CONVEX Multibus Terminal Controller (*dev4300*) Diagnostics Manual

Document No. 760-002230-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX Multibus Terminal Controller (dev4300) Diagnostics Manual
Order No. DHW-234
First Edition

Printed in the United States of America

Revision Sheet

CONVEX Multibus Terminal Controller (dev4300) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-002230-000	May 1991	First release. Contains the *dev4300* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.*

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Diagnostics Environment

1.1 Overview 1-1
1.2 Test Program Naming Conventions 1-1
1.2.1 Test Program Categories 1-1
1.2.2 Test Program Types 1-2
1.2.3 Test Program Device Types 1-2
1.2.4 Examples of Test Program Names 1-3

2 EGOS Overview

2.1 Overview 2-1
2.2 Purpose of EGOS for Diagnostic Testing 2-1
2.3 EGOS for the Multibus Interface 2-1
2.4 EGOS for HSP Interface, HSP EGOS 2-1
2.5 EGOS for VME Interface, VIOP EGOS 2-2
2.6 EGOS Position in the Environment 2-2

3 Dshell Overview

3.1 Overview 3-1
3.2 Diagnostic Shell (*dshell*) Overview 3-1
3.3 Syntax Help for *dshell* Commands 3-3

4 Multibus Terminal Controller Test (*dev4300*)

4.1 Overview 4-1
4.2 Prerequisites and Required Equipment 4-1
4.3 Test Invocation 4-1
4.3.1 Test Parameter Menu 4-3
4.3.2 Prompt Explanations 4-5
4.4 Hardware Initialization Sequence 4-7
4.5 Class Descriptions 4-8
4.5.1 Class 1 Subtests 4-8
4.5.1.1 Subtest 100, Reset and Self-test 4-9
4.5.1.2 Subtest 101, Firmware Check 4-9
4.5.1.3 Subtest 102, Timer 4-9
4.5.1.4 Subtest 103, Buffer Reconfiguration 4-10
4.5.2 Class 2 Subtests 4-10
4.5.2.1 Subtest 200, Port Configuration, Internal Loopback 4-11
4.5.2.2 Subtest 201, Input Channel Configuration, Internal Loopback 4-12
4.5.2.3 Subtest 202, Output Channel Configuration, Internal Loopback 4-12
4.5.2.4 Subtest 203, Input Channel Termination Mask, Internal Loopback 4-12
4.5.2.5 Subtest 204, Modem Configuration, Internal Loopback 4-13
4.5.2.6 Subtest 205, Single-character Input On/Off, Internal Loopback 4-13
4.5.2.7 Subtest 206, Block Input, Internal Loopback 4-13
4.5.2.8 Subtest 207, Block Output, Internal Loopback 4-13
4.5.2.9 Subtest 220, Single-character Mode, All Patterns, Internal Loopback 4-13
4.5.2.10 Subtest 221, Single-character Mode, Random Data, Internal Loopback 4-13
4.5.2.11 Subtest 222, Block Mode, All Patterns, Internal Loopback 4-14
4.5.2.12 Subtest 223, Block Mode, Random Data, Internal Loopback 4-14
4.5.2.13 Subtest 224, Block Mode, Various Block Sizes, Internal Loopback 4-14
4.5.3 Class 3 Subtests 4-14

4.5.3.1 Subtest 300, Port Configuration, Physical Loopback 4-15
4.5.3.2 Subtest 304, Modem Configuration, Physical Loopback 4-15
4.5.3.3 Subtest 306, Block Input, Physical Loopback 4-16
4.5.3.4 Subtest 307, Block Output, Physical Loopback 4-16
4.5.3.5 Subtest 320, Single-character Mode, All Patterns, Physical Loopback 4-16
4.5.3.6 Subtest 321, Single-character Mode, Random Data, Physical Loopback 4-16
4.5.3.7 Subtest 322, Block Mode, All Patterns, Physical Loopback 4-16
4.5.3.8 Subtest 323, Block Mode, Random Data, Physical Loopback 4-17
4.5.3.9 Subtest 324, Block Mode, Various Block Sizes, Physical Loopback 4-17
4.5.3.10 Subtest 330, Continuous Blk. Input, Read Buffered Data, Phys. Loopback 4-17
4.5.3.11 Subtest 332, Command Error Code Verification, Physical Loopback 4-17
4.6 Error Messages 4-17

Appendixes

A Reporting Problems

A.1 Overview A-1
A.2 Technical Assistance Center A-1
A.3 The *contact* Utility A-1
A.4 Prerequisites A-1
A.4.1 UUCP Connection A-1
A.4.2 Finding the Program Path Name A-2
A.4.3 Finding the Program Version Number A-2
A.5 Tips on Using the *contact* Utility A-2
A.5.1 Using a *.contact* File A-3
A.5.2 Aborting the Report A-3
A.5.3 Submitting the *dead.report* File A-3
A.5.4 Suspending a Report A-3
A.5.5 Ending a Response A-3
A.5.6 Tilde-Escape Sequences A-4
A.6 Using the *contact* Utility A-4

List of Tables

1–1 Test Program Categories 1-2
1–2 Test Program Types 1-2
1–3 Test Program Device Types 1-3
1–4 Example Test Program Names 1-3
3–1 *dshell* Commands 3-2
4–1 Hardware Requirements 4-1
4–2 Getting Help During Test Parameter Entry 4-3
4–3 *dev4800* Test Classes 4-8
4–4 Class 1 Subtests 4-9
4–5 MTI Nominal Timer Periods 4-10
4–6 Buffer Reconfigurations 4-10
4–7 Class 2 Subtests 4-11
4–8 Class 3 Subtests 4-15
4–9 Command Error Codes 4-18
4–10 USART Error Messages 4-19

