# CONVEX VMEbus UltraNet Controller (*dev_ultra*) Diagnostics Manual

Document No. 760-003430-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

*CONVEX VMEbus UltraNet Controller (dev_ultra) Diagnostics Manual*
Order No. DHW-246
First Edition



Printed in the United States of America

# Revision Sheet

*CONVEX VMEbus UltraNet Controller*
*(dev_ultra) Diagnostics Manual*

| Edition | Document No. | Date | Description |
|---|---|---|---|
| First | 760-003430-000 | May 1991 | First release. Contains the *dev_ultra* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.* |

THIS PAGE INTENTIONALLY LEFT BLANK

# Table of Contents



# Appendixes

# List of Tables



# List of Figures

# Preface

## Purpose and Intended Audience

This manual explains how to run the *dev_ultra* diagnostic, which checks the CONVEX VMEbus UltraNet controller. This document is not a tutorial, but rather a reference for the users of the *dev_ultra* 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 *dev_ultra* diagnostic.

## Scope

This manual applies to all CONVEX computers.

## Organization

This document consists of the following:

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

# Notational Conventions

The notational conventions used in this text are listed below:

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

# Warnings

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

**WARNING**

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

**CAUTION**

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

**NOTE**

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

# Associated Documents

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

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

# Ordering Documentation

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

The product number for this manual is DHW-246.
The document number for this manual is 760-003430-000.

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

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

# Technical Assistance

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

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

# Reader's Forum

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

# Chapter 1
# Diagnostics Environment

## 1.1 Overview

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

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

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

## 1.2 Test Program Naming Conventions

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

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

### 1.2.1 Test Program Categories

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

**Table 1–1, Test Program Categories**

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

### 1.2.2 Test Program Types

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

**Table 1–2, Test Program Types**

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

### 1.2.3 Test Program Device Types

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

**Table 1–3, Test Program Device Types**

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

### 1.2.4 Examples of Test Program Names

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

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

**Table 1–4, Example Test Program Names**

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

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 2
# EGOS Overview

## 2.1 Overview

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

## 2.2 Purpose of EGOS for Diagnostic Testing

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

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

## 2.3 EGOS for the Multibus Interface

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

## 2.4 EGOS for HSP Interface, HSP EGOS

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

## 2.5 EGOS for VME Interface, VIOP EGOS

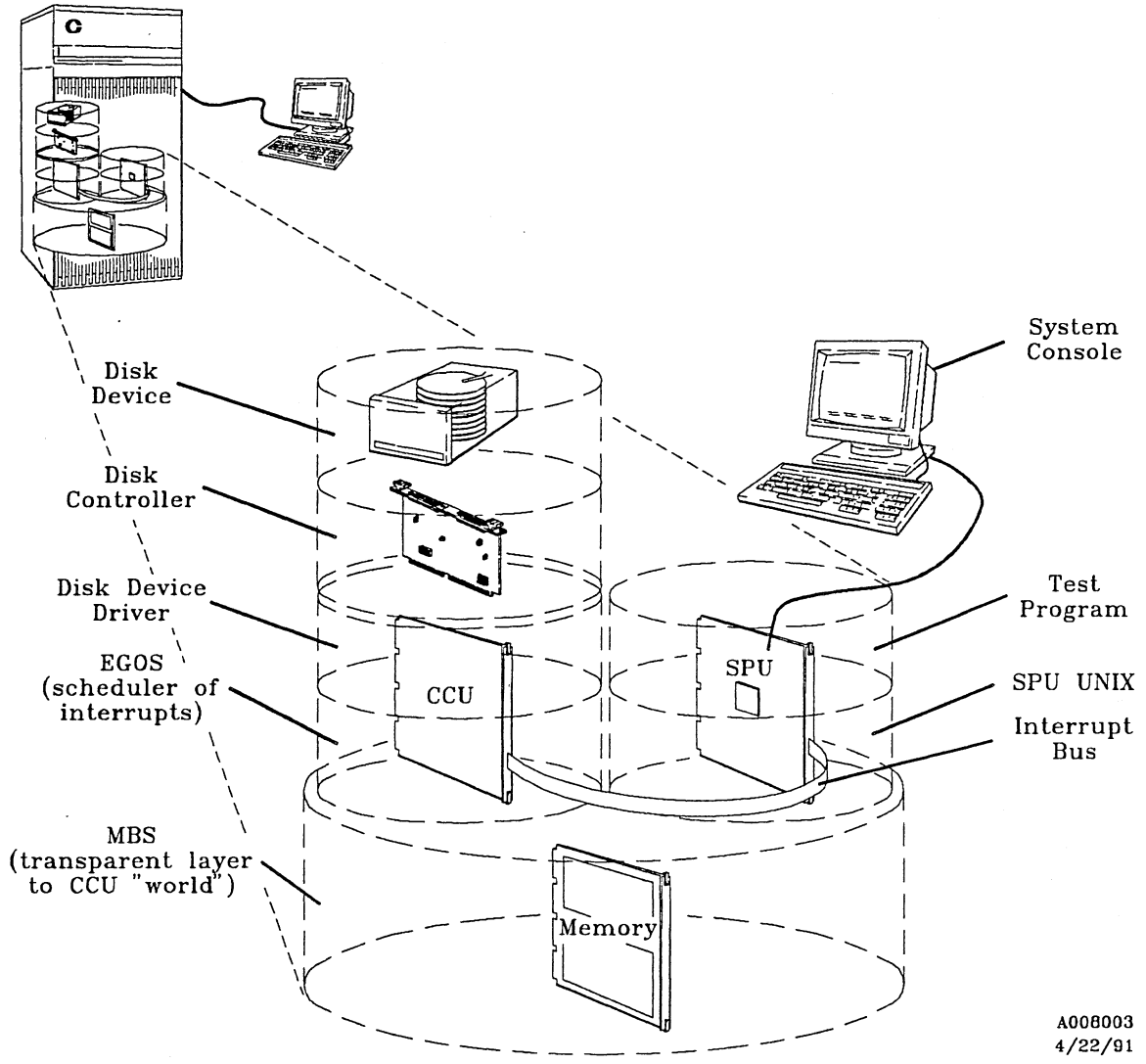
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes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

## 2.6 EGOS Position in the Environment

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

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

**Figure 2–1, EGOS' Position in the Environment**



THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 3
# Dshell Overview

## 3.1 Overview

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

## 3.2 Diagnostic Shell (*dshell*) Overview

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

The *dshell* has two basic functions:

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

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

## Table 3–1, *dshell* Commands

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

## 3.3 Syntax Help for *dshell* Commands

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

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

```
: loop
Proper syntax is:

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

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 4
# VMEbus UltraNet Controller Test (*dev_ultra*)

## 4.1 Overview

The *dev_ultra* test is a functional test for the CONVEX VMEbus UltraNet controller. This test verifies the proper operation of the CONVEX VMEbus UltraNet controller in the CONVEX VMEbus I/O environment.

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

- Verify the functional ability of the VMEbus UltraNet controller to operate in the CONVEX VMEbus I/O environment. This includes VMEbus access and interrupt generation.
- Verify that the VMEbus UltraNet controller can transmit and receive data via internal loopback paths.
- Verify that the VMEbus UltraNet controller can transmit and receive data via an external transceiver and loopback cable.

The *dev_ultra* test uses the same VMEbus I/O Processor (VIOP) code as the ConvexOS operating system. All communications with the VIOP use the same Event-Governed Operating System (EGOS) and the Message-Based System (MBS) used by the ConvexOS operating system. This means that *dev_ultra* tests the communication paths used in a normal operating environment.

## 4.2 Prerequisites and Required Equipment

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

**Table 4–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| VIOP | VIOP |
| VBCU (Rev-D or later) | PIA |
| VMEbus UltraNet controller | VBCU (Rev-D or later) |
| | VMEbus UltraNet controller |

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

Subtest 201 requires a loopback cable to execute. However, if two machines are directly connected via VMEbus UltraNet controllers, Subtest 201 can be executed.

> **NOTE**
>
> Refer to the description of Subtest 201 for the procedure required to execute Subtest 201 in the direct connection mode without loopback cables.

## 4.3 Test Invocation

The *dev_ultra* 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 *dev_ultra* 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 *dev_ultra* or 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 dev_ultra [-c [class number(s)]] [-s [subtest number(s)]] [-option] [-f FILE] [+>filename] RETURN
```

In actual usage, all 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 dev_ultra** executes all applicable *dev_ultra* 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, the **-d, -e, -i, -m,-u, -U,** 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 **dev_ultrax** 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

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

-e *act* Set VMEbus UltraNet controller error action to *act*

-F *FILE* Use alternate CCU driver file (*FILE*), default is */mnt/os/viop*, if it exists; if not, default is */mnt/test/dev_ultra.x00*

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

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

-m *mode* Set error mode to *mode*

-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 startup (use previous parameters from last test invocation — same as **dev_ultrax**)

-t Execute all applicable subtests sequentially (override parameter file subtest list)

-u Same as answering **y** to the `Use Controller even if Already Online` prompt

-U Same as answering **n** to the `Use Controller even if Already Online` prompt

-v Set verbose flag (detailed information printed)

The [+>*filename*] option allows the test results to be written to *filename* as well as displayed.

**NOTE**

The following alternate test invocation procedure is optimal when invoking *dev_ultra* 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 dev_ultrax [-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 **dev_ultra**. When invoking *dev_ultra* 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.

### 4.3.1 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 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

```
                    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) viop  3     0    LAN-202  0x7740  3
 2) viop  3     1    LAN-202  0x7740  3

         *** Enter 0 for manual configuration ***

 2: Ioconfig File Unit to Test [1-2,0,?]                  (1) -> 0
 3: VMEbus IOP Number [3-7,?]                             (3) -> (RETURN)
 4: VMEbus Chassis Number [0-1,?]                         (0) -> (RETURN)
 5: VMEbus CSR of the Controller [0x0-0xffe0,?]      (0x7740) -> (RETURN)
 6: VMEbus Interrupt Level of the Controller [1-7,?]      (3) -> (RETURN)
 7: Use Defaults for Remaining Parameters [y,n,?]          (y) -> n
       ** External Connections (bit mapped) **
    0x0001: Enable External Loopback
    0x0002: Direct Connection to CONVEX Machine

 8: Enable External Loopback [0x0-0x3,?]                (0x0) -> (RETURN)
 9: Use Controller even if Already Online [y,n,?]         (n) -> (RETURN)
10: Number of Iterations for Each Subtest [1-2147483647,?]
                                                          (1) -> (RETURN)
11: Use Defaults for Subqueue Assignments [y,n,?]         (y) -> n
12: SQ_HW_CTL Queue Number [0-7,?]                        (0) -> (RETURN)
13: SQ_CONTROL Queue Number [0-7,?]                       (1) -> (RETURN)
14: SQ_URGENT Queue Number [0-7,?]                        (2) -> (RETURN)
15: SQ_READ Queue Number (not used) [0-7,?]               (3) -> (RETURN)
16: SQ_WRITE Queue Number (not used) [0-7,?]              (4) -> (RETURN)
17: SQ_DEVICE Queue Number (not used) [0-7,?]             (5) -> (RETURN)
18: SQ_DIAG Queue Number [0-7,?]                          (6) -> (RETURN)
19: SQ_DEBUG Queue Number (not used) [0-7,?]              (7) -> (RETURN)
```

## Figure 4–3, Test Parameter Menu (continued)

```
        ** Debug Mode Flags (bit mapped) **
    0x0001: Print messages sent to Ultra
    0x0002: Print messages received from Ultra
    0x0004: Pause before message sent to Ultra
    0x0008: Pause after message received from Ultra
    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 Ultra after subtest is aborted
    0x0400: Don't disable SPU interrupts when done
    0x0800: Simulate all MBS messages (don't send them)

20: 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

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

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

22: Select Error Start/Finish Message Mode [0x0-0x3,?]  (0x0) -> 3
23: Error Start Character Sequence [<error start string>,?]
                                                  (\033[?5i) -> RETURN
24: Error Finish Character Sequence [<error finish string>,?]
                                                  (\033[?4i) -> RETURN
25: 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 4–2, 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 that 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
VMEbus IOP Number                                       : 3
VMEbus Chassis Number                                   : 0
VMEbus CSR of the Controller                            : 0x7740
VMEbus Interrupt Level of the Controller                : 3
Use Defaults for Remaining Parameters                   : n
Enable External Loopback                                : 0x0000
Use Controller even if Already Online                   : n
Number of Iterations for Each Subtest                   : 1
Use Defaults for Subqueue Assignments                   : n
SQ_HW_CTL Queue Number                                  : 0
SQ_CONTROL Queue Number                                 : 1
SQ_URGENT Queue Number                                  : 2
SQ_READ Queue Number (not used)                         : 3
SQ_WRITE Queue Number (not used)                        : 4
SQ_DEVICE Queue Number (not used)                       : 5
SQ_DIAG Queue Number                                    : 6
SQ_DEBUG Queue Number (not used)                        : 7
Select Debug Mode                                       : 0x0000
Select 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 Initialization Sequence for *dev_ultra*

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

- The diagnostic determines if the test was invoked for quick startup (i.e., **dev_ultrax** or **dev_ultra -q**). If so, the diagnostic reads the test parameters from the specified parameter file (default parameter file is */tmp/dev_ultra.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 *dev_ultra* CCU driver. If the driver is not loaded, the CCU is reloaded. 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 non-existent, it can be created with the command: **scn_util -b > /mnt/boot_db** from the (spu)> prompt.

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

### 4.4.1 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) viop  3      0     LAN-202  0x7740  3
2) viop  3      1     LAN-202  0x7740  3

*** Enter 0 for manual configuration ***

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

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

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

Enter the CCU slot number for the VMEbus IOP that controls the VMEbus chassis containing the UltraNet controller.

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

Enter the VMEbus chassis number for the chassis where the UltraNet controller is installed. If there is only 1 chassis, the number is usually 0.

```
5: VMEbus CSR of the Controller [0x0-0xffe0,?]      (0x7740) ->
```

Enter the VMEbus address of UltraNet Controller Status Register. The standard address for the UltraNet controller is 0x7740.

```
6: VMEbus Interrupt Level of the Controller [1-7,?]        (3) ->
```

Enter the CONVEX VMEbus interrupt number assigned to the UltraNet controller board under test. The interrupt number must be between 1 and 7, inclusive. The standard interrupt number for UltraNet is 3.

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

Enter **y** to select the default values for all remaining prompts. This bypasses entering a RETURN in response to every remaining prompt.

```
    ** External Connections (bit mapped) **
0x0001: Enable External Loopback
0x0002: Direct Connection to CONVEX Machine

8: Enable External Loopback [0x0-0x3,?]                    (0x0) ->
```

Select what external connections, if any, are available for the test. If a loopback jumper is attached to the transceiver, the external data path through the transceiver is tested. If no loopback jumper is present, the UltraNet controller is tested only with an internal loopback path and the transceiver and cables are not tested. Option 0x0002 is ignored at this time.

```
9: Use Controller even if Already Online [y,n,?]           (n) ->
```

Enter **y** to override the exclusive open flag on the VIOP driver. The exclusive open flag is clear if the VIOP was loaded by this test. This option is used for debugging test/driver problems.

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

Enter the number of times each subtest is to be executed on the UltraNet controller. This is most useful for repeating a subtest many times in quick succession. This may also be set using the **-i** command line option during test invocation.

```
11: Use Defaults for Subqueue Assignments [y,n,?]          (y) ->
```

Enter **y** to select the default values for subqueue assignments. Subqueue assignments can be changed to allow this test to work with non-standard VIOP code (i.e., all control queues are mapped into the hardware control queue). The default subqueue assignments are:

| | | |
|---|---|---|
| SQ_HW_CTL | 0 | # Hardware control functions |
| SQ_CONTROL | 1 | # Software control functions |
| SQ_URGENT | 2 | # Software control functions (priority) |
| SQ_READ | 3 | # (Not used by the dev_ultra diagnostic) |
| SQ_WRITE | 4 | # (Not used by the dev_ultra diagnostic) |
| SQ_DEVICE | 5 | # (Not used by the dev_ultra diagnostic) |
| SQ_DIAG | 6 | # Diagnostic tests |
| SQ_DEBUG | 7 | # (Not used by the dev_ultra diagnostic) |

```
12: SQ_HW_CTL Queue Number [0-7,?]                    (0) ->
```

Refer to the explanation of the **Use Defaults for Subqueue Assignments** prompt (number 11 in this test description) for information on default values for subqueue assignments.

```
13: SQ_CONTROL Queue Number [0-7,?]                   (1) ->
```

Refer to the explanation of the **Use Defaults for Subqueue Assignments** prompt (number 11 in this test description) for information on default values for subqueue assignments.

```
14: SQ_URGENT Queue Number [0-7,?]                    (2) ->
```

Refer to the explanation of the **Use Defaults for Subqueue Assignments** prompt (number 11 in this test description) for information on default values for subqueue assignments.

```
15: SQ_READ Queue Number (not used) [0-7,?]           (3) ->
```

Refer to the explanation of the **Use Defaults for Subqueue Assignments** prompt (number 11 in this test description) for information on default values for subqueue assignments.

```
16: SQ_WRITE Queue Number (not used) [0-7,?]          (4) ->
```

Refer to the explanation of the **Use Defaults for Subqueue Assignments** prompt (number 11 in this test description) for information on default values for subqueue assignments.

```
17: SQ_DEVICE Queue Number (not used) [0-7,?]         (5) ->
```

Refer to the explanation of the **Use Defaults for Subqueue Assignments** prompt (number 11 in this test description) for information on default values for subqueue assignments.

```
18: SQ_DIAG Queue Number [0-7,?]                      (6) ->
```

Refer to the explanation of the **Use Defaults for Subqueue Assignments** prompt (number 11 in this test description) for information on default values for subqueue assignments.

```
19: SQ_DEBUG Queue Number (not used) [0-7,?]          (7) ->
```

Refer to the explanation of the **Use Defaults for Subqueue Assignments** prompt (number 11 in this test description) for information on default values for subqueue assignments.

```
    ** Debug Mode Flags (bit mapped) **
0x0001: Print messages sent to Ultra
0x0002: Print messages received from Ultra
0x0004: Pause before message sent to Ultra
0x0008: Pause after message received from Ultra
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 Ultra after subtest is aborted
0x0400: Don't disable SPU interrupts when done
0x0800: Simulate all MBS messages (don't send them)
```

```
20: 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 VMEbus UltraNet 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** option 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 printer interrupt are not disabled when the diagnostic is completed. If flag 0x0800 is set, a simulation of the test is done without affecting the CCU or the controller.

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

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

Enter the desired subtest error action.

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

```
22: 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 the diagnostic multiple times or for long periods of time. This assumes a printer is connected to the auxiliary port on the display terminal where the diagnostic 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 |

```
23: 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 |

```
24: 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 |

```
25: 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.

## 4.5 Class Descriptions

The *dev_ultra* test contains the three classes of subtests as listed in the following table:

**Table 4–3, *dev_ultra* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | VMEbus UltraNet Controller Test |
| 2 | VMEbus UltraNet Controller Extended Tests |
| 3 | VMEbus UltraNet Controller Addressing Test |

All subtests contained in *dev_ultra* may be looped under the *dshell*.

## 4.6 Class 1 Subtest

Class 1 contains only one subtest. The following table lists the Class 1 subtest, its description, and the time required to execute the subtest:

**Table 4–4, Class 1 Subtest**

| Subtest | Description | Time (min:sec) |
|---|---|---|
| 100 | VMEbus UltraNet Controller Reset Test | 00:01 |

### 4.6.1 Subtest 100, VMEbus UltraNet Controller Reset Test

Subtest 100 verifies access to the VMEbus UltraNet controller and reports the results of the controller's most recent power-on reset test. The subtest verifies the following:

- The ability to communicate with the VMEbus Input/Output Processor (VIOP), VMEbus Control Unit (VBCU), and the VMEbus UltraNet controller.
- The ability of the VMEbus UltraNet controller to interrupt the VIOP.

The information from the power-on test varies with the version of the VMEbus UltraNet controller EPROM code. If the EPROM code version is less than five, the only information available is the EPROM version and the results of the power-on test. For EPROM code versions greater than or equal to five, the hardware model, version, revision, options, and serial number are shown.

## 4.7 Class 2 Subtests

Class 2 contains two subtests. The following table lists the Class 2 subtests, their descriptions, and the times required to execute each subtest:

**Table 4–5, Class 2 Subtests**

| Subtest | Description | Time (min:sec) |
|---|---|---|
| 200 | EPROM-based Test (Internal Loopback) | 00:04 |
| 201 | EPROM-based Test (External Loopback) | 00:04 |

### 4.7.1 Subtest 200, VMEbus UltraNet Controller EPROM-based Test (Internal Loopback)

Subtest 200 tests all major functional units of the VMEbus UltraNet controller. The testing sequence is:

- Check EPROM checksum
- Test controller RAM—the RAM test consists of pattern-testing the RAM with 8-bit, 16-bit, and 32-bit accesses
- Check Programmable Interrupt Controller (PIC) and Programmable Interval Timer (PIT)
- Verify that internal loopback is functioning
- Check FIFO RAM with 80386 processor accesses
- Test Checksum Gate Arrays (used to compute checksums on data packets)
- Test controller interrupt processing
- Test VMEbus memory access—the VMEbus RAM is tested with a pattern test
- Test FIFO RAM with DMA accesses

### 4.7.2 Subtest 201, VMEbus UltraNet Controller EPROM-based Test (External Loopback)

Subtest 201 tests all major functional units of the VMEbus UltraNet controller. The FIFO tests use an external loopback path that goes through the external transceiver and loopback cable. If the loopback cable is not installed, this test is not executed. However, if two CONVEX machines are directly connected via VMEbus UltraNet controllers, the subtest can be fully exercised by following these steps:

1. Execute the *dev_ultra* diagnostic from both machines (does not have to be concurrently) with the **-s 200** option.
2. Execute the *dev_ultra* diagnostic from one machine with the **-s 201** option and the **-o 3** option.

The Subtest 201 testing sequence is:

- Check EPROM checksum
- Test controller RAM—the RAM test consists of pattern-testing the RAM with 8-bit, 16-bit, and 32-bit accesses
- Check Programmable Interrupt Controller (PIC) and Programmable Interval Timer (PIT)
- Verify that external loopback is functioning
- Check FIFO RAM with 80386 processor accesses
- Test Checksum Gate Arrays (used to compute checksums on data packets)
- Test controller interrupt processing
- Test VMEbus memory access—the VMEbus RAM is tested with a pattern test
- Test FIFO RAM with DMA accesses

## 4.8 Class 3 Subtests

Class 3 contains only one subtest. The following table lists the Class 3 subtest, its description, and the time required to execute the subtest:

**Table 4–6, Class 3 Subtest**

| Subtest | Description | Time (min:sec) |
|---|---|---|
| 300 | VMEbus UltraNet Controller Addressing Test | 00:01 |

### 4.8.1 Subtest 300, VMEbus UltraNet Controller Addressing Test

Subtest 300 tests the ability of the VMEbus UltraNet controller to address all available addresses in VMEbus address space. The controller reads data from each page of VMEbus memory and writes a reply back to the address. This subtest must have complete control of the VIOP, as it temporarily disables MBS and EGOS interrupts while changing the VMEbus cache controller page maps.

## 4.9 Error Messages

There are two 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

### 4.9.1 Start-up Error Messages

The first type of error messages are errors reported during the start-up sequence of the diagnostic. These messages do not contain header and footer information. They consist of a start-up error message and, when long messages are enabled from *dshell*, a secondary subtest error message may also occur. When the verbose bit is set and long messages are enabled from *dshell*, a source file information message is also provided.

The following start-up error messages may occur:

```
CMI command returned fail status (0x17)

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

File not found (0x0A)

Invalid command (0x10)

MBS error occurred (0x09)

Subtest aborted by operator (0x21)

Subtest failed (0x20)

Timeout waiting for CCU driver reply (0x08)

Timeout waiting for Ultra reply (0x12)

Ultra is owned by another process (0x13)

Ultra not initialized (0x11)

Unable to allocate CCU windows for message (0x14)

Unable to allocate main memory (0x16)

Unable to configure LAN-202 driver (0x05)

Unable to initialize CCU (0x04)

Unable to initialize processor queues (0x01)

Unable to probe LAN-202 driver (0x06)
```

### 4.9.2 Subtest Error Messages

The second type of error messages are errors reported by an executing subtest (i.e., subtest failures). These messages contain a standard header and footer, subtest error message, and the following information:.

- Iteration—lists which iteration of the subtest was being executed at the time of the failure
- Reason(s)—describes the error condition and usually contains actual and expected values

The following figure contains an example of the standard header message:

**Figure 4–5, Standard Header for Error Messages**

```
*****  Thu Nov  9 15:46:48 CST 1989  *****
Test:    dev_ultra.t 1.2   Class: 1    Subtest: 100 1.1   Count: 1    Error: 0
Failed:  VMEbus UltraNet Controller Reset 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 'dev_ultra.t' failed
Elapsed time:   0:00:03
```

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

```
Reason(s) : Unable to initialize processor queues

Reason(s) : Unable to initialize CCU

Reason(s) : Unable to configure LAN-202 driver

Reason(s) : Unable to probe LAN-202 driver

Reason(s) : File is not in b.out format

Reason(s) : Timeout waiting for CCU driver reply

Reason(s) : MBS error occurred
```

```
Reason(s): File not found

Reason(s): Invalid command

Reason(s): Ultra not initialized

Reason(s): Timeout waiting for Ultra reply

Reason(s): Ultra is owned by another process

Reason(s): Unable to allocate CCU windows for message

Reason(s): Unable to allocate main memory

Reason(s): CMI command returned fail status

Reason(s): Subtest failed

Reason(s): Subtest aborted by operator

Reason(s): Unknown cmi status: 0xXX

Reason(s): Non-specific error status

Reason(s): Device driver not found

Reason(s): Device not found

Reason(s): Unit not found

Reason(s): Unit already connected

Reason(s): Unit disconnected

Reason(s): Unit already online

Reason(s): Unit offline

Reason(s): Unit already mounted

Reason(s): Unit not mounted

Reason(s): No memory for initialization

Reason(s): