ->
```

Enter the interrupt level of the controller within the Multibus (this is selected with switches on the controller).

```
Enter OK, or :NN to return to question NN [OK]
                                                  (OK) ->
```

If **OK** or RETURN is entered, the test parameter menu terminates and all inputs are no longer changeable.

When all prompts have been answered, the screen displays a test parameter summary which echos the prompts that have been answered. The following figure illustrates an example of a "Test Parameter Summary" screen. The actual values and responses vary according to the input.

Figure 4–4, Sample Test Parameter Summary

```
            TEST PARAMETER SUMMARY

IOP                                          : 3
Multibus Chassis                             : 0
Controller Offset in Multibus                : 0x5c0
Interrupt number                             : 5
Enter OK, or :NN to return to question NN    : OK
```

4.4 Hardware Initialization Sequence

After the last prompt is entered, and before test code execution, the following events occur:

- A sysreset is performed
- Main memory is allocated for the test
- SPU windows to main memory are initialized
- SPU local test variables are initialized
- The IOP is booted and loaded
- A driver on the IOP is started
- IOP local test variables are initialized

After all the above events have occurred, the test code is started.

4.5 Class Descriptions

The *dev4510* test contains the following two classes of subtests as shown in the following table:

Table 4–3, *dev4510* Test Classes

CLASS	DESCRIPTION
1	IOP/IKON 10077-NSC communications tests
2	IKON 10077-NSC loopback tests

Any subtest may be started by itself. There are no subtest dependencies. However, on an unknown board, it is best to execute all the subtests in their default order since the subtests increase in complexity.

4.5.1 Class 1 Subtests

Class 1 subtests verify the basic interface from the IOP to the controller is functional. This ensures that the IOP can read and write the three control and status registers on the IKON board and reset the board. These tests can be run with or without the loopback switch on (switch 2 of block U26) and with or without the loopback cable. Therefore, the controller can be configured for loopback and then run both classes. Class 1 subtests are listed in the following table:

Table 4–4, Class 1 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
100	Master Clear and ISR/ICR Register Check	0:01
101	Master Clear Through PCR Register	0:01

4.5.1.1 Subtest 100, Master Clear and ISR/ICR Register Check

Subtest 100 checks basic communications between the IOP and Interface board using the Interface Status Register (ISR) and Interface Control Register (ICR). Basically, values are loaded into the ICR and read back in the ISR.

A master clear is performed through the ICR and verified since the pattern in the ISR after a master clear must be a binary 00X0XXX010000001 where X is an unknown state bit.

Four of the ISR bits can be programmed from the ICR without loopback. These four bits are altered with walking '1's and walking '0's patterns. This operation is repeated 1000 times.

4.5.1.2 Subtest 101, Master Clear Through PCR Register

Subtest 101 checks that the Pulse Command Register (PCR) can successfully perform a master clear. First, a 4 bit pattern is written to the CYCL, FCN1, FCN2, and FCN3 bits of the ICR and then master clear is pulsed through the PCR. All four bits previously set high should now be low. The master clear is repeated 1000 times with a verify each time.

4.5.2 Class 2 Subtests

Class 2 subtests are performed with the loopback switch of the IKON 10077-NSC board set on (switch 2 of block U26 on the board) and with a loopback cable attached to the board. These loopback features allow most of the board to be tested.

> **NOTE**
>
> A loopback cable, CONVEX cable 601-330001-200, is required for Class 2 subtests.

- Function and status bits in the control and status registers can be verified.
- 16-bit at a time transfers can be performed.
- DMA input and output of blocks of data can be tested.
- Interrupts from the IKON board to the A400 and to the IOP are exercised.

Class 2 subtests are listed in the following table:

Table 4–5, Class 2 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
200	Verify Programmability of ISR/ICR	0:01
201	Perform 16-bit Transfers	0:02
202	Perform DMA from Main Memory Using ICR	3:50
203	Perform DMA to Main Memory Using ICR	2:05
204	Perform DMA from Main Memory Using PCR	1:45
205	Perform DMA to Main Memory Using PCR	1:00
206	Verify DMA Transfers Without Interrupts	0:10
207	Verify IKON Responds to Attention Interrupt	0:30

4.5.2.1 Subtest 200, Verify Programmability of ISR/ICR

Subtest 200 performs a more comprehensive test of the ISR/ICR registers using the internal loopback enabled by switch 2 of switch block U26. This test is repeated 1000 times.

4.5.2.2 Subtest 201 Perform 16-bit Transfers

Subtest 201 performs two-byte transfers with a rotating ones bit and then a rotating zero bit. This operation verifies that each communications line toggles and that there is no crosstalk on the lines. The pattern test is repeated 1000 times.

4.5.2.3 Subtest 202, Perform DMA from Main Memory Using ICR

Subtest 202 verifies the controller can fetch bytes from main memory and write them to the output port. The last word transferred can be verified by reading the Input Data register (IDR). To ensure that every available window can be read, all windows are allocated and read from. Two bytes are read starting at each window address 0xXXX000 and 0xXXXffe. Then windows are set up for one large area of 64 Kbytes. DMA transfers of varying lengths of 1024, 2048, 4096, etc. bytes up to the 64k limit are performed. Each length of data is repeated for 32 data patterns consisting of walking '1's and walking '0's patterns. The ICR is used to start the DMA transfer.

4.5.2.4 Subtest 203, Perform DMA to Main Memory Using ICR

Subtest 203 uses the controller's onboard DMA to transfer data to main memory from the Output Data Register (ODR). A two-byte pattern is stored in the ODR and then is written repeatedly to main memory until the entire DMA block has been transferred. DMA transfers vary in length from 1024, 2048, 4096, etc. bytes up to 64 Kbytes. Each length of data is repeated for 32 data patterns consisting of walking '1's and walking '0's patterns. The ICR is used to start the DMA transfer.

4.5.2.5 Subtest 204, Perform DMA from Main Memory Using PCR

Subtest 204 performs the varying length DMA operation described in Subtest 202 but uses the PCR to start the DMA transfer.

4.5.2.6 Subtest 205, Perform DMA to Main Memory Using PCR

Subtest 205 is the same as Subtest 203 but uses the PCR to start the DMA transfer.

4.5.2.7 Subtest 206, Verify DMA Transfers Without Interrupts

Subtest 206 transfers varying lengths of data as in Subtest 202 and waits for the indication in the ISR that each transfer is performed. After the transfer completes, interrupts are also checked to verify none occurred. The DMA flag is then reset and verified that it reset properly. The next transfer then begins.

4.5.2.8 Subtest 207, Verify IKON Responds to Attention Interrupt

Subtest 207 performs a master clear using the ICR and confirms it was successful by inspecting the results in the ISR. Then it confirms no interrupt is pending and sets FCN2 (a special loopback bit) and IENB (interrupt enable) in the ICR. This operation is supposed to cause an interrupt as if the adapter had sent it. The interrupt is verified.

A second check verifies an ATTN interrupt occurs by inducing it with the loopback bit FCN2 and IENB (interrupt enable). The interrupt is verified.

4.6 Troubleshooting Guide

This is a functional test. Standard error reporting routines will report whether a problem is internal or external to the controller board. In addition, many errors have associated with them a list of possible causes that are printed if long error reporting is selected within the dshell (long error reporting is the default). The IKON 10077-NSC Hyperchannel controller board is the only FRU covered by this test.

4.7 Interprocessor Protocol

The IOP/SPU communication protocol is supplied by the Message-Based System (MBS).

4.8 Error Messages

0x40 Sysreset of ccus and mem failed

The SPU UNIX command *sysreset ccus; sysreset mem* returned a non-zero value. Try the command manually. May require running *io4000* or *mem4000* to isolate the problem.

0x42 IOP print utility memory error
Error in memory access within MBS. IOP-generated output could not be printed.

0x80 Message received on wrong subqueue
Errors 0x80 through 0x83 should never occur. If they do, see the note at the end of this section.

0x81 Wrong subqueue active and locked
Refer to error 0x80.

0x82 Wrong subqueue active and MBS error
Refer to error 0x80.

0x83 Wrong subqueue active, error invalid
Refer to error 0x80.

0x84 Driver executed for no apparent reason
The IOP driver was brought into execution but no reason could be found. Normally, the driver is executed because of a message sent to it from the SPU or because a controller has interrupted the IOP with the correct interrupt signal. Refer to the note at the end of this section.

0x85 Master Clear through the ICR failed
When the master clear bit in the ICR was set, the ISR failed to return to the correct bit pattern. Probably a bad controller.

0x86 Write/read of ISR bits failed
Unable to alter the ISR bits. Probably a bad controller.

0x87 Controller is not present
The IOP received a bus error when attempting to write to the controller's registers. It is most likely the board is not inserted correctly in the Multibus chassis. The controller could be bad.

0x88 Master Clear through the PCR failed
When the master clear bit in the PCR was set, the ISR failed to return to the correct bit pattern. Probably a bad controller.

0x89 Switch 2 of U26 on Cntlr is not ON
An internal loopback was attempted which only succeeds if the described switch is set ON. An internal failure of the controller board is possible.

0x8A Loopback cable is not installed
An external loopback was attempted which only succeeds if the loopback cable is attached properly to both ports of the controller. The cable may be built incorrectly. Refer to the section Prerequisites and Required Hardware for the cable part number. An internal failure of the controller board is possible.

0x8C DMA from memory failed compare
The last two bytes transferred from memory are checked in the IDR for correctness. The compare failed. The failure could be located anywhere along the path from main memory to the controller board, the controller board itself, or the loopback cable.

0x8D DMA to memory failed compare
Refer to error **0x8C.**

0x8E Expected interrupt did not occur
The controller was expected to interrupt in less than 128 msec. No interrupt was detected in this time. Either the interrupt is strapped incorrectly on the board or the board is failing to interrupt when it should.

0x8F Programmed I/O failed data compare
The transmitted and received data from two-byte transfers failed. Either the loopback cable is not working properly, some component along the path from main memory to the controller is malfunctioning, or the controller itself is bad.

0x92 IOP Window Allocation Failed
Setup of a window from the IOP to main memory resulted in an error. Either the IOP or the driver software on the IOP failed. Refer to the note at the end of this section.

0x93 ISR not reset when RDMA & RATN pulsed
At the beginning of several tests, a master clear is performed by pulsing two bits in the ICR or PCR. This message is printed when the ISR is checked afterward and found not to be in a reset state.

0x94 ISR not in proper state after DMA
The ISR is checked after every DMA operation to verify attention is not set (ATTF = 0), DMA is active (DMAF = 1), and ready is true (REDY = 1). One of these conditions was not correct.

0x95 Unexpected interrupt occurred
An interrupt occurred when none was expected. This occurs in Subtests 206 and 207 during portions of the subtests when no interrupt is expected.

0x96 Extended Write/read of ISR bits failed
The test of bits in the ICR and ISR failed because the ISR did not correctly reflect changes made to the ICR.

0xc0 Error when attempting to receive a message
The MBS reported an error in attempting to provide the SPU with a message.

0xc1 Timeout because queue locked
The SPU received a message, but all ten attempts to read the message resulted in the error *MBS queue locked* being reported by the MBS.

0xc2 Msg interrupt but no msg found
The SPU attempted ten times to receive a message when an interrupt occurred, but all attempts failed with a *no message* error from the MBS.

0xc4 Timeout awaiting msg
SPU expected the IOP to send a message, but it did not receive a message. This error message indicates that the controller failed to generate an interrupt when done, or a break exists in the path of the interrupt signal, which prevents it from being detected.

0xc5 Rtn msg from subqueue other than 7
The SPU received a message to the wrong subqueue; all messages are expected to arrive on subqueue seven. Refer to the note at the end of this section.

0xc6 MBS returned bad error code
A call to an MBS routine returned an error code that was unrecognizable.

0xc7 Block transfer routine failed
A memory to memory transfer failed using the function 'blt'. Refer to the note at the end of this section.

0xc8 Too many devices failed. Fail subtest
This message indicates that the test is terminating because the device failure limit was exceeded.

0xc9 Return msg not of valid type
The SPU received a message, but it was not a device result or system error message. Refer to the note at the end of this section.

0xca IOP's queue locked too long
Ten attempts to send a message to the IOP failed, with a locked condition being reported by MBS.

0xcb No room on SPU side for msg from the SPU to the IOP
Ten attempts to send a message failed because no free message MBS blocks were available for the SPU to use.

0xcc Return msg error
An attempt was made to send a message from the IOP to the SPU but resulted in an MBS error being reported by the Message-Based System.

0xcd Error while configuring I/O
The IOP returned a configuration message that specified a controller that does not exist. Refer to the note at the end of this section.

0xcf Too many controllers specified
More than six controllers were specified during user inputs.

0xd0 IOP echoed msg when none sent
The SPU received an echo message, but no message had been sent. Refer to the note at the end of this section.

0xd1 Bad cntlr # in returned msg
A drive result message specified a bad controller number. Refer to the note at the end of this section.

0xd2 Outstanding msg cnt greater than 0
About to start a task in a subtest when a check uncovers that there are still messages expected back from the IOP for the last task. Refer to the note at the end of this section.

0xd3 No msg err when sending to SPU
Ten attempts to send a message from the IOP to the SPU resulted in a in use. Refer to the note at the end of this section.

0xd4 MBS error when sending to SPU
Same as error code 0xd3 except that MBS is reporting an 'MBS_ERROR'.

0xd5 Invalid MBS error when sending to SPU
MBS is failing. It reported back an error code which is invalid. Refer to the note at the end of this section.

0xd6 MBS locked when sending to SPU
Same as error code 0xd3 except that MBS is reporting an 'MBS_LCKD' error.

0xd7 IOP driver device is bad
A request to test an unknown device was sent from the SPU to the IOP. Immediately after the IOP was loaded, it was sent information about the device that was to be tested. However, the latest request to test the device had bad information in it about the device. Refer to the note at the end of this section.

0xd8 MBS error during IOP msg recv
Same as error code 0xd4 except that it occurred during receipt of a message from the SPU to the IOP.

0xd9 MBS queue locked during receive
Same as error code 0xd6 except that it occurred during receipt of a message from the SPU to the IOP.

0xda MBS no message available
Same as error code 0xd3 except that it occurred during receipt of a message from the SPU to the IOP.

0xdb Data chk of echoed msg failed
A data compare of all echoed messages occurs on the SPU. If the compare fails, MBS is having difficulty sending message between the IOP and SPU. A main memory or software error is indicated. Most likely main memory is not working properly so try *mminit* and then rerun. If it still fails, try *mem4000.* Refer to the note at the end of this section.

0xdc Bad message type to get_rtn_msg
A C-function called get_rtn_msg was called to receive an echoed message or the IOP did not properly flag a message so that it looked like an echoed message. On the SPU side, the routine mbs_wait should have been called if an echoed message is actually expected. Refer to the note at the end of this section.

0xdd Processor queue setup failed
A call to the OS system utility, *pqutil*, returned a non-zero value which indicates it failed. Try it manually at the SPU prompt by typing **pqutil -I**. If it fails for any reason, then it cannot initialize memory in the first eight pages of main memory. Try running *mem4000.* If that fails, try reloading */mnt/os* from a backup or release tape and try again. *Dev4100* and *dev4500* also use *pqutil* so maybe try them.

0xde Open for window to main mem failed

An open of the system file */dev/wndw* was attempted and failed. Verify */dev/wndw* exists and has proper permissions.

0xe0 IOP load module (*.x00) not fnd

A check was made of the current directory first and then */mnt/test* for a file called *dev4510.x00*. It was not found. It should exist in */mnt/test*. If it is not there, restore */mnt/test* from a backup or release tape.

0xe1 Error attempting to load iop

A system command to execute *loadccu* failed. First, run *sysreset* and *mminit* and then try it again. If this does not work, try typing **loadccu -3s /mnt/test/dev4510.x00** at the SPU prompt. If this fails, verify */mnt/os/loadccu* exists and has the right permissions. Also verify that */mnt/test/dev4510.x00* has the right permissions.

0xe2 Main memory allocation error

One large block of main memory needs to be allocated by the test on the SPU side but failed. Run *mminit* and then rerun the test. If it still fails, try *mem4000* to verify memory is functional.

NOTE

Certain error messages may indicate a software error. To verify for a software error, perform the following steps:

1. Perform *sysreset* and *mminit -s*.
2. Rerun the failing subtest. If the problem repeats continue with step 3. If no error occurs, record the error in the error log and look for recurrences of the error.
3. Reload */mnt/test* from a backup or release tape. Rerun the failing subtest. If the problem repeats, report it.

Appendix A
Reporting Problems

A.1 Overview

This appendix introduces the CONVEX Technical Assistance Center (TAC) and the *contact* utility. The *contact* utility is an online system for reporting problems to the TAC. To learn *contact* by using it, enter **contact** at the system prompt and then answer the questions as they appear on the screen. To find out more about using *contact*, read through this appendix. It describes prerequisites and tips for using *contact* and the step-by-step process *contact* takes you through.

A.2 Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address the diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation problem, contact the TAC. This group stands ready to solve such problems.

A.3 The *contact* Utility

The TAC recommends using the *contact* utility to report a hardware, software, or documentation problem. The *contact* utility is an interactive utility that helps the TAC track reports and route them to the the CONVEX personnel most qualified to fix them.

After invoking *contact*, it prompts for information about the problem. When you finish your report, *contact* electronically mails it to the TAC. You are notified within 48 hours that the TAC has received your report.

A.4 Prerequisites

To use *contact* requires

- a UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- the full path name of the program or utility in question
- the version number of the program or utility in question

A.4.1 UUCP Connection

Before using *contact*, check with your system administrator to be sure there is a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX system to another. The *uucp* (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

A.4.2 Finding the Program Path Name

To determine the full path name of the program or utility in question, use the *which* command. The following screen illustrates using the *which* command to find the full path name of the loader (*ld*) utility:

```
>which ld
/bin/ld
>
```

In this example, the full path name of the loader is */bin/ld*.

For more information on the *which* command, refer to the *which*(1) man page. You can also use the *info* online information system. Enter **info which** at the system prompt. If you use the C shell (*csh*), you can also use the *whence* command to find the program path name. The *whence* command works like *which*, only faster.

A.4.3 Finding the Program Version Number

To determine the version number of the program or utility in question, use the *vers* command. The following screen illustrates using the *vers* command (enter **vers**, then the path name of the program or utility) to find the version number of the loader (*ld*) utility.

```
>vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader utility version number is 7.0.

For more information on the *vers* command, refer to the *vers*(1) man page. You can also use the *info* online information system. To do so, enter **info vers** at the system prompt.

A.5 Tips on Using the *contact* Utility

The *contact* utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to

- use a *.contact* file
- abort a contact session
- resubmit an aborted report
- suspend a contact session
- move from one prompt to another
- use tilde-escape sequences in the *contact* utility

A.5.1 Using a *.contact* File

When invoked, *contact* prompts for information regarding the problem. The first prompt is for your name, title, phone number, and company name. You can, however, create a *.contact* file to skip this first prompt. Follow these steps:

1. Create a *.contact* file in your home directory.
2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke *contact*, it automatically includes the *.contact* file as input for the first prompt and proceeds to the next prompt.

A.5.2 Aborting the Report

To abort a contact report, either enter the interrupt key (usually CTRL-C) or choose the abort option when prompted by the *contact* utility. Using CTRL-C to abort does not save the contents of the report. Using the abort option saves the contents of the report in a file named *dead.report* in your home directory.

A.5.3 Submitting the *dead.report* File

When aborting a contact session, the *contact* utility saves the report in a file named *dead.report* in your home directory. Using the *contact* command with the *-r* option automatically merges the contents of the *dead.report* file into the new contact session. Enter

contact -r

and *contact* finds the *dead.report* file in your home directory and merges it into the contact report. You can then edit the report. When you end the editing session, *contact* returns to the final prompt, which asks you to review, edit, submit, or abort the report.

A.5.4 Suspending a Report

Sometimes it is necessary to stop in the middle of a contact report and return to the shell (for instance, to suspend the contact session to find the program path name or version number). To suspend the contact session, press CTRL-Z. To return to the contact session, enter **fg**. Using CTRL-Z and the *fg* (foreground) command lets you switch back and forth between the *contact* utility and the shell. You cannot, however, use CTRL-Z and *fg* to switch back and forth if you are using a Bourne shell (*sh*).

A.5.5 Ending a Response

The *contact* utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press RETURN. Other prompts require more than a one-line response; to move to the next prompt, press CTRL-D.

