CONVEX VMEbus Ethernet Controller (*dev5500*) Diagnostics Manual

Document No. 760-003330-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX VMEbus Ethernet Controller (dev5500) Diagnostics Manual
Order No. DHW-245
First Edition

© 1991 CONVEX Computer Corporation
All rights reserved.

This document is copyrighted. All rights reserved. This document may not, in whole or part, be copied, duplicated, reproduced, translated, electronically stored or reduced to machine readable form without prior written consent from CONVEX Computer Corporation (CONVEX).

Although the material contained herein has been carefully reviewed, CONVEX does not warrant it to be free of errors or omissions. CONVEX reserves the right to make corrections, updates, revisions, or changes to the information contained herein. CONVEX does not warrant the material described herein to be free of patent infringement.

UNLESS PROVIDED OTHERWISE IN WRITING WITH CONVEX COMPUTER CORPORATION (CONVEX), THE EQUIPMENT DESCRIBED HEREIN IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES. THE ABOVE EXCLUSION MAY NOT BE APPLICABLE TO ALL PURCHASERS BECAUSE WARRANTY RIGHTS CAN VARY FROM STATE TO STATE. IN NO EVENT WILL CONVEX BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, ARISING OUT OF THE USE OR INABILITY TO USE THIS EQUIPMENT. CONVEX WILL NOT BE LIABLE EVEN IF IT HAS BEEN NOTIFIED OF THE POSSIBILITY OF SUCH DAMAGE BY THE PURCHASER OR ANY THIRD PARTY.

CONVEX and the CONVEX logo ("C") are registered trademarks of CONVEX Computer Corporation
C1, C120, C201, C202, C210, C220, C230 and C240 are trademarks of CONVEX Computer Corporation
C100 Series and C200 Series are trademarks of CONVEX Computer Corporation
UNIX is a registered trademark of AT&T Bell Laboratories
ConvexOS is a registered trademark of CONVEX Computer Corporation

Printed in the United States of America

Revision Sheet

CONVEX VMEbus Ethernet Controller (dev5500) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-003330-000	May 1991	First release. Contains the *dev5500* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.*

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

Appendixes

List of Tables

List of Figures

Preface

Purpose and Intended Audience

This manual explains how to run the *dev5500* diagnostic, which checks the Excelan EXOS/202 Ethernet controller board. This document is not a tutorial, but rather a reference for the users of the *dev5500* 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 *dev5500* 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. VMEbus Ethernet Controller Test (*dev5500*)**—Describes how to operate the diagnostic, including prerequisites, test invocation, hardware initialization sequence, and class descriptions. It also describes subtest completion, error, and test end messages.
- **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-245.
The document number for this manual is 760-003330-000.

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

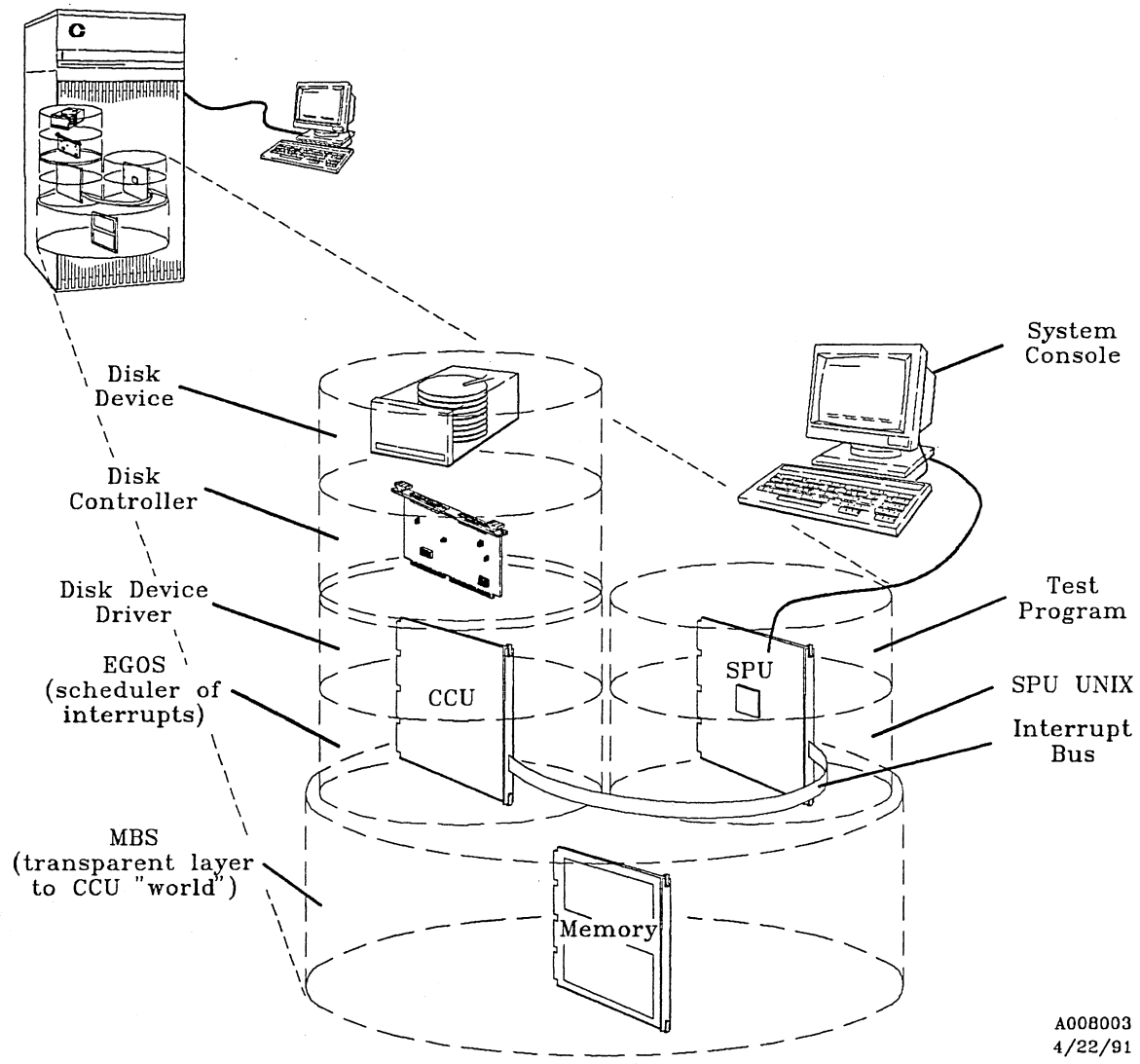
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes 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
VMEbus Ethernet Controller Test (*dev5500*)

4.1 Overview

The *dev5500* test is designed to verify that the Excelan EXOS/202 Ethernet controller board operates properly. The *dev5500* test verifies the VIOP-to-controller interface and triggers the controller board's built-in self-test.

4.2 Prerequisites and Required Equipment

In order to run the *dev5500* test, the following are needed:

Table 4–1, Hardware Requirements

C1, C120	C200 Series
EXOS/202 Controller	EXOS/202 Controller
VBCU	VBCU
VIOP	VIOP
MCU	Memory System[1]
MAU	CPX
SPU	SP2
HIA	HIA
HSP	HSP
FSE	FSE
	PIA

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

NOTE

A *delni* module must be used to run the Class 1 and Class 4 tests. The *delni* module must be connected to the board under test with a CONVEX cable 601-150001-200.

4.3 Test Invocation

The *dev5500* 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 *dev5500* 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

CONVEX DIAGNOSTICS SHELL

: test dev5500 [-c [class number(s)]] [-s [subtest number(s)]] [+>filename] RETURN
```

NOTE

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

Entering only **test dev5500** executes all subtests sequentially. Specific class(es) of subtest(s) or one or more individual subtests can be executed by using the **-c** or **-s** options, respectively. Detailed information for using these options can be found in the "Dshell Overview" chapter within this manual. Using the [+> *filename*] extension causes all test results to be appended to *filename* in addition to printing on the console.

The following alternate test invocation procedure may be required in some cases; however, the CPU *must* be running:

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
CONVEX DIAGNOSTICS SHELL
: test dev5500 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

4.3.1 Test Parameter Menu

Once the test is invoked, test menu prompts are displayed allowing selection of default switches. The first prompt determines which test parameters are displayed. To test a device not listed in the configuration file, answer the first prompt with **0** and prompts two through nine are displayed one line at a time. Prompts two through eight allow the user to describe the location of the controller to be tested. However, answering **1** (the number representing the device to test) to the first prompt causes prompts two through fives to be skipped.

The following figure shows all prompts, with their possible answers (in brackets [ ]), and their default answers (in parenthesis ( )). The prompts and responses in the following figure appear sequentially on the screen, one line at a time. All possible prompts and responses are shown in the following 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
?	Displays this help menu
h	Provides help for a specific prompt
i	Displays the Iioconfig file

After the desired help 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, *dev5500* 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


              PERIPHERAL CONFIGURATION DATA
      CCU     Chassis Type          CSR    Int     Unit    Type
    -------   ------- --------      -----  ---     ----    ------
1)  viop  5     0     LAN-201       0x0     0       0      ex


1: Device Selection [0,1]                          (0) ->
2: VIOP [3-7]                                      (5) ->
3: VMEbus Chassis [0-1]                            (0) ->
4: Controller Offset in VMEbus [0x0-0xffff]
                                                (0x8000) ->
5: Controller Interrupt [0-7]                      (5) ->
6: Subtest loop count [10-1000]                  (100) ->
7: Transceiver connected to active ethernet? [y,n]
                                                   (y) ->
8: Run machine <-> machine comm test? [y,n]        (n) ->
9: Enter OK, or :NN to return to question NN [OK]
                                                  (OK) ->

Device 0 = user defined configuration
```

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) ->
```

The first prompt determines which test parameters are displayed. By responding with **0** or (RETURN), prompts 2 through 8 are displayed one line at a time. These prompts allow specification and testing of a device not listed in the configuration file . Prompts 2 through 8 allow description of the location of the controller to be tested. However, if **1** (the number representing the device to test) is entered to the first prompt, prompts 2 through 4 are displayed one line at a time.

```
VIOP [3-7]                                      (5) ->
```

This prompt allows selection fo the position of the VIOP in the CPU card cage with a value ranging from [**3-7**].

```
VMEbus Chassis [0-1]                            (0) ->
```

If a response of **0** or (RETURN) is entered, then VMEbus chassis 0 is selected. However, if **1** is entered, VMEbus chassis 1 is selected.

```
Controller Offset in VMEbus [0x0-0xffff]
                                          (0x8000) ->
```

If **0x8000** or (RETURN) is entered, then the test assumes this is the controller address. If the controller to be tested is not at this address then the test fails.

```
Controller Interrupt [1-7]                      (5) ->
```

If a **1** or (RETURN) is entered, then interrupt one is used by the controller when an I/O operation completes to interrupt the VIOP. However, if a value in the range of [**2-7**] is entered, this interrupt is used. There is only one requirement when choosing an interrupt number, each controller in a VME chassis must have unique interrupt number. Controllers in different chassis can use the same interrupt number.

```
Subtest loop count [10-1000]                  (100) ->
```

If a **100** or (RETURN) is entered, the number of subtest iterations is set to 100. However, if a value in the range of [**10-99**, **101-1000**] is entered, the number of subtest iterations is set to this value. If a value not in the range of [10-1000] is entered, the prompt is redisplayed.

```
Transceiver connected to active ethernet? [y,n]
                                            (y) ->
```

If a **y** or (RETURN) is entered, Class 2 subtests are not allowed to execute since the subtests require an inactive Ethernet for proper operation. However, if **n** is entered, then Class 2 subtests are allowed to execute.

```
Run machine <-> machine comm test? [y,n]     (n) ->
```

If **n** or (RETURN) is entered, then the Class 4 subtest is not allowed to execute. However, is **y** is entered, the Class 4 subtest is allowed to execute.

```
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.

After all the test parameters are entered, a test parameter summary displays the entries. If standard output is directed to a disk file, the test parameter summary is also written to the disk file.

Figure 4–4, Sample Test Parameter Summary

```
                TEST PARAMETER SUMMARY

Device selection                                  : 0
VIOP                                              : 5
VMEbus Chassis                                    : 0
Controller Offset in VMEbus                       : 0x8000
Controller Interrupt                              : 5
Subtest loop count                                : 100
Transceiver connected to active ethernet?         : y
Run machine <-> machine comm test?                : n
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 VIOP is booted and loaded
- A driver on the VIOP is started
- VIOP local test variables are initialized

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

4.5 Class Descriptions

The VME Ethernet controller test *dev5500* contains four classes of subtests. The following table lists each class of subtests:

Table 4–3, *dev5500* Test Classes

CLASS	DESCRIPTION
1	EXOS/202 Ethernet controller access test
2	EXOS/202 Ethernet controller functional tests
3	EXOS/202 Ethernet controller online test
4	EXOS/202 Ethernet controller net transmit test

4.5.1 Class 1 Subtest

The EXOS/202 Ethernet controller access test verifies the basic interface from the VIOP to the controller. This ensures that the controller responds to the I/O address expected, and passes an internal self-test.

Table 4–4, Class 1 Subtest

SUBTEST	DESCRIPTION	TIME (min:sec)
100	Controller to Host Access	0:05

4.5.1.1 Subtest 100, Controller to Host Access

Subtest 100 verifies the basic access paths. The subtest resets the controller board to verify that VIOP can access the controller, and that the controller will automatically run a self-test after reset. Next, the VIOP commands the controller to configure itself from a table in VIOP local memory. This tests the controller's ability to access VIOP local memory. Then the controller is commanded to configure itself from main memory to test controller-to-main memory access.

4.5.2 Class 2 Subtests

The EXOS/202 Ethernet controller functional tests verify the ability of the controller to transmit and receive various Ethernet packets. While in the *Ethernet* mode (with the use of a *delni* module connected to the board under test), these subtests transmit packets using an internal transmit with self-receive command, and transmit messages to verify the controller's address filtering. The "scatter/gather" operation of the controller is also tested by targeting data blocks to specific memory blocks.

> **NOTE**
>
> The *delni* module must be connected to the board under test with a CONVEX cable 601-150001-200.

Each Class 2 subtest resets and reconfigures the controller at the start of the subtest. This allows the subtests to be run in any order. Each subtest can loop for a user-entered number of passes.

Table 4–5, Class 2 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
200	Real Physical Slot Xmit/Recv	0:20
201	Modified Physical Slot Xmit/Recv	0:20
202	Broadcast Slot Xmit/Recv	0:20
203	Real Physical/Broadcast Slot Xmit/Recv	0:40
204	Modified Physical/Broadcast Slot Xmit/Recv	0:40
205	Foreign Address Rejection Xmit/Recv	1:10
206	Scatter/Gather Operation	0:30

4.5.2.1 Subtest 200, Real Physical Slot Xmit/Recv

Subtest 200 transmits and receives packets sent to the *real* physical address of the controller. (The *real* address slot is the one that is assigned to the board by the manufacturer.)

4.5.2.2 Subtest 201, Modified Physical Slot Xmit/Recv

Subtest 201 transmits and receives packets sent to the modified physical address of the controller. This tests the controller's ability to change the address associated with a given slot, but still transmit and receive Ethernet packets.

4.5.2.3 Subtest 202, Broadcast Slot Xmit/Recv

Subtest 202 transmits and receives packets sent to the Ethernet broadcast address. Each broadcast packet is received by all controllers on the Ethernet.

4.5.2.4 Subtest 203, Real Physical/Broadcast Slot Xmit/Recv

Subtest 203 transmits and receives packets sent to the broadcast and physical address of the controller. The subtest transmits both a broadcast and a physical address packet before attempting to receive any packet. This checks the controller's ability to save packets while waiting for a receive buffer to be supplied by the host computer.

4.5.2.5 Subtest 204, Modified Physical/Broadcast Slot Xmit/Recv

Subtest 204 transmits and receives packets sent to the broadcast and modified physical address of the controller.

4.5.2.6 Subtest 205, Foreign Address Rejection Xmit/Recv

Subtest 205 transmits and receives packets sent to the broadcast and physical address of the controller. Another packet address is also sent out to verify that the controller is rejecting packets destined for other controllers.

4.5.2.7 Subtest 206, Scatter/Gather Operation

Subtest 206 tests for both transmits and receives. The address used is the controller's *real* physical address. This subtest varies the scatter/gather function from 2 blocks to 8 blocks. Each packet is transmitted and received with the same packet split.

4.5.3 Class 3 Subtest

The EXOS/202 Ethernet controller online test consists of subtest that can be executed on an active Ethernet. This allows the test program to execute without changing the configuration of the hardware. These subtests do not transmit or receive the "broadcast" type Ethernet packets.

Table 4–6, Class 3 Subtest

SUBTEST	DESCRIPTION	TIME (min:sec)
300	Real Physical Slot Only Xmit/Recv	0:20

4.5.3.1 Subtest 300, Real Physical Slot Only Xmit/Recv

Subtest 300 only transmits and receives packets on the *real* physical slot. The broadcast slot is disabled. This subtest can be run on an active Ethernet without becoming confused by other packets, unless the board is not rejecting packets properly.

4.5.4 Class 4 Subtest

The EXOS/202 Ethernet controller online test consists of a subtest that transmit and receive packets between two machines on an Ethernet. This requires that Subtest 400 be running on the two machines used in the test. This further requires that no other machines are active on the Ethernet. The test will send a user-entered number of packets and expect to receive the same number of packets from the target machine.

> **NOTE**
>
> The *delni* module must be connected to the board under test with a CONVEX cable 601-150001-200.

Table 4–7, Class 4 Subtest

SUBTEST	DESCRIPTION	TIME (min:sec)
400	Machine to Machine Xmit/Recv	0:20

4.5.4.1 Subtest 400, Machine to Machine Xmit/Recv

Subtest 400 transmits and receives packets over an active Ethernet. The test uses broadcast message addresses initially to determine the address of the other target machine on the Ethernet. The test then sends a user-entered number of packets to the target machine's address and expects to receive an equal number of packets. This subtest cannot be run on an active Ethernet without confusing the other machines on the Ethernet.

4.6 Subtest Completion Messages

When the requested subtest(s) has completed, a subtest(s) completion message(s) is displayed. Sample subtest completion messages are shown in the following figure:

Figure 4–5, Sample Subtest Completion Messages

```
Subtest 100    0:00:02        passed
Subtest 200    0:00:19        passed
Subtest 201    0:00:19        passed
Subtest 202    0:00:19        passed
Subtest 203    0:00:37        passed
Subtest 204    0:00:37        passed
Subtest 205    0:00:02        failed
```

4.7 Subtest Error Messages

Whenever an error is detected, an appropriate error message based on the error reporting flags of the *dshell* is displayed. A sample subtest error message is shown in the following figure:

Figure 4–6, Sample Error Message

```
*****  Mon Feb 22 17:02:27 1988  *****
Test:    dev5500.t  1.1    Class: 1    Subtest: 100 1.2   Count: 1  Error: 0
Failed:  Controller to host access test

Cmd: access registers command, Error: Unknown error 0
```

The following error messages are for all subtests:

`Cmd: <CMD>, Error: <ERR>`

Where <CMD> is equal to one of the following strings:

- nop command to the IOP
- access registers command
- initialize in local memory
- initialize in C1 memory
- send message
- test for pending message
- receive message

And <ERR> is equal to one of the following strings:

> **NOTE**
>
> XX and YY are variables representing hexadecimal status numbers.

- unknown return code 0
- command timeout return code
- unknown command
- Reg acc fail/ self-test fail Expected Status XX, Actual Status YY
- not ready for byte 1 Expected Status XX, Actual Status YY
- not ready for byte 2 Expected Status XX, Actual Status YY
- self-test fail on configure
- accept fail on configure Expected Status XX, Actual Status YY
- configuration error
- command to controller timeout

- `host to exos buffer busy`
- `exos to host buffer busy`
- `message request code invalid`
- `xmit/recv block cnt invalid`
- `statistics block cnt invalid`

A valid message could be the following:

```
Cmd: initialize in local memory, Error: accept fail on configure
Expected Status XX, Actual Status YY
```

or

```
Cmd: send message, Error: xmit/recv block cnt invalid
```

4.7.1 Test End Message

An end message is displayed when the test has completed. A sample end message is shown in the following figure:

Figure 4–7, Sample End Message

```
Frames sent/received with no errors          1300/1200
Frames aborted with excess collisions        0
Frames transmitted with heartbeat absent     0
Frames received with alignment errors        0
Frames received with CRC errors              0
Frames lost                                  0

Test 'dev5500.t' passed
Elapsed time:   0:02:17
:
```

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

S

T

U

V

W

CONVEX VMEbus Ethernet Controller
(*dev5500*) Diagnostics Manual
Document No. 760-003330-000
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)