CONVEX Multibus X.25 Controller (*dev4540*) Diagnostics Manual

Document No. 760-002730-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX Multibus X.25 Controller (dev4540) Diagnostics Manual
Order No. DHW-239
First Edition

Printed in the United States of America

Revision Sheet

CONVEX Multibus X.25 Controller (dev4540) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-002730-000	May 1991	First release. Contains the *dev4540* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.*

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

Appendixes

A Reporting Problems

List of Tables

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

Purpose and Intended Audience

This manual explains how to run the *dev4540* diagnostic, which verifies that the Multibus X.25 Controller operates properly in the CONVEX Multibus I/O environment. This document is not a tutorial, but rather a reference for the users of the *dev4540* 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 *dev4540* 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 X.25 Controller Test (*dev4540*)**—Describes how to operate the diagnostic, including prerequisites, test invocation, hardware initialization sequence, X.25 Controller EPROM self-tests, 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-239.
The document number for this manual is 760-002730-000.

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

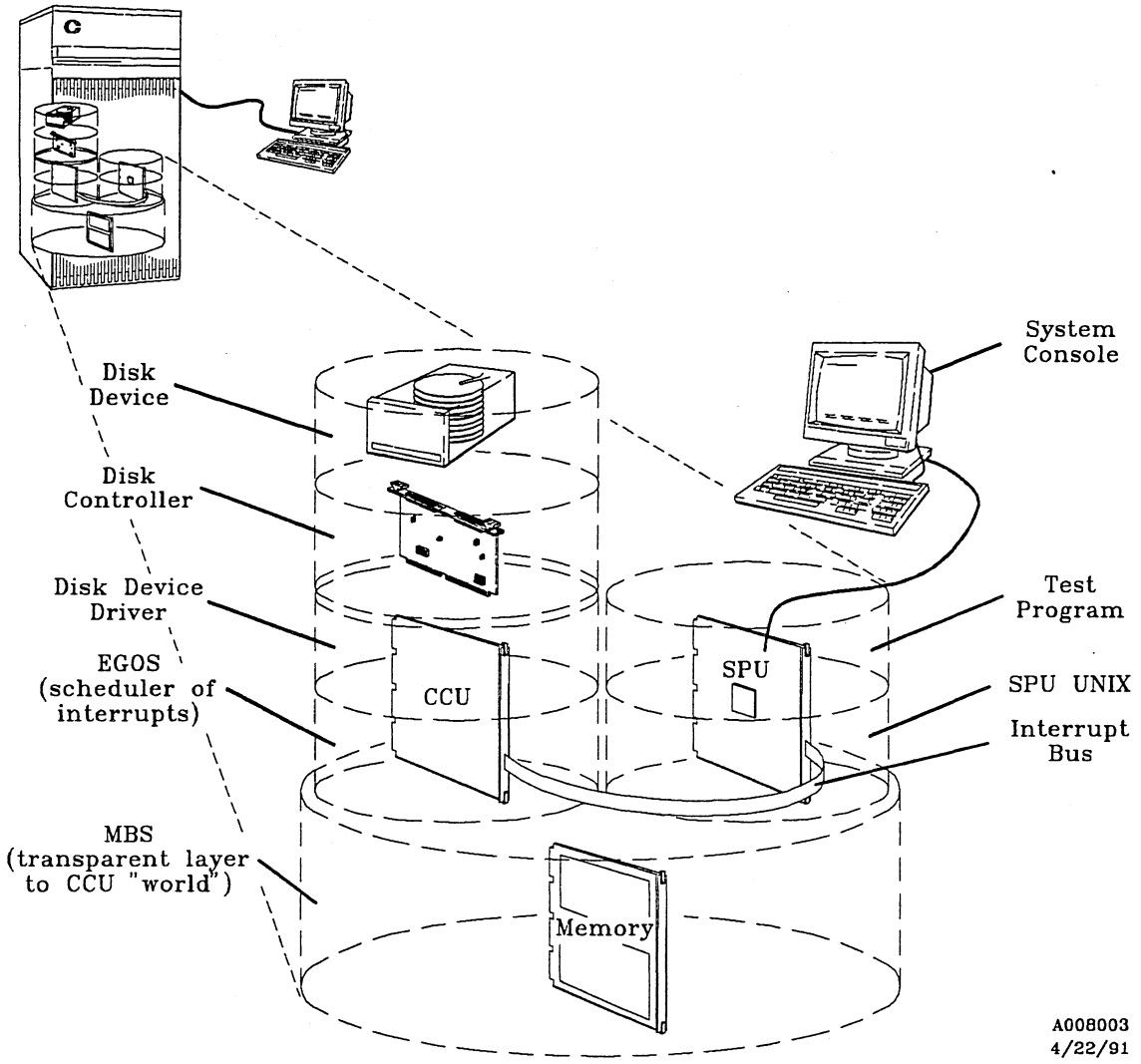
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes 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 X.25 Controller Test (*dev4540*)

4.1 Overview

The *dev4540* test is a functional test for the CONVEX Multibus X.25 controller. This test attempts to verify that the Multibus X.25 Controller operates properly in the CONVEX Multibus I/O environment.

Specifically, *dev4540* is designed to accomplish the following:

- Verify the functional ability of the Multibus X.25 Controller to operate in the CONVEX Multibus I/O environment. This includes main memory access and interrupt generation. g
- Verify that the Multibus X.25 Controller can detect parity errors in the controller's RAM. g
- Verify that the Multibus X.25 Controller can transmit and receive data through the on-board Serial Communications Controllers (SCC) via the Direct Memory Access (DMA) controllers. g
- Verify that the cable interface and external communications modules are working properly (requires external loopback cables; refer to the "Prerequisites and Required Equipment" section in this chapter for the CONVEX part number).

Channel Control Unit (CCU) communications use the Event Governed Operating System (EGOS) and the Message Based System (MBS) used by ConvexOS. The intent is to test the communication paths that are used in a normal operating environment.

4.2 Related Documents

This test description is intended as a reference for users who are familiar with the SBE, Inc. COM-4 board and the operation of the CONVEX I/O system. Additional information on the SBE, Inc. COM-4 board can be found in the *ModulasTen SBE/COM-4 User's Reference Manual* from SBE, Inc.

> **NOTE**
>
> The Multibus X.25 Controller is a modified version of the COM-4 board. Within this test description, COM-4 refers to the Multibus X.25 Controller. Also, any prompts or error messages using COM-4 refer to the Multibus X.25 Controller.

4.3 Prerequisites and Required Equipment

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

Table 4–1, Hardware Requirements

C1, C120	C200 Series
MCU	Memory System[1]
MAU	CPX
SPU	SP2
MIOP	MIOP
MBCU	PIA
Multibus X.25 Controller	MBCU
	Multibus X.25 Controller

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

Some subtests (301 and 401) require loopback cables to execute. The following table contains the Serial Communications Interface (SCI) module and the corresponding part numbers for their required loopback cables:

Table 4–2, Required Loopback Cables

SCI Type	CONVEX Loopback Cable
RS-232	601-040001-200
RS-449	601-080001-200
V.35	601-060001-200

4.4 Test Invocation

The *dev4540* 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 *dev4540* test, use the procedure shown in the following figure. All responses in **boldface** are entered by the user.

> **NOTE**
>
> Use the following test invocation sequence for the initial invocation of *dev4540* or for when the state of the machine is unknown. Also, the following invocation sequence should be used if any hard errors have occurred since the last system initialization.

Figure 4–1, Initial Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
:test dev4540 [-c [class number(s)]] [-s [subtest number(s)]] [-option] [-f FILE] [+>filename] RETURN
```

In the previous figure, 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.

> **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 dev4540** executes all applicable *dev4540* subtests sequentially. To execute a specific class(es) of subtest(s) or one or more individual subtests, use the **-c** or **-s** options during test invocation, respectively. Detailed information for using these options can be found in the "Dshell Overview" chapter of this manual.

The following list defines the dash options ([-*option*]):

> **NOTE**
>
> In the following list of options, the **-b, -d, -e, -i, -l,** and **-o** options either override the settings specified in the parameter save file (when the test is invoked with the **-q** option or with the **dev4540x** invocation sequence) or the defaults for the prompts in the "Test Parameter Menu."

-V Print the version (build date, or last modification) of this test

-b *rate* Set baud rate for loopback test to *rate*

-d *flgs* Set debug flags to *flgs*

-e *act* Set Multibus X.25 Controller error action to *act*

-f *FILE* Use *FILE* as the parameter save file. If this option is omitted, the parameter file used is */tmp/dev4540.tmp*

-i *iter* Set iteration count to *iter*

-l *line* Select Multibus X.25 Controller line(s) to test

-n Do not actually execute any subtest (used to create a parameter file without actually executing any subtest or display parameter file information with **-q**)

-o *opts*	Set external connections to *opts*
-q	Quick start-up (use previous parameters from last test invocation — same as **dev4540x**)
-t	Execute all applicable subtests sequentially (override parameter file subtest list)
-v	Set verbose flag (detailed information printed)
-x *file*	Use alternate Multibus X.25 Controller test *file*

The [+>*filename*] option allows the test results to be appended to *filename*.

> **NOTE**
>
> The following alternate test invocation procedure is optimal when invoking *dev4540* multiple times. Using this invocation sequence insures that the test is invoked and executed with all the setup parameters supplied when the test was last executed with the initial invocation sequence.

Figure 4–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
:test dev4540x [-c [class number(s)]] [-s [subtest number(s)]] [-option] [-f FILE] [+>filename] RETURN
```

The only difference in this alternate invocation sequence is the **x** after **dev4540**. When invoking *dev4540* in this manner, no prompts are displayed. The diagnostic obtains all prompt information from the parameter file created when the initial invocation sequence was performed. Also note that **mminit -s** and **sysreset** are only required if the state of the machine is unknown or if hard errors have occurred since the last system initialization.

4.4.1 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 appear sequentially on the screen, one line at a time. The figure 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.

Figure 4–3, Test Parameter Menu

```
Test 'dev4540.t'              Mon Nov 13 15:27:03 1989

                        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
        ?       Provides specific help where available

 1: Select ioconfig file [<filepath>,?]            (/ioconfig) -> RETURN

      PERIPHERAL CONFIGURATION DATA
     CCU    Chassis   Type     CSR    Int
    -------- ------- -------- ------ ---
 1) iop  4      0    SCI-001  0x08c0  7
 2) iop  4      0    SCI-001  0x08c2  6
 3) iop  4      1    SCI-001  0x08c0  7
 4) iop  4      1    SCI-001  0x08c2  6

                      *** Enter 0 for manual configuration ***

 2: Ioconfig File Unit to Test [1-4,0,?]                  (1) -> 0
 3: Multibus IOP Number [3-7,?]                           (4) -> RETURN
 4: Multibus Chassis Number [0-1,?]                       (0) -> RETURN
 5: Multibus CSR of the Controller [0x0-0xffe,?]      (0x8c0) -> RETURN
 6: Multibus Interrupt Level of the Controller [0-7,?]    (7) -> RETURN
 7: Use Defaults for Remaining Parameters [y,n,?]         (y) -> n
 8: RAM Pattern Subtest: pattern # 1 [<pattern>,?]
                                    (Use default patterns) -> 01234567
 9: RAM Pattern Subtest: pattern # 2 [<pattern>,?]
                                    (Use default patterns) -> 89abcdef
10: RAM Pattern Subtest: pattern # 3 [<pattern>,?]
                                    (Use default patterns) -> RETURN

       ** External Connections (bit mapped) **
   0x0001: Loopback jumper is installed
   0x0002: Transmit clock source (1=DCE,0=DTE)
   0x0004: Modem is attached to connector

11: Select External Connections [0x0-0x7,?]             (0x2) -> RETURN
12: Select Serial Communication Lines [0x1-0xf,?]       (0xf) -> RETURN
13: Baud Rate for Loopback Subtests [110-73728,?]     (57600) -> RETURN
14: Number of Iterations for Each Subtest [1-2147483647,?]
                                                         (1) -> RETURN
```

Figure 4–3, Test Parameter Menu (continued)

```
        ** Debug Mode Flags (bit mapped) **
    0x0001: Print messages sent to COM-4
    0x0002: Print messages received from COM-4
    0x0004: Pause before message sent to COM-4
    0x0008: Pause after message received from COM-4
    0x0010: Print MBS messages sent to CCU
    0x0020: Print MBS messages received from CCU
    0x0040: Pause before MBS message sent to CCU
    0x0080: Pause after MBS message received from CCU
    0x0100: Print extra informative messages
    0x0200: Don't reset COM-4 after subtest is aborted
    0x0400: Don't disable SPU interrupts when done
    0x0800: Simulate all MBS messages (don't send them)

15: Select Debug Mode [0x0-0xfff,?]                    (0x0) -> RETURN

        ** Subtest Error Actions **
    1:  Silently ignore subtest errors
    2:  Pause for subtest errors
    3:  Abort subtest on first error

16: Select COM-4 Subtest Error Action [1-3,?]            (2) -> RETURN

        ** Error Start/Finish Text Options (bit mapped) **
    0x0001: Enable error start text
    0x0002: Enable error finish text

17: Select Error Start/Finish Message Mode [0x0-0x3,?]  (0x0) -> 0x3
18: Error Start Character Sequence [<error start string>,?]
                                                  (\033[?5i) -> RETURN
19: Error Finish Character Sequence [<error finish string>,?]
                                                  (\033[?4i) -> RETURN
20: Enter OK, or :NN to return to question NN [OK]       (OK) -> OK
```

If **OK** or RETURN is entered, the "Test Parameter Menu" terminates and all inputs are no longer changeable.

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

Table 4–3, Getting Help During Test Parameter Entry

Character	Description
:?	Reviews previous entries
?	Provides specific help where available

After the desired help information displays, the system redisplays the last prompt.

When all prompts are answered, the screen displays a TEST PARAMETER SUMMARY which displays the prompts that were answered and their responses. The following figure illustrates an example of a TEST PARAMETER SUMMARY screen. The actual values and responses vary according to the input.

Figure 4–4, Sample Test Parameter Summary

```
                    TEST PARAMETER SUMMARY

Select ioconfig file                                    : ioconfig
Ioconfig File Unit to Test                              : 0
Multibus IOP Number                                     : 4
Multibus Chassis Number                                 : 0
Multibus CSR of the Controller                          : 0x8c0
Multibus Interrupt Level of the Controller              : 7
Use Defaults for Remaining Parameters                   : n
Select External Connections                             : 0x0002
          -> Transmit clock source (1=DCE,0=DTE) <-
Select Serial Communication Lines                       : 0xf
Baud Rate for Loopback Subtests                         : 57600
Number of Iterations for Each Subtest                   : 1
Select Debug Mode                                       : 0x0000
Select COM-4 Subtest Error Action                       : 2
        -> Pause for subtest errors <-
Select Error Start/Finish Message Mode                  : 0x0003
        -> Enable error start text <-
        -> Enable error finish text <-
Error Start Character Sequence                          : \033[?5i
Error Finish Character Sequence                         : \033[?4i
Enter OK, or :NN to return to question NN               : OK
```

4.4.2 Prompt Explanations

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

```
1: Select ioconfig file [<filepath>,?]          (/ioconfig) ->
```

This option allows specification of an alternate */ioconfig* file. The file must still be in the same format as a conventional */ioconfig* file.

```
      PERIPHERAL CONFIGURATION DATA
     CCU    Chassis   Type     CSR    Int
   -------- ------- -------- ------ ---
1) iop  4      0    SCI-001  0x08c0  7
2) iop  4      0    SCI-001  0x08c2  6
3) iop  4      1    SCI-001  0x08c0  7
4) iop  4      1    SCI-001  0x08c2  6

       *** Enter 0 for manual configuration ***

2: Ioconfig File Unit to Test [1-4,0,?]                 (1) ->
```

Above, the applicable */ioconfig* file entries are listed (if any). To select a predefined controller configuration entry from the */ioconfig* file, enter the number which is found to the left of the desired entry. Entering a **0** allows each item within the controller configuration entry to be independently specified. The default values are initially the same as the first file entry. If most, but not all, the items associated with a given entry in */ioconfig* file are desired, select the number for that entry, back up to this prompt (via the ^ command), and specify **0** the second time. This

changes the default values for the independent items to be the same as the originally selected entry.

```
3: Multibus IOP Number [3-7,?]                    (4) ->
```

Enter the CCU slot number for the Multibus IOP which controls the Multibus chassis containing the Multibus X.25 Controller.

```
4: Multibus Chassis Number [0-1,?]                (0) ->
```

Enter the Multibus chassis number for the chassis where the Multibus X.25 Controller is installed. If there is only 1 chassis, the number is most likely 0.

```
5: Multibus CSR of the Controller [0x0-0xffe,?]   (0x8c0) ->
```

Enter the Multibus address of the Multibus X.25 Controller Status Register (CSR). This address is controlled by jumper block J28 on the controller. The standard address is 0x8c0. The following tables contain the address and jumper designations for jumper block J28 (near the Multibus edge connector P1). Also the standard address and interrupt levels for jumper blocks J28 and J37 are listed. The information contained in these tables can be viewed online by entering ? in response to the prompt.

Table 4–4, Jumper Block J28 Address and Jumper Designations

JUMPER BLOCK J28							
Address:	A7	A6	A5	A4	A3	A2	A1
Jumper:	1	2	3	4	5	6	7

Table 4–5, Standard Address and Interrupt Levels (J28, J37):

X.25 #	Address	Interrupt Level	Jumper J28 1234567	Jumper J37 123456789X
1	0x8c0	7	1100000	XX
2	0x8c2	6	1100001	X X

NOTE

In the previous table, the Xs under the J37 column indicate jumper pins on the J37 jumper that should be connected together.

```
6: Multibus Interrupt Level of the Controller [0-7,?]   (7) ->
```

Enter the CONVEX Multibus interrupt number assigned to the Multibus X.25 Controller under test. This value is between 0 and 7, inclusive. The interrupt level is selected by jumper block J37 (next to Multibus edge connector P1) on the controller. The standard level is 7. The

following table lists the pin/interrupt levels and the standard address/interrupt levels for jumper blocks J37 and J28. The information contained in the following tables can also be viewed online by entering **?** as a response to the prompt.

> **NOTE**
>
> Identify the desired interrupt level in the following table and connect the corresponding pin to J37 pin 10. For example, if the desired interrupt level is 2, connect J37 pin 4 to J37 pin 10.

Table 4–6, Jumper Block J37, Pin and Interrupt Levels

JUMPER BLOCK J37								
J37 Pin:	2	3	4	5	6	7	8	9
Interrupt Level:	0	1	2	3	4	5	6	7

Table 4–7, Standard Address and Interrupt Levels (J28, J37):

X.25 #	Address	Interrupt Level	Jumper J28 1234567	Jumper J37 123456789X
1	0x8c0	7	1100000	XX
2	0x8c2	6	1100001	X X

> **NOTE**
>
> In the previous table, the Xs under the J37 column indicate jumper pins on the J37 jumper that should be connected together.

```
7: Use Defaults for Remaining Parameters [y,n,?]          (y) ->
```

Enter **y** to select the default values for all remaining prompts. This avoids explicit entry of (RETURN) in response to remaining prompts.

```
8: RAM Pattern Subtest: pattern # 1 [<pattern>,?]
                                    (Use default patterns) ->
9: RAM Pattern Subtest: pattern # 2 [<pattern>,?]
                                    (Use default patterns) ->
10: RAM Pattern Subtest: pattern # 3 [<pattern>,?]
                                    (Use default patterns) ->
```

Select the pattern(s) to use for RAM pattern testing. Patterns are specified by typing the hexadecimal pattern (up to 8 hexadecimal digits). More than one pattern may be entered by entering the pattern, pressing (RETURN), and entering another pattern. This process may be continued until

the desired number of patterns are entered. When all patterns are entered, press RETURN without typing a pattern. There is a limit of 32 patterns.

```
    ** External Connections (bit mapped) **
0x0001:   Loopback jumper is installed
0x0002:   Transmit clock source (1=DCE,0=DTE)
0x0004:   Modem is attached to connector

11: Select External Connections [0x0-0x7,?]              (0x2) ->
```

This prompt controls whether or not the diagnostic allows use of an external loopback jumper. External loopback provides a cable check to the external connector panel, while internal loopback does not check any external cables. The external connections affect which Class 3 and Class 4 subtests are executed. The following list specifies which subtests require which of these options:

- Subtest 300—requires options 2 and 4 to be set (0x6 or 0x7)
- Subtest 301—requires options 1 and 2 to be set (0x3 or 0x7)
- Subtest 400—requires options 4 to be set and 2 to be clear (0x4 or 0x5)
- Subtest 401—requires options 1 to be set and 2 to be clear (0x1 or 0x5)

If the value selected is not one of the values contained in the previous list, no class 3 or class 4 subtests are executed. The selection of external connections may also be set by using the **-o** command line option during the test invocation sequence.

```
12: Select Serial Communication Lines [0x1-0xf,?]        (0xf) ->
```

Select which lines to test. Normally all lines are tested. This prompt allows testing on a subset of the lines. Lines are specified as a bitmask, with the least significant bit corresponding to line 1. The selection of serial communication lines may also be set by using the **-l** command line option at test invocation.

```
13: Baud Rate for Loopback Subtests [110-73728,?]      (57600) -> ?
```

Select the baud rate to use for loopback testing. If the baud rate is not valid, a warning will be displayed and the baud rate changed to a valid value. This option may also be set using the **-b** command line option during the test invocation sequence.

```
14: Number of Iterations for Each Subtest [1-2147483647,?]
                                                          (1) -> ?
```

Enter the number of times each subtest is to be executed on the Multibus X.25 Controller. The count is passed to the controller, so the repetitions are done as quickly as possible. This is most useful for repeating a subtest many times in quick succession. This option may also be set using the **-i** command line option.

```
    ** Debug Mode Flags (bit mapped) **
0x0001: Print messages sent to COM-4
0x0002: Print messages received from COM-4
0x0004: Pause before message sent to COM-4
0x0008: Pause after message received from COM-4
0x0010: Print MBS messages sent to CCU
0x0020: Print MBS messages received from CCU
0x0040: Pause before MBS message sent to CCU
```

```
0x0080: Pause after MBS message received from CCU
0x0100: Print extra informative messages
0x0200: Don't reset COM-4 after subtest is aborted
0x0400: Don't disable SPU interrupts when done
0x0800: Simulate all MBS messages (don't send them)

15: Select Debug Mode [0x0-0xfff,?]                    (0x0) ->
```

These options allow the standard test behavior to be altered in ways that are often useful for troubleshooting. In addition, the test can be paused before and after (or both) each command (i.e., typically one controller operation) is sent to the CCU driver. These flags may also be set using the **-v** and **-d** command line options during test invocation. The first four flags (0x0001–0x0008) control printing of all messages that are sent to the Multibus X.25 Controller. The next four flags (0x0010–0x0080) independently control printing (before or after) of messages that are sent to the CCU. If the flag 0x0100 is set, verbose messages are printed just as if the **-v** options had been set from the command line. If the flag 0x0200 is set, the controller is not reset after a subtest is aborted (used for debugging). If the flag 0x0400 is set, the SPU interrupts, MBS message interrupt and CCU print interrupt, are not disabled when *dev4540* is completed. If flag 0x0800 is set, a simulation of the test is done without affecting the CCU or the controller.

> **NOTE**
>
> Selection of the options is achieved by entering the hexadecimal mask obtained by ORing the desired options together. For example, if the first three options are desired, enter **0x7**. If all options are desired, enter **0xfff**.

```
    ** Subtest Error Actions **
1:  Silently ignore subtest errors
2:  Pause for subtest errors
3:  Abort subtest on first error

16: Select COM-4 Subtest Error Action [1-3,?]          (2) ->
```

Enter the desired subtest error action:

- Choice 1 ignores all errors (as if they never happened). This could be useful when it is necessary to just generate constant action (for example, EMI test).
- Choice 2 passes all errors to the *dshell* for processing. Choose this option for running the test normally [default].
- Choice 3 causes the subtest to abort when the first error is seen, regardless of any *dshell* error actions specified.

```
    ** Error Start/Finish Text Options (bit mapped) **
0x0001: Enable error start text
0x0002: Enable error finish text

17: Select Error Start/Finish Message Mode [0x0-0x3,?]  (0x0) ->
```

This prompt allows a specified character string to be sent to the printer before and after (or both) an error (and its data) is displayed. Although any string up to 64 characters in length may be specified, the intended use is to selectively turn a printer on before an error is displayed and to turn a printer off after an error has been displayed. This is a paper saving feature when running *dev4540* multiple times or for long periods of time. This assumes a printer is connected to the

auxiliary port on the display terminal where *dev4540* is running. It also assumes the auxiliary port can be turned on and off via an escape character sequence.

Selection of both options is achieved by entering the hexadecimal mask obtained by ORing the two options together (0x03). Control characters may be specified by using standard "C" programming style escape sequences. For example:

\033	Octal value of the ASCII escape character (0x1b)
\x1b	Same as above value but specified in hex.
\r	Carriage Return
\n	Line Feed
\t	Tab character
\b	Backspace character
\f	Form Feed character
\a	Bell ("alert") character
\v	Reverse line feed character

```
18: Error Start Character Sequence [<error start string>,?]
                                                    (\033[?5i) ->
```

Enter the string to be displayed before showing an error message. The default value is the "Enter Auto Print Mode" sequence for terminals that support VT100 terminal protocol.

Control characters may be specified by using standard "C" programming style escape sequences. For example:

\033	Octal value of the ASCII escape character (0x1b)
\x1b	Same as above value but specified in hex.
\r	Carriage Return
\n	Line Feed
\t	Tab character
\b	Backspace character
\f	Form Feed character
\a	Bell ("alert") character
\v	Reverse line feed character

```
19: Error Finish Character Sequence [<error finish string>,?]
                                                    (\033[?4i) ->
```

Enter the string to be displayed after showing an error message. The default value is the "Exit Auto Print Mode" sequence for terminals that support VT100 terminal protocol.

Control characters may be specified by using standard "C" programming style escape sequences. For example:

\033	Octal value of the ASCII escape character (0x1b)
\x1b	Same as above value but specified in hex.
\r	Carriage Return
\n	Line Feed
\t	Tab character
\b	Backspace character
\f	Form Feed character
\a	Bell ("alert") character
\v	Reverse line feed character

```
20: 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.

4.5 Initialization Sequence for *dev4540*

After the last prompt is entered, when not in quick start-up mode, and before subtest code execution begins, the following events occur:

- The diagnostic determines if the test was invoked for quick start-up (i.e., **dev4540x** or **dev4540 -q**). If so, the diagnostic reads the test parameters from the specified parameter file (default parameter file is */tmp/dev4540.tmp*). If not, the input parameters are written to the parameter file (default or user specified).
- The diagnostic checks to see if the CCU is already loaded with the *dev4540* CCU driver. If the driver is not loaded, the CCU is loaded. After the load completes, the driver is configured for EGOS and then the EGOS probe message starts the driver.

> **NOTE**
>
> The file */mnt/boot_db* is used to determine what memory is installed. If this file is nonexistent, it can be created with the command: **scn_util -b > /mnt/boot_db** from the **(spu)>** prompt.

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

4.6 Multibus X.25 Controller EPROM Self-tests

The Multibus X.25 Controller contains self-diagnostic routines that are executed immediately after any reset to the controller or anytime power is applied. The controller's EPROM is always resident in memory starting at address 0xfc0000 through 0xfc3fff. The controller is reset before the first subtest and after any subtest failures. If the EPROM self-tests fail, a standard timeout error message is output by *dev4540*. When an EPROM self-test fails, the controller blinks the 8-

bit error code on its single, red LED indicating the failed test. The 8-bit error code (most significant bit to least significant bit) can be identified by the sequence and duration of the blinks. Each bit of the error code occupies a constant interval of time; however, the duration of the LED blink indicates a "1" or a "0" as follows:

- A long pause (4 bit times) indicates the start of an error code
- A long blink indicates a "1"
- A short blink indicates a "0"

The error code continues blinking until the controller is reset (which causes the EPROM self-test to re-execute). The following list describes all possible EPROM error codes and their meanings (all values are binary values):

00010000 EPROM checksum wrong
00100000 EPROM system RAM (0x800–0xfff) column functionality failed
00100001 EPROM system RAM (0x800–0xfff) uniqueness test failed
00100010 EPROM system RAM (0x800–0xfff) pattern test (0x5555, first pattern) failed
00100011 EPROM system RAM (0x800–0xfff) pattern test (0xAAAA) failed
00100100 EPROM system RAM (0x800–0xfff) pattern test (0x5454) failed
00100101 EPROM system RAM (0x800–0xfff) pattern test (0xA8A8) failed
00110000 High RAM (0x01000–0x7ffff) column functionality failed
00110001 High RAM (0x01000–0x7ffff) uniqueness test failed
00110010 High RAM (0x01000–0x7ffff) pattern test (0x5555, first pattern) failed
00110011 High RAM (0x01000–0x7ffff) pattern test (0xAAAA) failed
00110100 High RAM (0x01000–0x7ffff) pattern test (0x5454) failed
00110101 High RAM (0x01000–0x7ffff) pattern test (0xA8A8) failed

4.6.1 EPROM Self-Test Descriptions

The following sections contain descriptions of the EPROM self-tests. In the following sections:

- System RAM refers to addresses 0x800 through 0xfff
- High RAM refers to addresses 0x01000 though 0x7ffff

4.6.1.1 Self-Test 00010000, EPROM Checksum

Self-test 00010000 verifies the EPROM checksum. The EPROM checksum is computed by summing the contents of the EPROM 32 bits at a time over the entire EPROM. If the result is non-zero, the checksum is incorrect.

4.6.1.2 Self-Test 00100000, EPROM System RAM Column Functionality

Self-test 00100000 verifies the ability to independently change each bit of local RAM. This self-test performs a walking ones test (0x0001, 0x0002, 0x0004, 0x0008, ..., 0x4000, 0x8000) and a walking zeros test (0xFFFE, 0xFFFD, 0xFFFB, 0xFFF7, ..., 0xBFFF, 0x7FFF) on the 16-bit word at location 800 in local RAM.

4.6.1.3 Self-Test 00100001, EPROM System RAM Uniqueness Test

Self-test 00100001 verifies the ability to address each system RAM location. The self-test writes an incrementing pattern (0x0001, 0x0002, 0x0003, ..., 0x0400) to system RAM. Each location is then checked to verify that it contains the correct value.

4.6.1.4 Self-Test 00100010, EPROM System RAM Pattern Test (0x5555)

Self-test 00100010 verifies the ability to check system RAM with the sequence pattern 0x5555. The self-test writes all locations with the complement of the pattern (0xAAAA), writes all locations with the pattern 0x5555, and reads and verifies all RAM locations contain the value 0x5555.

4.6.1.5 Self-Test 00100011, EPROM System RAM Pattern Test (0xAAAA)

Self-test 00100011 verifies the ability to check system RAM with the sequence pattern 0xAAAA. The self-test writes all locations with the complement of the pattern (0x5555), writes all locations with the pattern 0xAAAA, and reads and verifies all RAM locations contain the value 0xAAAA.

4.6.1.6 Self-Test 00100100, EPROM System RAM Pattern Test (0x5454)

Self-test 00100100 verifies the ability to check system RAM with the sequence pattern 0x5454. The self-test writes all locations with the complement of the pattern (0xABAB), writes all locations with the pattern 0x5454, and reads and verifies all RAM locations contain the value 0x5454.

4.6.1.7 Self-Test 00100101, EPROM System RAM Pattern Test (0xA8A8)

Self-test 00100101 verifies the ability to check system RAM with the sequence pattern 0xA8A8. The self-test writes all locations with the complement of the pattern (0x5757), writes all locations with the pattern 0xA8A8, and reads and verifies all RAM locations contain the value 0xA8A8.

4.6.1.8 Self-Test 00110000, High RAM Column Functionality

Self-test 00110000 verifies the ability to independently change each bit of local RAM. This self-test performs a walking ones test (0x0001, 0x0002, 0x0004, 0x0008, ..., 0x4000, 0x8000) and a walking zeros test (0xFFFE, 0xFFFD, 0xFFFB, 0xFFF7, ..., 0xBFFF, 0x7FFF) on the 16-bit word at location 0x1000 in local RAM.

4.6.1.9 Self-Test 00110001, High RAM Uniqueness Test

Self-test 00110001 verifies the ability to address each high RAM location. The self-test writes an incrementing pattern (0x0001, 0x0002, 0x0003, ..., 0xFFFF, 0x0001, 0x0002, ...) to high RAM. Each location is then checked to verify that it contains the correct value.

4.6.1.10 Self-Test 00110010, High RAM Pattern Test (0x5555)

Self-test 00110010 verifies the ability to check high RAM with the sequence pattern 0x5555. The self-test writes all locations with the complement of the pattern (0xAAAA), writes all locations with the pattern 0x5555, and reads and verifies all RAM locations contain the value 0x5555.

4.6.1.11 Self-Test 00110011, High RAM Pattern Test (0xAAAA)

Self-test 00110011 verifies the ability to check high RAM with the sequence pattern 0xAAAA. The self-test writes all locations with the complement of the pattern (0x5555), writes all locations with the pattern 0xAAAA, and reads and verifies all RAM locations contain the value 0xAAAA.

4.6.1.12 Self-Test 00110100, High RAM Pattern Test (0x5454)

Self-test 00110100 verifies the ability to check high RAM with the sequence pattern 0x5454. The self-test writes all locations with the complement of the pattern (0xABAB), writes all locations with the pattern 0x5454, and reads and verifies all RAM locations contain the value 0x5454.

4.6.1.13 Self-Test 00110101, High RAM Pattern Test (0xA8A8)

Self-test 00110101 verifies the ability to check high RAM with the sequence pattern 0xA8A8. The self-test writes all locations with the complement of the pattern (0x5757), writes all locations with the pattern 0xA8A8, and reads and verifies all RAM locations contain the value 0xA8A8.

4.7 Class Descriptions

The *dev4540* test contains the four classes of subtests listed in the following table:

Table 4–8, *dev4540* Test Classes

CLASS	DESCRIPTION
1	Multibus X.25 EPROM-Based Controller Tests
2	Multibus X.25 Internal Loopback Tests
3	Multibus X.25 External Connection Tests (External Transmit Clock)
4	Multibus X.25 External Connection Tests (Internal Transmit Clock)

All subtests contained in *dev4540* are loopable under the *dshell*.

4.8 Class 1 Subtests

Class 1 subtests consist of basic controller tests involving EPROM, parity, interrupt, and RAM. Class 1 subtests test only the Multibus X.25 Controller; no Serial Communications Interface (SCI) modules are required. Class 1 subtests verify:

- The ability to communicate with the Multibus Input/Output Processor (MIOP), Multibus Control Unit (MBCU), and the Multibus X.25 Controller
- The ability of the Multibus X.25 Controller to interrupt the MIOP
- The EPROM checksum is correct
- The Multibus X.25 Controller RAM is fully functional:
  - Each RAM location is unique (no aliasing)
  - The RAM is pattern tested for data correctness
- Parity checking logic is fully functional

Table 4–9, Class 1 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
100	Multibus X.25 Basic Controller Test	00:40[1]
101	Multibus X.25 RAM Pattern Test	00:35[2]
102	Multibus X.25 RAM Parity Test	00:01

[1] Time varies depending on whether or not the controller must complete its power-on self-check.
[2] Time varies depending on the number of patterns input for testing.

4.8.1 Subtest 100, Multibus X.25 Basic Controller Test

Subtest 100 verifies the EPROM checksum and tests RAM locations 0x01000 through 0x7FFFF. The EPROM checksum is computed by summing the contents of the EPROM 32 bits at a time over the entire EPROM. If the result is non-zero, the checksum is incorrect.

The RAM test consists of the following parts (in order of execution):

- Column test: verify ability to independently change each bit
- Uniqueness test: verify ability to address each RAM location
- Pattern test: check RAM with the sequence of patterns 0x5555, 0xAAAA, 0x5454, 0xA8A8.

> **NOTE**
>
> This subtest duplicates tests executed in the EPROM power-on self-test code. This means any non-intermittent problems in these areas are likely to cause self-test failures (indicated with the red LED on the controller). For detailed descriptions of these subtest routines, refer to the "EPROM Self-Test Descriptions" section in this chapter.

4.8.2 Subtest 101, Multibus X.25 RAM Pattern Test

Subtest 101 verifies RAM locations 0x01000 through 0x7FFFF with a pattern test, using up to 32 patterns specified by the user. The default patterns are shown in the following table:

Table 4–10, Subtest 101 Default Test Patterns

Description	Hexadecimal Pattern
All zeros	00000000
All ones	FFFFFFFF
Individual 1 bits	01010101, 02020202, 04040404, 08080808, 10101010, 20202020, 40404040, 80808080
Individual 0 bits	FEFEFEFE, FDFDFDFD, FBFBFBFB, F7F7F7F7, EFEFEFEF, DFDFDFDF, BFBFBFBF, 7F7F7F7F
Alternate 1 and 0	55555555, AAAAAAAA
Two 1's and a 0	DB6DB6DB, B6DB6DB6, 6DB6DB6D
Two 0's and a 1	24924924, 49249249, 92492492
Two 1's and two 0's	CCCCCCCC, 99999999, 33333333, 66666666

4.8.3 Subtest 102, Multibus X.25 RAM Parity Test

Subtest 102 verifies correct operation of the Multibus X.25 Controller RAM parity checking. It verifies both even and odd parity, including both correct and incorrect parity. The patterns used to check the parity operations are shown in the following table:

Table 4–11, Subtest 102 Parity Check Patterns

Description	Hexadecimal Pattern
All zeros	00
All ones	FF
Individual 1 bits	01, 02, 04, 08, 10, 20, 40, 80
Individual 0 bits	FE, FD, FB, F7, EF, DF, BF, 7F
Alternate 1 and 0	55, AA
Two 1's and a 0	DB, B6, 6D
Two 0's and a 1	24, 49, 92
Two 1's and two 0's	CC, 99, 33, 66

4.9 Class 2 Subtest

The class 2 subtest verifies only the Multibus X.25 Controller. No Serial Communications Interface (SCI) modules are required. The class 2 subtest performs a system check that tests the following functional units:

- Serial Communications Controllers (SCC) in loopback mode
- Direct Memory Access (DMA) controllers
- Interrupt generation and arbitration logic

The system check is executed using both local (Multibus X.25 Controller) memory and main memory mapped via Multibus as the source of DMA data to the SCC.

Table 4–12, Class 2 Subtest

SUBTEST	DESCRIPTION	TIME (min:sec)
200	Multibus X.25 System Test (Internal Loopback)	00:12[1]

[1] Time varies depending on the selected baud rate.

4.9.1 Subtest 200, Multibus X.25 System Test (Internal Loopback)

Subtest 200 tests the DMA controllers, SCC controllers, interrupt controller, and bus interface. The subtest is designed to use the components together as a system to check for interactions between parts as well as testing individual parts. The subtest executes with the SCC in local loopback mode. This test does not require any external connections; the Multibus X.25 Controller is all that is necessary.

Subtest 200 is designed to use DMA to send data through the SCC, loop the data back to the SCC (in local loopback mode), and read the data from the SCC with DMA. Various SCC and DMA interrupts are enabled and checked, including DMA completion interrupts.

The subtest logic is:

- Fill source buffer with the incrementing pattern (00, 01, 02, ..., FE, FF, 00, ...)
- Reset all SCC channels
- Initialize selected SCC channels for transfer in SDLC mode
- Initialize DMA channels corresponding to the SCC channels
- Set up a timeout based on baud rate and transfer size
- Start the transfer
- Wait until transfer completes or timeout expires
- Verify that received data pattern matches source data

This subtest logic is executed three times. The first execution uses a small buffer (128 bytes) in controller local memory as the source buffer. The second execution uses a large buffer (8 Kbytes) in controller local memory as the source buffer. The third execution uses a buffer (one page – 4,096 bytes) in IOP local memory accessed via the cache controller (mapped to Multibus memory).

4.10 Class 3 Subtests

Class 3 subtests verify communication through the Serial Communications Interface (SCI) modules. Subtest 301 requires the use of an external loopback jumper (refer to the "Prerequisites and Required Equipment" section in this chapter for the CONVEX part number). The subtests run the same system check used by class 2 except that subtest 301 sends the data through the SCI modules and subtest 300 uses the SCC loopback mode with the clock supplied by the modem.

For any given system configuration, only subtests from either class 3 or class 4 are executed. Class 3 is executed for the standard configuration with the Transmit Clock signal supplied by the external device (typically a modem).

Class 3 subtests are different variations of a system test that tests the DMA controllers, SCC controllers, interrupt controller, and bus interface. The subtests use the components together as a system to check for interactions between parts as well as testing individual parts.

The subtests use DMA to send data through the SCC, loop the data back to the SCC (depending on which subtest), and read the data from the SCC with DMA. Various SCC and DMA interrupts are enabled and checked, including DMA completion interrupts.

The logic of the subtests is:

- Fill source buffer with the incrementing pattern (00, 01, 02, ..., FE, FF, 00, ...)
- Reset all SCC channels
- Initialize selected SCC channels for transfer in SDLC mode
- Initialize DMA channels corresponding to the SCC channels
- Set up a timeout based on baud rate and transfer size
- Wait for Data Carrier Detect (DCD) to be active for selected SCC channels
- Start the transfer
- Wait until transfer completes or timeout expires
- Verify that received data pattern matches source data

This subtest logic is executed three times. The first execution uses a small buffer (128 bytes) in controller local memory as the source buffer. The second execution uses a large buffer (8 Kbytes) in controller local memory as the source buffer. The third execution uses a buffer (one page – 4,096 bytes) in IOP local memory accessed via the cache controller (mapped to Multibus memory). The following table lists all class 3 subtests, their descriptions, and the time required to execute them:

> **NOTE**
>
> Subtest 301 is only applicable for versions of *dev4540* with build dates later than July 19, 1989. Earlier versions of this diagnostic do not contain subtest 301 and subtest 300 has different functionality. Refer to subtest 300's description for details.

Table 4–13, Class 3 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
300	Multibus X.25 System Test (External Clock Source, DCE Tx Clock)	00:12[1]
301	Multibus X.25 System Test (External Loopback, DCE Tx Clock)	00:12[1]

[1] Time varies depending on the selected baud rate.

4.10.1 Subtest 300, Multibus X.25 System Test (External Clock source, DCE Tx Clock)

Subtest 300 is for use with Multibus X.25 Controllers and SCI modules that are jumpered to use the transmit clock from the modem as the Tx clock source (the standard configuration). Subtest 300 tests the clocks from the modem to ensure that the jumpers are set up correctly on the controller, the SCI module, and the modem. The transmit and receive clocks from the modem are used, but the data is looped back through the SCC internal loopback option. No external loopback connectors are necessary to execute this subtest, although a modem must be connected to provide the transmit and receive clock signals.

> **NOTE**
>
> This description of Subtest 300 applies to versions of *dev4540* built after July 19, 1989. The version of *dev4540* can be determined with the **-V** command line option issued during test invocation. In earlier versions of the diagnostic, subtest 300 performed the same functionality as subtest 401 with external connect option set to 0x1.

4.10.2 Subtest 301, Multibus X.25 System Test (External Loopback, DCE Tx Clock)

Subtest 301 is for use with the Multibus X.25 Controller and SCI modules which are jumpered to use the transmit clock from the modem as the Tx clock source (the standard configuration). Subtest 301 tests the data paths from the Multibus X.25 Controller to the external panel and back to the controller. The transmit and receive clocks used depend on whether or not a modem is present. If a modem is present, the clocks from the modem are used; if no modem is present, the SCC's baud rate generator is used as both the transmit and receive clocks. A loopback cable (refer to the "Prerequisites and Required Equipment" section in this chapter for the CONVEX part number) must be installed for this subtest to execute correctly.

> **NOTE**
>
> This description of Subtest 301 applies to versions of *dev4540* built after July 19, 1989. The version of *dev4540* can be determined with the **-V** command line option. This subtest does not exist in earlier versions of the diagnostic.

4.11 Class 4 Subtests

Class 4 subtests verify communication through the Serial Communications Interface (SCI) modules. Subtest 401 requires the use of an external loopback jumper (refer to the "Prerequisites and Required Equipment" section in this chapter for the CONVEX part number). The subtests run the same system check used by class 2 except that subtest 401 sends the data through the SCI modules and subtest 400 uses the SCC loopback mode with the clock supplied by the modem.

Class 4 subtests are different variations of a system test that tests the DMA controllers, SCC controllers, interrupt controller, and bus interface. The subtests use the components together as a system to check for interactions between parts as well as testing individual parts.

For any given system configuration, only subtests from either class 3 or class 4 are executed. Class 4 is executed for alternate configurations with the Transmit Clock signal supplied by the Multibus X.25 Controller.

The subtests use DMA to send data through the SCC, loop the data back to the SCC (depending on which subtest), and read the data from the SCC with DMA. Various SCC and DMA interrupts are enabled and checked, including DMA completion interrupts.

The logic of the subtests is:

- Fill source buffer with the incrementing pattern (00, 01, 02, ..., FE, FF, 00, ...)
- Reset all SCC channels
- Initialize selected SCC channels for transfer in SDLC mode
- Initialize DMA channels corresponding to the SCC channels
- Set up a timeout based on baud rate and transfer size
- Wait for Data Carrier Detect (DCD) to be active for selected SCC channels
- Start the transfer
- Wait until transfer completes or timeout expires
- Verify that received data pattern matches source data

Each subtest executes this subtest logic three times. The first execution uses a small buffer (128 bytes) in controller local memory as the source buffer. The second execution uses a large buffer (8 Kbytes) in controller local memory as the source buffer. The third execution uses a buffer (one page – 4,096 bytes) in IOP local memory accessed via the cache controller (mapped to Multibus memory).

> **NOTE**
>
> Subtest 401 is only applicable for versions of *dev4540* with build dates later than July 19, 1989. Earlier versions of this diagnostic do not contain subtest 401 and have different functionality in subtest 400. Refer to subtest 400's description for details.

Table 4–14, Class 4 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
400	Multibus X.25 System Test (External Clock source, DTE Tx Clock)	00:12
401	Multibus X.25 System Test (External Loopback, DTE Tx Clock)	00:12

4.11.1 Subtest 400, Multibus X.25 System Test (External Clock source, DTE Tx Clock)

Subtests 400 and 401 are for use with the Multibus X.25 Controller and SCI modules that are jumpered to use the receive clock from the modem as the Tx clock source. This is a nonstandard configuration that allows more thorough diagnostic testing.

Subtest 400 tests the clocks from the modem to ensure that the jumpers are set up correctly on the controller, the SCI module, and the modem. The receive clock from the modem is used as both the transmit and receive clocks, but the data is looped back through the SCC internal loopback option. No external loopback connectors are necessary to execute this subtest, although a modem must be connected to provide the receive clock signal.

> **NOTE**
>
> This description of Subtest 400 applies to versions of *dev4540* built after July 19, 1989. The version of *dev4540* can be determined with the **-V** command line option issued during test invocation. In earlier versions of the diagnostic, subtest 400 performed the same functionality as subtest 401 with external connect option set to 0x5.

4.11.2 Subtest 401, Multibus X.25 System Test (External Loopback, DTE Tx Clock)

Subtest 401 is for use with the Multibus X.25 Controller and SCI modules which are jumpered to use the receive clock from the modem as the Tx clock source. This is a nonstandard configuration which allows more thorough diagnostic testing.

Subtest 401 tests the data paths from the Multibus X.25 Controller to the external panel and back to the controller. The controller always sends a transmit clock to the SCI module as the external transmit clock. If a modem is present, the receive clock from the modem is used as the transmit clock; if no modem is present, the baud rate generator is used to generate a transmit clock. The receive clock is always taken from the external connection. If a modem is present, the receive clock is from the modem. If no modem is present, the loopback cable connects the external transmit clock to the receive clock, so the receive clock is an echo of the transmit clock. A loopback cable must be installed for this subtest to execute correctly (refer to the "Prerequisites and Required Equipment" section in this chapter for the CONVEX part number).

> **NOTE**
>
> Subtest 401 is only applicable for versions of *dev4540* with build dates later than July 19, 1989. Earlier versions of this diagnostic do not contain subtest 401 and have different functionality in subtest 400. Refer to subtest 400's description for details.

4.12 Error Messages

There are three basic types of error messages associated with this diagnostic:

- Error messages associated with the diagnostic's start-up
- Error messages associated with the execution of the diagnostic's subtests (related with system errors)
- Error messages associated with the execution of the diagnostic's subtests (related with subtest errors)

4.12.1 Start-Up Error Messages

The first type of error message is output after start-up errors (i.e., before the first subtest is executed). These messages usually do not contain header or footer information. The following error messages may occur during the start-up sequence of the diagnostic:

```
COM-4 68010 Bus Error Exception  (0x02)

COM-4 68010 Address Error Exception  (0x03)

COM-4 68010 Parity Error Exception  (0x1E)

Unable to initialize processor queues  (0x01)

Unable to initialize CCU  (0x04)

Unable to configure SCI-001 driver  (0x05)

Unable to probe SCI-001 driver  (0x06)

File is not in b.out format  (0x07)

Timeout waiting for CCU driver reply  (0x08)

MBS error occurred  (0x09)

File not found  (0x0A)

Invalid command  (0x10)

COM-4 not initialized  (0x11)

Timeout waiting for COM-4 reply  (0x12)

COM-4 is owned by another process  (0x13)

Unable to allocate CCU windows for message  (0x14)

No message buffers available for message  (0x15)

Unable to reset board (bad CSR?)  (0x16)

Unable to write to send interrupt  (0x17)
```

```
Unable to write to clear interrupt  (0x18)

COM-4 did not go idle/not stopped for debug  (0x19)

COM-4 did not respond to FL_READY  (0x1A)

COM-4 did not interrupt when done  (0x1B)

Unable to allocate main memory  (0x1C)

Pending message flushed by IOP  (0x1D)

Write to COM-4 memory failed  (0x1F)

Error setting up test  (0x20)

COM-4 has wrong status  (0x21)
```

4.12.2 General Error Messages

The second type of *dev4540* error message can occur at anytime during the execution of the diagnostic. These error messages are usually non-subtest related errors that occur while a subtest is executing. Each error message contains a standard header and footer. The message, placed between the header and footer, will consist of one or more of the error messages listed in the first type of error messages (start-up error messages). The following figure contains an example of the standard header message:

Figure 4–5, Standard Header for Error Messages

```
*****  Sun Jul 16 15:04:10 1989  *****
Test:    dev4540.t  1.1    Class: 1    Subtest: 100 1.2   Count: 1    Error: 0
Failed:  Multibus X.25 Basic Controller Test
```

The following figure contains an example of a standard footer that is printed after the standard header and error message:

Figure 4–6, Standard Footer for Error Messages

```
Test 'dev4540.t' failed
Elapsed time:   0:00:03
```

4.12.3 Subtest Error Messages

The third type of error messages are errors reported by an executing subtest (i.e., subtest failures). Each error message contains a standard header and footer that are included with the message. Also, the error message contains the following information:

- Iteration—lists which iteration of the subtest was being executed at the time of the failure
- State—describes the current program status, usually "Paused for error."
- Reason(s)—describes the error condition and usually contains actual and expected values

The following list contains all possible reasons that could be listed in the error message:

```
Reason(s):      Unexpected odd parity at address 0xXXXXXX
                Expected: ddd  Actual: ddd

Reason(s):      Unexpected even parity at address 0xXXXXXX
                Expected: ddd  Actual: ddd

Reason(s):      Didn't get expected odd parity at address 0xXXXXXX
                Expected: ddd  Actual: ddd

Reason(s):      Didn't get expected even parity at address 0xXXXXXX
                Expected: ddd  Actual: ddd

Reason(s):      Serial Communications Controller error
                Unexpected SCC interrupt, line d
                Didn't receive SCC interrupt, line d
                Transmit underrun, line d
                Abort received, line d
                CRC error or Receive overrun, line d

Reason(s):      Timeout waiting for DMA to finish
                Unexpected interrupt, DMA channel d (line d transmit)
                Unexpected interrupt, DMA channel d (line d receive)
                Didn't receive interrupt, DMA channel d (line d transmit)
                Didn't receive interrupt, DMA channel d (line d receive)

Reason(s):      Data verify failed at offset 0xXXXXXX on line d
                Expected: 0xXXXX  Actual: 0xXXXX

Reason(s):      Controller EPROM checksum error (checksum is 0xXXXXXXXX)

Reason(s):      RAM check failed at 0xXXXXXX
                Expected: 0xXXXX  Actual: 0xXXXX  (subtest 100)
                Expected: 0xXXXXXXXX  Actual: 0xXXXXXXXX  (subtest 101)
                Expected: 0xXX  Actual: 0xXX  (subtest 102)

Reason(s):      Timeout waiting for DCD to become active
                DCD not active, line d

Reason(s):      Invalid Diagnostic command

Reason(s):      Invalid subtest number

Reason(s):      Subtest failed
```

```
Reason(s):      Subtest aborted

Reason(s):      A test is already running

Reason(s):      No test is currently running

Reason(s):      No test is waiting to continue

Reason(s):      COM-4 RAM test program not running
```

Examples of the standard header and footer messages are shown in the description of the General Error error messages.

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

P

R

S

T

U

V

W

CONVEX Multibus X.25 Controller
(*dev4540*) Diagnostics Manual
Document No. 760-002730-000
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)