A.5.6 Tilde-Escape Sequences

The *contact* utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. The following tilde sequences are recognized by *contact*:

~e	Start the text editor (defined in your EDITOR environment variable).
~h	Display a list of available tilde-escape sequences.
~p	Print the contact report to the terminal screen.
~r *filename*	Read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence only works for prompts that allow more a than one-line response.
~~	Insert a single tilde as the first character in the line.

A.6 Using the *contact* Utility

The *contact* utility prompts for the following information:

- your name, title, phone number, and corporate name
- the name and version of the product involved
- a one-line summary of the problem
- a detailed description of the problem
- the priority of the problem
- instructions on how to reproduce the problem
- comments about the problem
- comments about the documentation supporting the problem
- files to include in the contact report

The following is a step-by-step discussion of these prompts:

1a. To invoke the *contact* utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software, or documentation question. The following screen illustrates the *contact* command and the system response:

```
>contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

1b. If there is a *.contact* file in your home directory, *contact* skips the first prompt. The following screen illustrates the *contact* command and the system response when a *.contact* file is in your home directory:

```
>contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

2. The *contact* utility prompts for the version number of the product. If you do not know the version number, use CTRL-Z to suspend the session. Use the *which* (or *whence* if using *csh*) and *vers* commands to find the version number of the product. Use the *fg* command to return to the session and enter the version number in the form X.X or X.X.X.X.
3. The *contact* utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Make this summary as descriptive as possible in one line.
4. The *contact* utility prompts for a detailed description of the problem. Make this description as complete as possible. Include source code and a stack backtrace whenever possible. (Refer to the *adb*(1) or *csd*(1) man page for information on obtaining a stack backtrace.) The more information provided, the quicker the TAC can isolate and solve the problem.
5. The *contact* utility prompts for the priority of the problem. The following screen illustrates this prompt and the priority levels from which to choose; you must enter a priority number.

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

6. The *contact* utility prompts for an explanation of how to reproduce the problem. Include the command syntax and options you used and anything else you did to make your program run.
7. The *contact* utility prompts for any other pertinent comments. Include any relevant information.
8. The *contact* utility prompts for suggestions regarding the documentation supporting the product. Indicate if the documentation could be revised to address the question.
9. The *contact* utility asks for the names of files necessary to reproduce the problem. The following screen illustrates the *contact* prompt and sample user response:

```
Are there any files that should be included in this report (yes | no)?
>yes
Please enter the names of the files, one to a line (^D to terminate)
>test.f
>~/subroutines/sub.f
>
```

> **NOTE**
>
> Tilde-escape sequences are not recognized in responses to this prompt. Instead, *contact* treats a tilde in this section to mean your home directory. This convention is based on use of the tilde for expanding file names in *csh*.

If the files specified are small text files, they are automatically included in the contact report. If the files are too big to be included in this report, *contact* gives further instructions on how to submit these files.

To specify a directory, combine the directory files into a single file using the *tar* command (refer to the *tar*(1) man page for further information) or enter each file name in the directory on a single line in the contact report.

10. The *contact* utility prompts you to review, edit, submit, or abort the contact report. The following screen illustrates this prompt:

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review — Review the text of your contact report. You are then prompted again to select an option.

Edit — Edit the text of the contact report. If you choose to edit the report, *contact* puts you in your default text editor.

Submit — Send the report to the CONVEX TAC. You are notified within 48 hours that the TAC has received the report. This option exits the *contact* utility and returns you to the shell environment.

Abort — Save the text of your report in a file named *dead.report* in your home directory. This option exits the *contact* utility and returns you to the shell environment.

Index

T

U

V

W

CONVEX Multibus HYPERchannel Controller
(*dev4510*) Diagnostics Manual
Document No. 760-002630-000
First Edition

Reader's Forum

Please use this form to submit comments or questions concerning the clarity and service of this manual. Constructive critical comments are most welcome and help us continue in our efforts to generate quality customer documentation. Please list the page number for questions or comments.

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____Title_____

Company_____Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION OR DOCUMENTATION:

Location	Phone Number
From all locations in continental U.S.	1(800)952-0379
From locations in Alaska & Hawaii	1(214)497-4379
From locations in Canada	1(800)345-2384
From all other locations	Contact nearest CONVEX office

Direct mail orders to:

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851 USA

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851

(Fold Here Second)

(Tape or Staple)