4–11 IOP Access Error Messages 4-20
4–12 IOP Command Processing Error Messages 4-21

List of Figures

2–1 EGOS' Position in the Environment 2-3
3–1 Syntax Help for the *loop* Command 3-3
4–1 Test Invocation Sequence 4-2
4–2 Alternate Test Invocation Sequence 4-3
4–3 Test Parameter Menu 4-4
4–4 Sample Test Parameter Summary 4-7

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

Purpose and Intended Audience

This manual explains how to run the *dev4300* diagnostic, which verifies the functionality of a MTI-800A, MTI-1600A, MTI-850, or MTI-1650 asynchronous communications controller attached through a Multibus chassis to an IOP. This document is not a tutorial, but rather a reference for the users of the *dev4300* 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 *dev4300* 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 Terminal Controller Test (*dev4300*)**—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-234.
The document number for this manual is 760-002230-000.

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

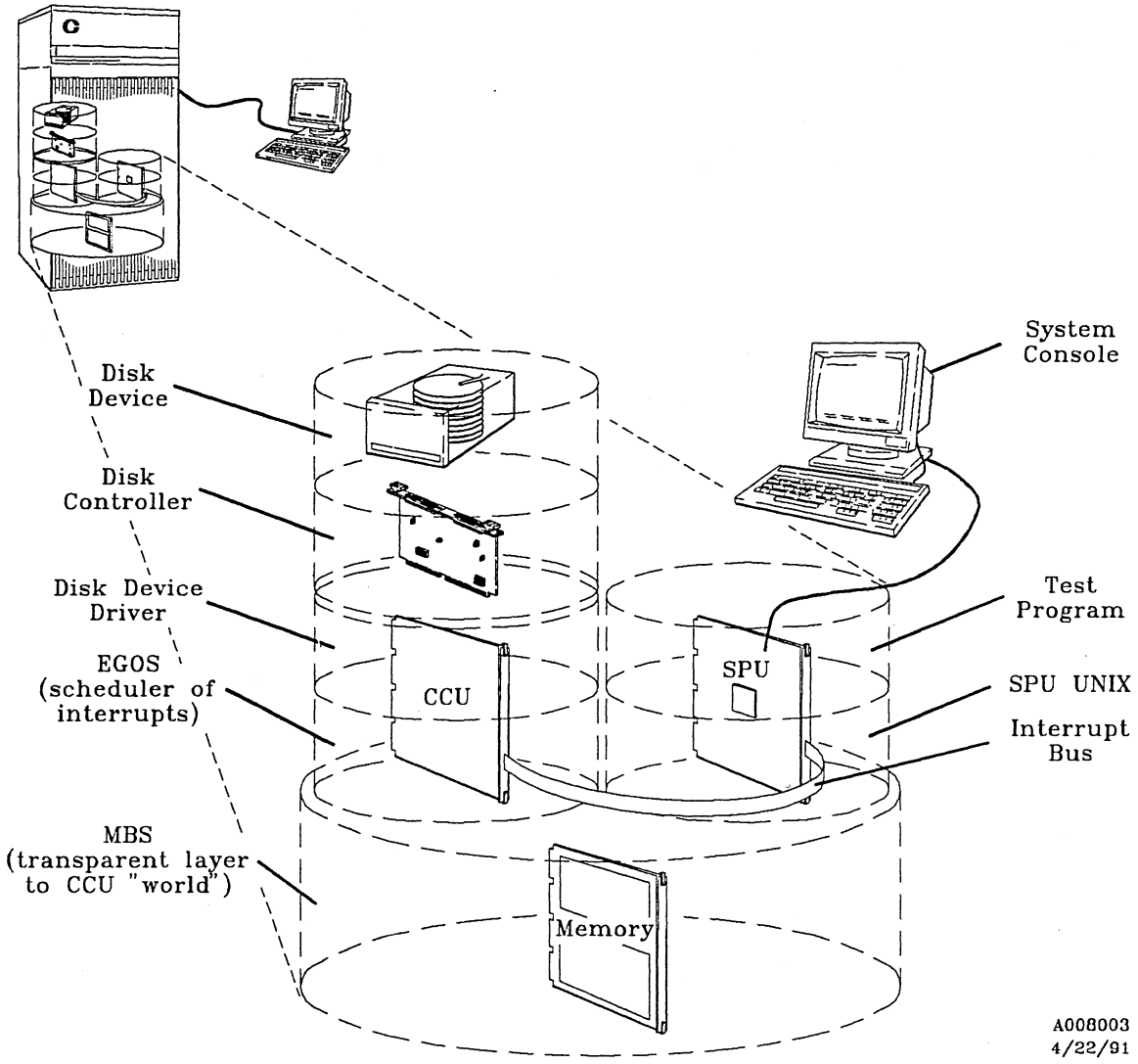
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes 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 Terminal Controller Test (*dev4300*)

4.1 Overview

The *dev4300* test verifies the functionality of a MTI-800A, MTI-1600A, MTI-850, or MTI-1650 asynchronous communications controller attached through a Multibus chassis to an IOP.

4.2 Prerequisites and Required Equipment

In order for this test to run, a Systech controller must be operational. 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).

4.3 Test Invocation

The *dev4300* 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 *dev4300* 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 dev4300 [-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 dev4300** executes all *dev4300* 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 dev4300 [-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
?	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, Test Parameter Menu

```
                    DEFINE 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
    -------   -------  -------       ---     ---

Device 0 = user defined configuration

 1: Device Selection [0]                          (0) ->
 2: IOP [3-7][1]                                  (5) ->
 3: Multibus Chassis [0-3]                        (0) ->
 4: Controller Offset in Multibus [0x0-0xfff]
                                                (0x20) ->
 5: Controller Interrupt [0-7]                    (7) ->
 6: Port count [8,16]                            (16) ->

    Controller Models

     0) MTI-850/1650
     1) MTI-800/1600

 7: Select Model [0-1]                            (0) ->
 8: Port test Mask (hex), bit 2^n for port n
     [0x1-0xffff]                             (0xffff) ->

    Valid baud rate masks are:
    (Or'ed to get mask)
    8000=  50      4000=  75
    2000=  110     1000=  134.5
    0800=  150     0400=  300*
    0200=  600     0100=  1200*
    0080=  1800    0040=  2000
    0020=  2400*   0010=  3600
    0008=  4800    0004=  7200
    0002=  9600*   0001=  19200*

    * is in default mask

 9: Baud Rate Mask (hex) [0x1-0xffff]         (0x523) ->
10: Are test cables installed?  Enter ? for info
     [y,n,?]                                      (y) ->
11: 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

The test parameter prompts are repeated and explained in the following paragraphs.

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

This prompt selects a device for testing 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 selected, the prompts are not displayed.

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

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]
                                                (0x20) ->
```

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

```
Controller Interrupt [0-7]                      (0) ->
```

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

```
Port count [8,16]                               (16) ->
```

Enter the number of ports per panel. The number of ports per panel depends on the controller model.

```
Select Model [0-1]                              (0) ->
```

Enter the model number of the controller under test.

```
Port test Mask (hex), bit 2^n for port n
[0x1-0xffff]                              (0xffff) ->
```

Enter the mask used to determine which pairs to test during loopback tests. The pairs must be adjacent.

```
Baud Rate Mask (hex) [0x1-0xffff]         (0x523) ->
```

Enter the baud rate to be used to test controllers.

```
Are test cables installed?  Enter ? for info
[y,n,?]                                     (y) ->
```

This prompt is displayed only when the tests to be executed include Class 3 subtests. If **n** is entered to this prompt, the following message is displayed when Class 3 subtests execute:

```
Not executed.  Loopback cables not installed.
```

However, if **y** is entered to the prompt, then the subtests execute normally.

```
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. If standard output is directed to a disk file, the test parameter summary is also directed to the file.

Figure 4–4, Sample Test Parameter Summary

```
                TEST PARAMETER SUMMARY

Device Selection                                        : 0
IOP                                                     : 3
Multibus Chassis                                        : 0
Controller Offset in Multibus                           : 0x20
Controller Interrupt                                    : 7
Port Count                                              : 16

         Controller Models

          0) MTI-850/1650
          1) MTI-800/1600

Select Model                                             : 0
Port Test Mask (hex), bit 2^n for port n                : 0xffff

         Valid baud rate masks are:
         (Or'ed to get mask)
          8000=  50                  4000=  75
          2000=  110                 1000=  134.5
          0800=  150                 0400=  300*
          0200=  600                 0100=  1200*
          0080=  1800                0040=  2000
          0020=  2400*               0010=  3600
          0008=  4800                0004=  7200
          0002=  9600*               0001=  19200*

        * is in default mask

Baud Rate Mask (hex)                                     : 0x523
Are test cables installed?  Enter ? for info             : y
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 *dev4300* test contains three classes of tests shown in the following table.

Table 4–3, *dev4300* Test Classes

CLASS	DESCRIPTION
1	Controller basic functional tests
2	Internal loopback tests
3	Physical loopback tests

Where data transmission or modem control testing is to be performed, signal paths between transmitters and receivers are completed by using one of the following methods:

1. Internal loopback on the USARTs
2. Physical loopback between adjacent channels

The first method, internal loopback, which is used in Class 1 and 2 subtests whenever necessary, does not require changing the existing cable connections at the Systech port panel.

> **NOTE**
>
> To run Class 3 subtests, the cables must be disconnected at the ports to be tested and replaced temporarily with test cables, part number 603-030012-200.

Untested commands are those not formally tested in their own subtests. Some of these commands have been used in the construction of subtests for other commands.

Controller commands not formally tested include the following:

- Read USART status register
- Write USART command register
- Abort input
- Abort output
- Suspend output
- Resume output

4.5.1 Class 1 Subtests

Class 1 subtests verify some of the basic functionality of the controller, including, board reset and self-test, firmware level check, timer rate, and internal buffer allocation.

The *dev4300* test contains the following Class 1 subtests, which execute in the times shown:

Table 4–4, Class 1 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
100	Reset and Self-test	:02
101	Firmware Check	:02
102	Timer	:51
103	Buffer Reconfiguration	:14

4.5.1.1 Subtest 100, Reset and Self-test

Subtest 100 ensures that the controller resets properly by issuing a *reset* command and waiting for the controller to become ready.

4.5.1.2 Subtest 101, Firmware Check

Subtest 101 checks that firmware revision 3 or newer is installed in the controller. This is accomplished by executing a *read buffered data* command, which is not available in prerevision 3 firmware.

4.5.1.3 Subtest 102, Timer

Subtest 102 checks the controller timer interrupt rate. An acceptable controller must have at least one rate in the range of 10 to 20 milliseconds. Since the timer interrupt on the controller is not very accurate, the test checks all the timer rates to determine if the controller under test has at lease one rate in the required range. If a valid rate exists, the subtest passes. The actual interrupt rates for the controller under test displays at the completion of the test.

Timer rates are selected by programming the controller with a rate code between 0 and 10 (decimal). Rate codes higher than 10 (decimal) are not used as the interrupt rate is too high.

The nominal timer periods expected depend on the MTI controller model; the time periods are as follows:

Table 4–5, MTI Nominal Timer Periods

RATE CODE	ADVERTISED PERIOD (ms)	1600A NOMINAL PERIOD(ms)	1650 NOMINAL PERIOD (ms)
0	530	290	795
1	350	190	525
2	240	130	360
3	200	110	300
4	175	95	263
5	90	50	135
6	45	25	68
7	20	12	30
8	15	8	23
9	13	7	20
10	10	6	15

4.5.1.4 Subtest 103, Buffer Reconfiguration

Subtest 103 checks the *buffer configuration* command of the controller by picking a channel and configuring one of the sizes from the following table for that channel. This test simultaneously configures all other channels to use all possible remaining buffer space in the controller.

Depending on the port count, the following combinations of legal and illegal buffer sizes (in bytes) are configured:

Table 4–6, Buffer Reconfigurations

16 PORT	8 PORT
32	32
40	40
48	48
56	56
15392	15904
15384	15896
15376	15888
15368	15880
0	0
33	33
15400	16002

The test does not attempt block I/O to verify valid configurations.

4.5.2 Class 2 Subtests

Class 2 subtests verify the operation of the controller, using internal loopback when data transfer is necessary.

Table 4–7, Class 2 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
200	Port Configuration, Internal Loopback	:33
201	Input Channel Configuration, Internal Loopback	:35
202	Output Channel Configuration, Internal Loopback	:05
203	Input Channel Termination Mask, Internal Loopback	:32
204	Modem Configuration, Internal Loopback	:13
205	Single-character Input On/Off, Internal Loopback	:07
206	Block Input, Internal Loopback	:05
207	Block Output, Internal Loopback	:02
220	Single-character Mode, All Patterns, Internal Loopback	:48
221	Single-character Mode, Random Data, Internal Loopback	:51
222	Block Mode, All Patterns, Internal Loopback	:17
223	Block Mode, Random Data, Internal Loopback	:18
224	Block Mode, Various Block Sizes, Internal Loopback	:52

4.5.2.1 Subtest 200, Port Configuration, Internal Loopback

Subtest 200 checks port configuration programming. Essentially, the test programs the USART mode and command registers to the various configurations as indicated in the following paragraphs. All tests are executed in the internal loopback mode on all ports in the port test mask.

Stop Bit Programming

This test programs the three possible stop bit configurations (1, 1.5, and 2 bits) and 8-bit characters. It passes a block of data at 9600 baud to verify transmission and reception.

Parity Bit Programming

This test programs the three possible parity configurations (none, even, and odd), one stop bit, and 8-bit characters. It passes a block of data at 9600 baud to verify transmission and reception.

Character Length Programming

This test programs the four possible character lengths (5 through 8 bits) and one stop bit. It passes a block of data at 9600 baud to verify transmission and reception.

Baud Rate Programming

This test programs each baud rate specified in the baud rate mask, one stop bit, and 8-bit characters. It passes a block of data at each baud rate to verify transmission and reception.

RTS Assertion Test

Request to Send (RTS) is looped backed to Clear to Send (CTS) by the internal loopback mechanism. The test transmits a block of data with RTS unasserted. Completion should take place only after the USART is reprogrammed to assert RTS.

USART Force Break and Framing Error Test

This test programs a forced break on each port to be tested. It verifies a single null with framing error is received.

DTR Assertion Test

During testing, Data Terminal Ready (DTR) is looped back to Data Carrier Detect (DCD) by the internal loopback mechanism. The test transmits a block of data with DTR unasserted. It verifies completion of output only; input does not complete because DCD is not seen.

USART Transmitter Enable Test

This test programs the USARTs with the transmitter off. Data is looped around, with no response, until the USARTs are reprogrammed to turn on the transmitters.

Four tests (Stop Bit, Parity Bit, Character Length, and Baud Rate) are complemented by Subtest 300, Port Configuration Test which passes data between separate USARTs.

The USART receiver enable test cannot be made. Apparently, the controller firmware turns on the receiver whenever a *receive* command is executed.

4.5.2.2 Subtest 201, Input Channel Configuration, Internal Loopback

Subtest 201 checks input channel programming on all ports in the port test mask using internal loopback mode. It tests input block and single-character permissions. Termination via ASCII nonprintables and selectable character set is tested. The test does not verify firmware echo and CONTROL-S/CONTROL-Q protocol capabilities.

4.5.2.3 Subtest 202, Output Channel Configuration, Internal Loopback

Subtest 202 verifies output channel programming, in internal loopback mode, for each port in the port test mask. The block output permission bit is tested to ensure that block output is permitted when programmed and not permitted when not programmed.

4.5.2.4 Subtest 203, Input Channel Termination Mask, Internal Loopback

Subtest 203 verifies the *input channel termination mask* command. The test also checks the command to ensure that it overrides any termination character selection in the input channel configuration. All tests are conducted in internal loopback mode.

The bytes of the termination masks for each channel are written to sequential numbers, beginning with zero. Then, the test checks the entire mask for proper action of each bit on each input channel.

4.5.2.5 Subtest 204, Modem Configuration, Internal Loopback

Subtest 204 checks modem change response capability, in internal loopback mode, on all ports in the port test mask. The controller is programmed to return a response when DCD or DSR change, and the response capability for DCD is tested. It ensures that no response is returned when it is not programmed.

4.5.2.6 Subtest 205, Single-character Input On/Off, Internal Loopback

Subtest 205 checks the individual commands to enable and disable single-character input on all ports in the port test mask. Internal loopback mode is used for all testing.

4.5.2.7 Subtest 206, Block Input, Internal Loopback

Subtest 206 checks block input termination caused by transmission error and host abort, using internal loopback mode. The test uses *force break* to test transmission error. Host abort is tested by setting up block input and terminating it while pending. Tests are conducted on all ports in the port test mask. (Termination caused by *DMA* error is not tested.)

4.5.2.8 Subtest 207, Block Output, Internal Loopback

Subtest 207 checks block output termination caused by host abort, using internal loopback mode. Testing is performed by initiating block output at a very low baud rate and then terminating it before it can complete. Tests are conducted on all ports in the test mask. The test does not include *DMA* error response.

4.5.2.9 Subtest 220, Single-character Mode, All Patterns, Internal Loopback

Subtest 220 performs simultaneous input/output in single-character mode on all tested ports, using internal loopback mode. The test is conducted at all selected baud rates. All possible data patterns are transmitted over each tested port. The test pattern for each port consists of sequentially increasing byte values, modulo 256, beginning with the value of the port number.

The test performs 16 sequential single-character output operations on each output channel. It retrieves 16 single-character responses from each input channel. The process repeats until all 256 patterns are transferred over each channel.

4.5.2.10 Subtest 221, Single-character Mode, Random Data, Internal Loopback

Subtest 221 performs simultaneous input/output in single-character mode on all tested ports, using random data, in internal loopback mode. The test is conducted at each selected baud rate.

It performs 16 sequential single-character output operations on each output channel and receives 16 single-character responses for each input channel. The process repeats until 256 bytes are transferred over each channel.

Random data is generated using *rand* and *srand*; the seed value is 1309073. The test reinitializes the seed for each baud rate.

4.5.2.11 Subtest 222, Block Mode, All Patterns, Internal Loopback

Subtest 222 performs simultaneous block mode input/output on all channels to be tested, using internal loopback. The maximum block size used is limited by firmware to 252 bytes. The test is conducted at all selected baud rates.

This subtest passes all possible data patterns through each internally looped back port to be tested. The test pattern for each port is defined to be the sequentially increasing byte values, modulo 256, beginning with the port number.

4.5.2.12 Subtest 223, Block Mode, Random Data, Internal Loopback

Subtest 223 performs simultaneous block mode input/output on all internally looped back ports to be tested at all selected baud rates. The data block size is arbitrarily fixed at 252 bytes.

This subtest passes pseudo-random data patterns over each port. Random data is generated using the *rand* and *srand* function of the C library; the seed value is 1120597. The test reinitializes the seed for each baud rate.

4.5.2.13 Subtest 224, Block Mode, Various Block Sizes, Internal Loopback

Subtest 224 performs simultaneous block mode input/output on all ports to be tested, with variable-size blocks, using internal loopback. The test is conducted at all selected baud rates. The data test pattern is random, using a seed value of 137; the test reinitializes the seed at each baud rate. The test uses block sizes 1 through 11, 40, 69, 98, 173, and 252 (the firmware maximum).

4.5.3 Class 3 Subtests

Class 3 subtests verify the correct operation of the controller, using physical loopback cables on ports to be tested.

Table 4–8, Class 3 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
300	Port Configuration, Physical Loopback	:12
304	Modem Configuration, Physical Loopback	:13
306	Block Input, Physical Loopback	:02
307	Block Output, Physical Loopback	:03
320	Single-character Mode, All Patterns, Physical Loopback	:48
321	Single-character Mode, Random Data, Physical Loopback	:51
322	Block Mode, All Patterns, Physical Loopback	:18
323	Block Mode, Random Data, Physical Loopback	:17
324	Block Mode, Various Block Sizes, Physical Loopback	:52
330	Continuous Block Input, Read Buffered Data, Physical Loopback	:54
332	Command Error Code Verification, Physical Loopback	:37

4.5.3.1 Subtest 300, Port Configuration, Physical Loopback

Subtest 300 checks portions of the *port configuration* command, using physical loopback mode. All port pairs in the port test mask are tested. Subtest 300 performs the following tests:

Stop Bit Programming

This test programs the 3 possible stop-bit configurations (1, 1.5, and 2 bits) and 8-bit characters. To verify transmission and reception, it passes a block of data at 9600 baud.

Parity Bit Programming

This test programs the three possible parity configurations (none, even, and odd), one stop bit, and 8-bit characters. It passes a block of data at 9600 baud to verify transmission and reception.

Character Length Programming

This test programs the four possible character lengths (5 through 8 bits) and one stop bit. To verify transmission and reception, the test passes a block of data at 9600 baud.

Baud Rate Programming

This test programs each baud rate specified in the baud rate mask, one stop bit, and 8-bit characters. To verify transmission and reception, it passes a block of data at each baud rate.

4.5.3.2 Subtest 304, Modem Configuration, Physical Loopback

Subtest 304 checks modem change response capability in physical loopback mode on all port pairs in the port test mask. The controller is programmed to return a response when DCD or DSR change, and the response capability for DCD is tested. It assures that no response is returned when it is not programmed. The test does not include DSR response.

4.5.3.3 Subtest 306, Block Input, Physical Loopback

Subtest 306 checks block input termination caused by host abort, using physical loopback mode on all port pairs in the port test mask. It tests host abort by setting up block input and terminating it while pending. The test does not check termination caused by transmission error or *DMA* error.

4.5.3.4 Subtest 307, Block Output, Physical Loopback

Subtest 307 checks block output termination caused by host abort, using physical loopback mode on all port pairs in the port test mask. Testing is performed by initiating block output at a very low baud rate and then terminating it before it can complete. It does not test *DMA* error response.

4.5.3.5 Subtest 320, Single-character Mode, All Patterns, Physical Loopback

Subtest 320 performs simultaneous input/output in single-character mode on all port pairs to be tested using physical loopback mode. The test performs at all selected baud rates. It transmits all possible data patterns over each tested pair. The test pattern consists of sequentially increasing byte values, modulo 256, beginning with the value of the transmitter port number.

The test performs 16 sequential single-character output operations on each output channel; it receives 16 single-character responses for each input channel. The process repeats until all 256 patterns have been transmitted.

4.5.3.6 Subtest 321, Single-character Mode, Random Data, Physical Loopback

Subtest 321 performs simultaneous input/output in single-character mode on all tested port pairs, using random data and physical loopback mode. Random data is generated using *rand* and *srand*; the kernel value is 1309073. Tests are conducted at each selected baud rate.

It performs 16 sequential single-character output operations on each output channel, and it receives 16 single-character responses for each input channel. The process repeats until 256 bytes have been transferred over each channel.

4.5.3.7 Subtest 322, Block Mode, All Patterns, Physical Loopback

Subtest 322 performs simultaneous block mode input/output on all channel pairs to be tested, using physical loopback cables. Firmware limits the maximum block size to 252 bytes. The test is conducted at all selected baud rates.

This subtest passes all possible data patterns over each channel pair to be tested. The test pattern for each port is defined to be the sequentially increasing byte values, modulo 256, beginning with the port number of the transmitter.

4.5.3.8 Subtest 323, Block Mode, Random Data, Physical Loopback

Subtest 323 performs simultaneous block mode input/output on all channel pairs to be tested, using physical loopback cables. The data block size is arbitrarily fixed at 252 bytes. The test is conducted at each selected baud rate.

This subtest passes pseudo-random data patterns over each channel pair. It generates data bytes using the *rand/srand* function of the C library, using a seed value of 1120597. The seed is reinitialized for each baud rate.

4.5.3.9 Subtest 324, Block Mode, Various Block Sizes, Physical Loopback

Subtest 324 performs simultaneous block mode input/output on all port pairs to be tested, with variable-size blocks, using physical loopback. The test performs at all selected baud rates. The data test pattern is random, using a seed value of 137, which is reinitialized at each baud rate. The test uses block sizes 1 through 11, 40, 69, 98, 173, and 252 (the firmware maximum).

4.5.3.10 Subtest 330, Continuous Blk. Input, Read Buffered Data, Phys. Loopback

Subtest 330 performs continuous block input on each of the looped back ports. Each port is tested for its ability to execute the continuous block input at all selected baud rates. The input is terminated by issuing a *read buffered data* command, and the termination cause is verified as well as the data transferred before termination. The amount of data transferred varies according to the baud rate.

4.5.3.11 Subtest 332, Command Error Code Verification, Physical Loopback

Subtest 332 verifies the firmware's ability to recognize command errors and generate the appropriate error codes. The command errors attempted are: buffer length error, buffer not a multiple of eight (bytes), buffer size greater than 32K (bytes), undefined command nibble, command too short, command too long, channel already configured, input request too big, and output request too big.

4.6 Error Messages

Block size error, port *dd*, **expected** *dd*, **actual** *dd*

A data block of the wrong size was received or transmitted. Expected and actual sizes are given.

Command Error Code *0xnn 0xnn*

<message text detailing command error>

Command errors normally result from giving an illegal command to the controller. However, it is possible that controller hardware errors or firmware changes could cause some of these errors. Also, they can be caused by test program errors. Some command errors are:

Table 4–9, Command Error Codes

TEXT	MEANING
INPUT REQUEST TOO BIG FOR PORT dd	Possible firmware change
OUTPUT BUFFER FULL FOR PORT dd	Possible firmware change
OUTPUT REQUEST TOO BIG FOR PORT dd	Possible firmware change
PORT dd OFFLINE (SELF-TEST ERROR)	Hardware error
UNKNOWN ERROR STATUS I0xnn	Program/hardware error

Data error, port *dd*, **expected** *0xnn*, **actual** *0xnn*

Data verification failed. Expected and actual byte values are given.

Erroneous Block I/O Termination Code, Port *dd*

<message detailing termination code>

The termination cause is incorrect. The termination code displays one of the following messages:

```
TERMINATION CHARACTER
TRANSMISSION ERROR
DMA TO NON-MEMORY
BLOCK COUNT COMPLETE
HOST ABORTED COMMAND
```

Expected and actual values of the termination cause are printed.

Erroneous USART status code

<message detailing status code>

Improper USART status in controller response. The detail of status is:

USART Status Code *0xnn*

<message>

Expected and actual values of the USART status register are printed. The following messages may appear when a USART error occurs:

Table 4–10, USART Error Messages

TEXT	MEANING
DSR SENSED	Data set ready sensed
DCD SENSED	Data carrier detected
FRAMING ERROR	Framing error occurred
OVERRUN	Overrun in USART
PARITY ERROR	Parity error detected

Error allocating SPU window map: UNIX error message.

There is a system problem allocating window maps. Contact the Technical Assistance Center.

Error in load_iop
<message detailing error>
An error has occurred while bootstrapping the IOP. Make sure that the IOP program image file *dev4300.x00* is available and readable. It must reside either in the current directory or */mnt/test/*. Refer to "Error in setup_iop" for message detail.

Error in recv_iop
<message detailing error>
An error has occurred using the SPU/IOP interface to wait for a signal from the IOP indicating that the command is finished. Refer to "Error in setup_iop" for message detail.

Error in send_iop
<message detailing error>
An error has occurred using the SPU/IOP interface to signal the IOP that a command is ready to execute. Refer to "Error in setup_iop" for message detail.

Error in setup_iop
<message detailing error>
An error occurred when preparing to access the IOP. When this error occurs, the message may be:

Table 4–11, IOP Access Error Messages

TEXT	MEANING
HARD ERROR	Hardware error
IOP BUS ERROR	Controller address error
IOP CACHE ERROR	IOP hardware error
IOP PBUS ERROR	IOP hardware error
MMIO ERROR	Main memory error
MULTIBUS ERROR	Hardware error
TIMEOUT	Possible hardware error

Error in start_iop
<message detailing error>
An error occurred while starting the IOP. Refer to "Error in setup_iop" for message detail.

Error opening device
<message detailing error>
An error occurred while setting up the IOP and device driver to access the requested device. Refer to "Function status" message for more information.

Error opening SPU window to main memory: UNIX error message.
There is a system problem getting access to window map. Contact the Technical Assistance Center.

Exception in mmalloc_int. Is main memory initialized?
There is a problem accessing main memory. Assure that main memory is present; initialize, if necessary, using *mminit*. If problems persist, contact the Technical Assistance Center.

Function status: message
<possible port and channel indication>
There is a problem in IOP command processing. The following messages may appear:

Table 4–12, IOP Command Processing Error Messages

TEXT	MEANING
OK	No error
MAIN MEM ALLOC ERROR	Main mem allocation error
IOP NOT OPENED	Open operation failed
INVALID RESPONSE STRING	Controller bad response
CONTROLLER WON'T GO READY	Controller malfunction
NO RESPONSE AVAILABLE	Controller malfunction
ILLEGAL RESPONSE	Controller malfunction
PARTIAL RESPONSE	Controller malfunction
COMMAND ERROR STUCK ON	Controller malfunction
TIMEOUT	Possible controller malfunction
TEST FAILED	Test failed
UNACK'D PBUS INTR	Pbus error
CONTROLLER NOT READY	Controller malfunction

Main memory allocation error
There is a problem allocating main memory. Be sure that main memory is present and initialized. If problems persist, contact the Technical Assistance Center.

Port *dd* unpaired, not tested
This warning appears when running subtests which require adjacent ports to be attached together with special back-to-back cables. When the port test mask (test parameter number 7) does not specify both or neither port in an adjacent pair, the specified port is eliminated from the test.

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

A

Alaska, reporting problems from, telephone number for x
Associated documents, how to order x
Associated documents, listed ix

B

Baud rate programming 4-11, 4-15
Block input, internal loopback (subtest 206) 4-13
Block input, physical loopback (subtest 306) 4-16
Block mode, all patterns, internal loopback (subtest 222) 4-14
Block mode, all patterns, physical loopback (subtest 322) 4-16
Block mode, random data, internal loopback (subtest 223) 4-14
Block mode, random data, physical loopback (subtest 323) 4-17
Block mode, various block sizes, internal loopback (subtest 224) 4-14
Block mode, various block sizes, internal loopback (subtest 324) 4-17
Block output, internal loopback (subtest 207) 4-13
Block output, physical loopback (subtest 307) 4-16
Buffer reconfiguration (subtest 103) 4-10

C

C Programming Language ix
Canada, reporting problems from, telephone number for x
cattypedevnn.suffix 1-1
Cautions, described ix
Character length programming 4-11, 4-15
Command error code verification, physical loopback (subtest 332) 4-17
Command scripts, user-created 3-1
contact, aborting the report A-3, A-6
contact, editing the report A-6
contact, ending a response A-3
contact, ending the report A-6
.contact file, skipping first prompt by using A-3
contact, including files in your report A-5
contact, invoking A-1, A-4
contact, prerequisites A-1
contact, prompts A-4
contact, prompts, step-by-step discussion of A-4
contact, report, suspending A-3
contact, reporting problems A-1
contact, restrictions, on tilde-escape sequences A-5
contact, reviewing the report A-6
contact, skipping first prompt by using a *.contact* file A-3
contact, submitting *dead.report* file A-3
contact, submitting the report A-6
contact, tilde-escape sequences A-4
contact, tips on using A-2
Continuous block input, read buffered data, physical loopback (subtest 330) 4-17
Controller basic functional tests 4-8
CONVEX, address, for ordering documents x
CONVEX Diagnostic Utilities Manual, C120 ix
CONVEX Diagnostic Utilities Manual, (C200 Series) ix
CONVEX Processor Operation Guide ix
CONVEX UNIX Tutorial Papers ix
CPU 1-1
CPU, *cpu*, test program for 1-2
cpu, test category 1-2

D

dead.report file, submitting A-3
dead.report file, using *-r* option to submit A-3
dev, test category 1-2
Dev4300 (Systech MTI-1600A/terminal test) 4-1
Dev4300 (test parameter menu) 4-3
Devices, *dev* for 1-1
Devices, test programs for, table 1-3
Devices, types, listed 1-2
Diagnostic environment, overview 1-1
Diagnostic shell. *See dshell*
Diagnostics, selecting 3-1
Disks 1-2
Disks, device, test program for 1-3
dshell, introduction 3-1
dshell, overview 3-1
DTR assertion test 4-12

E

Error messages 4-17
Error messages, selecting 3-1
error reporting A-1

F

Files, test outputs to 3-1
Firmware check (subtest 101) 4-9
Functional tests, controller 4-8

H

Hawaii, reporting problems from, telephone number for x
Help-for *dev4300* prompts 4-3

I

Input channel configuration, internal loopback (subtest 201) 4-12
Input channel termination mask, internal loopback (subtest 203) 4-12
Internal loopback tests 4-10
I/O, subsystem test, *io* for 1-2
I/O system, test program categories for 1-1
io, test category 1-2

K

Kernel, hardware tests 1-2
Kernel, hardware tests, program for 1-3

M

mem, test category 1-2
Memory, subsystem test, *mem* for 1-2
Memory system, test program name for 1-1
Modem configuration, internal loopback (subtest 204) 4-13
Modem configuration, physical loopback (subtest 304) 4-15
Modem control testing 4-8
Multibus Systech terminal test 4-1

N

Networks 1-2
Networks, device, test program for 1-3
Notational conventions, discussed ix
Notes, described ix

O

Offline tests 1-2
Offline tests, functional, program for 1-3
Online tests 1-2
Online tests, functional, program for 1-3
Output channel configuration, internal loopback (subtest 202) 4-12
Overview, diagnostic environment 1-1
Overview, *dshell* 3-1

P

Parity bit programming 4-11, 4-15
Peripheral devices, test program name for 1-1
Peripherals, *dev*, test program for 1-2
Physical loopback tests 4-14
Port configuration, physical loopback (subtest 300) 4-15
Port configuration subtest, internal loopback (subtest 200) 4-11
Printers 1-2
Printers, device, test program for 1-3
problems, reporting, overview A-1

R

Reader's Forum x
Reporting problems x
Reset and self-test (subtest 100) 4-9
Revision sheet 3
RTS assertion test 4-12

S

Screens, test outputs to 3-1
Scripts, predefined 3-1
Self-tests 1-2
Self-tests, test program for 1-3
Service Processor Unit. *See* SPU
Single-character input on/off, internal loopback (subtest 205) 4-13
Single-character mode, all patterns, internal loopback (subtest 220) 4-13
Single-character mode, all patterns, physical loopback (subtest 320) 4-16
Single-character mode, random data, internal loopback (subtest 221) 4-13
Single-character mode, random data, physical loopback (subtest 321) 4-16
SP2, subsystem test, *spu* for 1-2
SP2, *.t* programs and 1-1
SP2, test program name for 1-1
SPU, *dshell* and, introduction 3-1
spu, test category 1-2
Standalone tests 1-2
Stop bit programming 4-11, 4-15
Subsystems, *cat* for 1-1
Subtests 4-8

T

.t 1-1
TAC, reporting problems to x
TAC (Technical Assistance Center), problems, reporting to A-1
Tape units 1-2
Tape units, test program for 1-3
Technical Assistance Center (TAC), problems, reporting to A-1
Technical assistance, discussed x
Terminals 1-2
Terminals, test program for 1-3
Test parameter menu 4-3
Test programs, categories 1-1
Test programs, categories, table 1-2
Test programs, device types 1-2
Test programs, naming conventions 1-1
Test programs, types 1-2
Test programs, types, table 1-2, 1-3
Tests, options, selecting 3-1
Tests, output, selecting 3-1
tilde-escape sequences A-4
tilde-escape sequences, restrictions on use A-5
Timer subtest (subtest 102) 4-9
Trouble reports x
trouble reports A-1

U

UNIX-to-UNIX Communication Protocol A-1
UNIX-to-UNIX copy command, *uucp* A-1
USART force break and framing error test 4-12
USART transmitter enable test 4-12
User interface 4-3
UUCP, connection to TAC A-1
uucp, UNIX-to-UNIX copy command A-1

V

vers, program version number found by using A-2

W

Warnings, described ix
whence, program path name found by using A-2
which, program path name found by using A-2

CONVEX Multibus Terminal Controller
(*dev4300*) Diagnostics Manual
Document No. 760-002230-000
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)

CONVEX Multibus Terminal Controller
(*dev4300*) Diagnostics Manual
Document No. 760-002230-000
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:

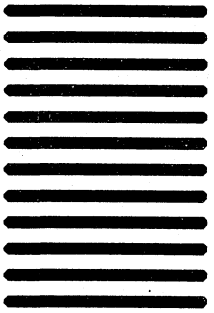
Location	Phone Number
From all locations in continental U.S.	1(800)952-0379
From locations in Alaska & Hawaii	1(214)497-4379
From locations in Canada	1(800)345-2384
From all other locations	Contact nearest CONVEX office

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)