

CONVEX Multibus Emulator Controller
(*dev4600*) Diagnostics Manual
Document No. 760-002830-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX Multibus Emulator Controller (dev4600) Diagnostics Manual
Order No. DHW-240
First Edition

© 1991 CONVEX Computer Corporation
All rights reserved.

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

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

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

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

Printed in the United States of America

Revision Sheet
CONVEX Multibus Emulator Controller
(dev4600) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-002830-000	May 1991	First release. Contains the <i>dev4600</i> diagnostic test information from the <i>CONVEX PBUS I/O Systems Diagnostics Manual</i> .

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 (<i>dshell</i>) Overview	3-1
3.3 Syntax Help for <i>dshell</i> Commands	3-3

4 Multibus Emulator Controller Test (*dev4600*)

4.1 Overview	4-1
4.2 Prerequisites and Required Equipment	4-1
4.3 Test Invocation	4-2
4.3.1 Test Parameter Menu	4-4
4.3.2 Prompt Explanations	4-6
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 Capability	4-8
4.5.1.2 Subtest 101, Programmed I/O Loopback	4-8
4.5.1.3 Subtest 102, FCNx, STTx Status	4-8
4.5.1.4 Subtest 103, Attention Signal/Flag	4-9
4.5.1.5 Subtest 104, DMA Preparatory	4-9
4.5.1.6 Subtest 105, DMA Abort Via Attention and Reset	4-9
4.5.1.7 Subtest 106, Interrupt	4-9
4.5.1.8 Subtest 107, DMA Output Loopback	4-9
4.5.1.9 Subtest 108, DMA Input Loopback	4-10
4.5.2 Class 2 Subtest	4-10
4.5.2.1 Subtest 200, Data Parity	4-10
4.5.3 Class 3 Subtests	4-11
4.5.3.1 Subtest 300, DMA Output	4-11
4.5.3.2 Subtest 301, DMA Input	4-11
4.6 Error Messages	4-12

Appendixes

A Reporting Problems

A.1 Overview	A-1
A.2 Technical Assistance Center	A-1
A.3 The <i>contact</i> 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 <i>contact</i> Utility	A-2
A.5.1 Using a <i>.contact</i> File	A-3
A.5.2 Aborting the Report	A-3
A.5.3 Submitting the <i>dead.report</i> 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 <i>contact</i> 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 <i>dshell</i> Commands	3-2
4-1 Untested External Signals	4-1
4-2 Partially Tested Signals	4-1
4-3 Hardware Requirements	4-2
4-4 Getting Help During Test Parameter Entry	4-4
4-5 <i>dev4600</i> Test Classes	4-8
4-6 Class 1 Subtests	4-8
4-7 DMA Buffers Sizes Tested	4-9
4-8 Class 2 Subtest	4-10
4-9 Class 3 Subtests	4-11
4-10 DMA Buffer Sizes Tested	4-11
4-11 IOP Access Error Messages	4-12
4-12 IOP Processing Error Messages	4-13

List of Figures

2-1 EGOS' Position in the Environment	2-3
3-1 Syntax Help for the <i>loop</i> Command	3-3
4-1 Test Invocation Sequence	4-2
4-2 Alternate Test Invocation Sequence	4-3
4-3 Test Parameter Menu	4-5
4-4 Sample Test Parameter Summary	4-7
4-5 Interpreted Status Word Error Messages	4-13

Preface

Purpose and Intended Audience

This manual explains how to run the *dev4600* diagnostic, which checks the IKON DR-11W Emulator model 10077. This document is not a tutorial, but rather a reference for the users of the *dev4600* 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 *dev4600* 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 Emulator Controller Test (*dev4600*)**—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-240.
The document number for this manual is 760-002830-000.

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 (<i>cat</i>)	Description
<i>cpu</i>	CPU subsystem related test
<i>dev</i>	Peripheral device test
<i>io, idc, tli</i>	I/O subsystem related test
<i>mem</i>	Memory subsystem related test
<i>spu</i>	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 (<i>type</i>)	Description
<i>0</i>	Standalone test
<i>1</i>	Self-test
<i>2</i>	Kernel hardware test
<i>4, 5</i>	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 (<i>dev</i>)	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
<i>cpu4041.t</i>	SP2 object code in <i>b.out</i> format for <i>cpu4041</i>
<i>cpu4041.rnn</i>	C210 or C220 machine object code in SOFF format (relocatable)
<i>cpu4041.x00</i>	C210 or C220 machine object code in SOFF format (linked to run in segment 0)
<i>mem4000.t</i>	SP2 object code in <i>b.out</i> format for <i>mem4000</i>
<i>mem4000.x00</i>	C210 or C220 machine object code in SOFF format (linked to run in segment 0)
<i>dev4100.t</i>	SP2 object code in <i>b.out</i> format for <i>dev4100</i>
<i>dev4100.x00</i>	IOP object code in <i>b.out</i> 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

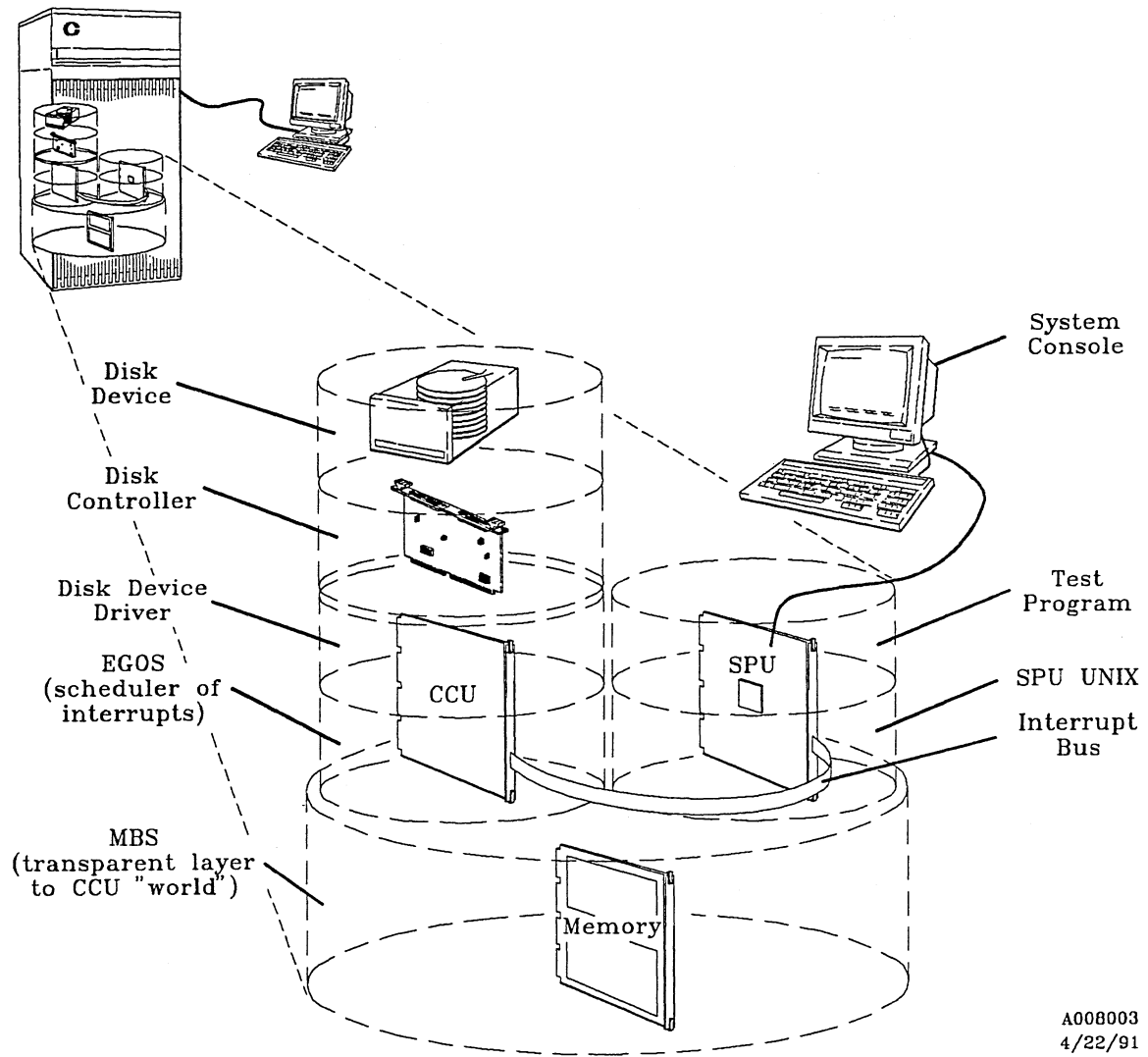
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes 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
<i>!</i> [<i>command</i>]	This command is used to access, or <i>fork</i> a UNIX shell to execute the command that follows <i>!</i> .
<i>exit</i>	The <i>exit</i> command causes immediate termination of the <i>dshell</i> process and any test processes that may have been forked.
<i>quit</i>	The <i>quit</i> command causes immediate termination of the <i>dshell</i> process and any test processes that may have been forked.
<i>^C</i>	Returns user to the <i>dshell</i> command level if no subtest is running.
<i>^B</i>	Immediately terminate the <i>dshell</i> and any associated active processes. Core is dumped.
<i>help</i>	The <i>help</i> command causes a standard <i>help</i> menu to be displayed. The menu describes the correct command syntax for each <i>dshell</i> command and gives a terse description of what each command does.
<i>status</i>	The <i>status</i> command generates a report on the current state of the <i>dshell</i> command options. This report gives the name of each flag, its current value, and an explanation of its current effect.
<i>log</i> [<i>options</i>]	The <i>log</i> command provides a mechanism for specifying the number of failures that will be allowed to occur before a test or subtest terminates execution.
<i>loop</i> [<i>options</i>]	The <i>loop</i> command causes the <i>dshell</i> to repeat the execution of a test or subtest.
<i>msgs</i> [<i>options</i>]	The <i>msgs</i> command enables or disables different levels of test, class, and subtest result messages.
<i>pause</i> [<i>options</i>]	The <i>pause</i> command returns program control to the <i>dshell</i> to the beginning, end, or failure of all or specific subtests.
<i>test</i> [<i>options</i>]	The <i>test</i> 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 Emulator Controller Test (*dev4600*)

4.1 Overview

The *dev4600* test checks the IKON DR-11W Emulator model 10077. Certain operating modes and signals are not tested. Items not tested include byte-mode I/O, and read-after-write DMA mode, and certain signals not used for interprocessor link. In addition, the following external signals are not tested:

Table 4-1, Untested External Signals

DIRECTION	NAME
Output	READY H GO H END CYCLE H

Also, the following external signals are only partially tested:

Table 4-2, Partially Tested Signals

DIRECTION	NAME	DESCRIPTION
Output	BUSY H INIT H	When switched for high assertion Assertion not tested
Input	CO CNTL H CYCLE RQ B H WC INC ENB H BA INC ENB H A00 H	Read-modify-write and write bytes not tested Request cycle input B not tested Suppress word count increment not tested Suppress word address increment not tested Byte write select not tested

4.2 Prerequisites and Required Equipment

The *dev4600* test requires a functional IKON controller model 10077. Class 3 subtests require the use of a second functional DR-11W mounted in the same Multibus.

NOTE

For most subtests, Classes 1 and 3, two normal DR-11W data cables, CONVEX part number 604-100004-002, are required. However, to execute the Class 2 Subtest, a special loopback cable, CONVEX part number 604-100004-001, is required.

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

Table 4-3, 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 *dev4600* 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 *dev4600* 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)
(sp�)> sysreset (RETURN)
(sp�)> mminit -s (RETURN)
(sp�)> dshell (RETURN)
: test dev4600 [-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 dev4600** executes all *dev4600* 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 dev4600 [-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 4-4, Getting Help During Test Parameter Entry

Character	Description
?	Displays this help menu
h	Provides help for a specific prompt
i	Displays the <i>/ioconfig</i> file

After the desired help information displays, the system beeps and redisplay the last prompt.

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

Figure 4-3, Test Parameter Menu

```

ENTER TEST PARAMETERS

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

PERIPHERAL CONFIGURATION DATA
CCU    Chassis Type    CSR    Int Unit Type
-----

Device 0 = user defined configuration

1: Device Under Test Selection [0,?]          (0) ->
2: Controller Under Test IOP [3-7]1         ( ) ->
3: Controller Under Test Multibus Chassis [0-3] ( ) ->
4: Controller Under Test Offset in Multibus [0x0-0xffff] ( ) ->
5: Controller Under Test Interrupt Number [0-7] ( ) ->

PERIPHERAL CONFIGURATION DATA
CCU    Chassis Type    CSR    Int Unit Type
-----

Device 0 = user defined configuration

6: Loopback Device Selection [0,?]           (0) ->
7: Loopback Controller Offset in Multibus [0x0-0xffff] ( ) ->
8: Loopback Controller Interrupt Number [0-7] ( ) ->
9: Test Instructions [y,n]                   (n) ->
10: Enter OK, or :NN to return to question NN [OK] (OK) ->

```

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

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

- :nn — Returns to an earlier prompt (n is the prompt number)
- : — Advances to the next unanswered prompt
- :? — Displays (reviews) all responses up to the current prompt
- ? — Requests help for the current prompt (if available)
- ^ — Returns to the previous prompt

4.3.2 Prompt Explanations

A description of the meaning of each prompt follows:

Device Under Test Selection [0, ?] (0) ->

This prompt selects the device for testing from the system configuration file. The number entered indicates the device to be tested. If **0** is entered, prompts 2-5 are displayed for manually specifying the device test parameters. Any valid value other than **0** will *not* display prompts 2-5 and skips to prompt 6, Loopback Device Selection.

Controller Under Test IOP [3-7] () ->

This prompt begins the queries for user-defined devices; additional prompts are displayed requesting hardware configuration information. Enter the CCU slot number of the IOP for the corresponding controller under test.

Controller Under Test Multibus Chassis [0-3] () ->

Enter the Multibus chassis number that the controller under test resides in.

Controller Under Test Offset in Multibus [0x0-0xffff] () ->

Enter the low-order 12 bits of the controller's address within the Multibus.

Controller Under Test Interrupt Number [0-7] () ->

Enter the interrupt level of the controller within the Multibus.

Loopback Device Selection [0, ?] (0) ->

If Class 3 subtests are to be executed, the specification of the second DR-11W must be entered. This prompt is only displayed when a Class 3 subtest is to be executed. The prompt allows for selection of a known good device (used for loopback testing of the device under test) from the system configuration file. The number entered for the prompt indicates the device to be tested. If **0** is entered for a device number, then prompts 7-8 are displayed. Any valid value other than **0** will *not* display prompts 7-8 and skips to prompt 9, Test Instructions. If **0** is entered, prompts 7-8 allow for manually specifying the loopback device parameters.

Loopback Controller Offset in Multibus [0x0-0xffff] () ->

Enter the low-order 12 bits of the loopback controller's address within the Multibus.

Loopback Controller Interrupt Number [0-7] () ->

Enter the interrupt level of the loopback controller within the Multibus.

Test Instructions [y,n] (n) ->

If **y** is entered, a page of information about the required test cables for loopback and parity testing is displayed.

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

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

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

Figure 4-4, Sample Test Parameter Summary

TEST PARAMETER SUMMARY	
Controller Under Test IOP	: 3
Controller Under Test Multibus Chassis	: 0
Controller Under Test Offset in Multibus	: 0x0
Controller Under Test Interrupt Number	: 0
Loopback Controller Offset in Multibus	: 0x100
Loopback Controller Interrupt Number	: 1
Test Instructions	: n
Enter OK, or :NN to return to question NN	: OK

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

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 *dev4600* test contains the following three classes of subtests as shown in the following table:

Table 4-5, *dev4600* Test Classes

CLASS	DESCRIPTION
1	Loopback tests
2	Parity-loopback test
3	Dual Emulator DMA tests

4.5.1 Class 1 Subtests

Class 1 subtests verify the basic functionality of the controller. The subtests include board reset and loopback tests. Class 1 subtests are listed in the following table:

Table 4-6, Class 1 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
100	Reset Capability	0:01
101	Programmed I/O Loopback	0:14
102	FCNx, STTx Status	0:01
103	Attention Signal/Flag	0:01
104	DMA Preparatory	2:00
105	DMA Abort Via Attention and Reset	0:02
106	Interrupt	0:02
107	DMA Output Loopback	1:17
108	DMA Input Loopback	0:11

4.5.1.1 Subtest 100, Reset Capability

Subtest 100 verifies that the controller resets properly by checking the status register for the correct value. The controller must generate no interrupt.

4.5.1.2 Subtest 101, Programmed I/O Loopback

Subtest 101 assures that programmed I/O works properly by checking the data paths and parity generation/check capability. The parity check is not complete until Subtest 200 is also successfully passed.

4.5.1.3 Subtest 102, FCNx, STTx Status

Subtest 102 checks the user-function codes and status flags.

4.5.1.4 Subtest 103, Attention Signal/Flag

Subtest 103 checks the attention signal status and attention flag status. This subtest also tests the commands for resetting the attention flag and checks for proper function on reset.

4.5.1.5 Subtest 104, DMA Preparatory

Subtest 104 checks for proper board status during the stages of DMA initiation. This subtest also verifies DMA direction.

4.5.1.6 Subtest 105, DMA Abort Via Attention and Reset

Subtest 105 checks that DMA operations can be properly aborted via the attention input signal or board reset.

4.5.1.7 Subtest 106, Interrupt

Subtest 106 tests the IKON capability to generate an interrupt on attention assertion or DMA completion. In addition, this subtest tests the capability for board reset to clear this interrupt.

4.5.1.8 Subtest 107, DMA Output Loopback

Subtest 107 performs DMA output to the output data register. This subtest checks the last value in each DMA buffer by reading the value from the input data register after the DMA block is finished. Only the last word in the block can be checked.

Word patterns are tested by passing buffers ending with $0xnnnn$, where n varies from 0 to 0xf. Each 100-word buffer is filled with random data (except for the last entry), generated using *rand()* with a seed value of 137 plus $0xnnnn$.

In addition, the subtest tests a range of buffer sizes; each buffer contains random data patterns. The random values are generated using *rand()* with a seed value of 137 plus the buffer size. The last value in each buffer, which is described in the following table, is complemented for each buffer size tested.

This subtest tests the following DMA buffer sizes (hex values):

Table 4-7, DMA Buffers Sizes Tested

INITIAL SIZE	SIZE INCREMENT	SIZE LIMIT	ALTERNATING FINAL VALUE
1	1	10	[~]AAAA
11	11	100	[~]CCCC
101	111	1000	[~]EEEE
1001	1111	10000	[~]1111
FFFE	1	10000	[~]0000

4.5.1.9 Subtest 108, DMA Input Loopback

Subtest 108 performs Direct Memory Access (DMA) input of word patterns from the input data register. The subtest primes the input data register by writing to the output data register and through the loopback cable. The same fixed value is transferred into all words of the input buffer since the input data register is not changed while DMA is in progress.

Word patterns are tested by priming the input data register to $0xnnnn$, where n varies from 0 to 0xf. The subtest reads in 100-word blocks.

Both this subtest and Subtest 107 test the same range of DMA buffer sizes; refer to Subtest 107 for a listing of buffer sizes tested.

4.5.2 Class 2 Subtest

The Class 2 subtest verifies correct operation of the parity circuits.

NOTE

A special loopback cable, CONVEX part number 604-100004-001, is required to execute this test.

The Class 2 subtest is listed in the following table:

Table 4-8, Class 2 Subtest

SUBTEST	DESCRIPTION	TIME (min:sec)
200	Data Parity	0:16

4.5.2.1 Subtest 200, Data Parity

Subtest 200, which complements Subtest 101, checks the parity-checking circuitry. This subtest uses data patterns of known parity to verify that the parity checking circuits pass good parity and also detect bad parity.

NOTE

A special loopback cable is required, CONVEX part number 604-100004-001, to execute this subtest. The cable allows input parity to be driven from a known good data bit (checked in Subtest 101).

4.5.3 Class 3 Subtests

Class 3 subtests verify correct operation of the DMA circuits by exchanging data between two emulators. Class 3 subtests are listed in the following table:

Table 4-9, Class 3 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
300	DMA Output	1:36
301	DMA Input	1:37

4.5.3.1 Subtest 300, DMA Output

Subtest 300 uses a second DR-11W Emulator to receive DMA output data from the DR-11W Emulator under test to test a data block containing all possible word patterns. In addition, this subtest tests blocks of varying sizes containing random patterns. Random numbers are generated using a seed value of 137 plus the buffer size.

This subtest tests the following DMA buffer sizes (hex values):

Table 4-10, DMA Buffer Sizes Tested

INITIAL SIZE	SIZE INCREMENT	SIZE LIMIT
1	1	10
11	11	100
101	111	1000
1001	1111	10000
FFFE	1	10000

4.5.3.2 Subtest 301, DMA Input

Subtest 301 uses a second DR-11W Emulator to transmit DMA input data to the DR-11W Emulator under test to test a data block containing all possible word patterns. In addition, this subtest tests blocks of varying sizes containing random patterns, which are generated using *rand()* with a seed value of 137 plus the buffer size. Both this subtest and Subtest 300 test DMA buffers of the same size; refer to Subtest 300 for a listing of buffer sizes tested.

4.6 Error Messages

Datum: [expected *0xnn*] [actual *0xnn*] [offset *0xnn*]

This logs a data byte, including expected value, in case of error. The offset, if given, specifies an offset in a data buffer.

Erroneous interrupt

IKON status: expected *0xnnn*, actual *0xnn*, differences:

{Interpretation}

An interrupt has occurred. Refer to “IKON status” for interpretation detail.

Exception for iop d from recv_iop

{error message detail}

An error has occurred using the SPU/IOP interface to wait for a signal from the IOP that the command is finished. Refer to “Exception from setup_iop” for message detail.

Exception from load_iop

{error message detail}

An error has occurred while bootstrapping the IOP. Make sure that the IOP program image file *dev4600.x00* is available and readable. It must reside either in the current directory or */mnt/test*. Refer to “Exception from setup_iop” for message detail.

Exception from send_iop (n)

{error message detail}

An error has occurred using the SPU/IOP interface to signal the IOP that a command is ready. Refer to “Exception from setup_iop” for message detail.

Exception from setup_iop

{error message detail}

An error occurred when preparing to access the IOP. When this error occurs, the message detail may be the following:

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

Exception from start_iop

{error message detail}

An error has occurred while starting the IOP. Refer to “Exception from setup_iop” for message detail.

Exception in `mmalloc_init`. 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, consult the Technical Assistance Center.

Function status: message

{UNIX error message if applicable}

There is a problem in IOP processing. The following messages may be displayed:

Table 4-12, IOP Processing Error Messages

TEXT	MEANING
UNIX ERROR	Error specified in message
SUBSYSTEM WON'T GO READY	Hardware problem

IKON status `0xnn` means:

{interpretation}

This is the IKON status word. The following interpretation may be displayed:

Figure 4-5, Interpreted Status Word Error Messages

DMAF: DMA END-RANGE FLAG [NOT] SET
ATTF: ATTENTION FLAG [NOT] SET
ATTN: ATTENTION INPUT [NOT] SET
PERR: PARITY FLAG [NOT] SET
STTA: STATUS A INPUT [NOT] SET
STTB: STATUS B INPUT [NOT] SET
STTC: STATUS C INPUT [NOT] SET
REDY: IKON [NOT] READY
IENB: IKON INTERRUPTS [NOT] ENABLED
FCN3: FCN3 LATCH [NOT] SET
FCN2: FCN2 LATCH [NOT] SET
FCN1: FCN1 LATCH [NOT] SET

IKON status: expected `0xnn`, actual `0xnn`, differences:

{interpretation}

An unexpected status code was obtained. The differences between the actual code and what was desired are interpreted (refer to previous error message).

Main memory allocation error

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

Missing interrupt

IKON expected interrupt code `0xnn`

{Interpretation}

An interrupt, which should have occurred, did not occur. Refer to "IKON status" for interpretation detail.

Spurious interrupt error: *dd* expected, *dd* actual

The count of spurious interrupts was not as expected. Spurious interrupts are those which are redundant.

Unexpected interrupt

IKON actual interrupt code 0xnn

{interpretation}

An interrupt that was not expected occurred. Refer to "IKON status" for interpretation detail.

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 than a 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 *csb*) 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 *cs*.

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, <i>contact</i> 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 <i>contact</i> utility and returns you to the shell environment. |
| Abort | Save the text of your report in a file named <i>dead.report</i> in your home directory. This option exits the <i>contact</i> 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
Attention signal/flag 4-9

C

C Programming Language ix
Canada, reporting problems from, telephone number for x
cattypedevmn.suffix 1-1
Cautions, described ix
Class 1, loopback tests 4-8
Class 2, parity-loopback test 4-10
Class descriptions 4-8
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
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

Data parity 4-10
dead.report file, submitting A-3
dead.report file, using *-r* option to submit A-3
dev, test category 1-2
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
DMA abort 4-9
DMA abort via attention and reset 4-9
DMA circuits 4-11
DMA initiation 4-9
DMA input 4-11
DMA input loopback 4-10
DMA output 4-11
DMA output loopback 4-9
DMA preparatory 4-9
DMA test buffer sizes 4-9
dshell, introduction 3-1
dshell, overview 3-1
Dual Emulator DMA 4-11

E

Emulator model 10077 test 4-1
Error messages 4-12
Error messages, selecting 3-1
error reporting A-1

F

FCNx 4-8
FCNx, STTx status 4-8
Files, test outputs to 3-1

H

Hawaii, reporting problems from, telephone number for x
Help-for, *dev/600* prompts 4-4

I

IKON controller test, class descriptions 4-8
IKON controller test, user interface 4-2
Interrupt 4-9
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
Multibus emulator controller 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
Overview, diagnostic environment 1-1
Overview, *dshell* 3-1

P

Parity circuits 4-10
Peripheral devices, test program name for 1-1
Peripherals, *dev*, test program for 1-2
Printers 1-2
Printers, device, test program for 1-3
problems, reporting, overview A-1
Programmed I/O loopback 4-8

Index

R

Reader's Forum x
Reporting problems x
Reset capability 4-8
Revision sheet 3

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
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
STTx 4-8
Subsystems, *cat* for 1-1
Subtest descriptions 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 query 4-4
Test program invocation, *dev4600* 4-2
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
Trouble reports x
trouble reports A-1

U

UNIX-to-UNIX Communication Protocol A-1
UNIX-to-UNIX copy command, *uucp* A-1
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
Word pattern 4-9

14869

CONVEX Multibus Emulator Controller
(dev4600) Diagnostics Manual
Document No. 760-002830-000
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)



CONVEX

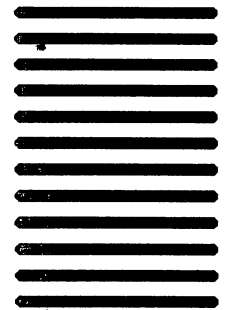


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

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)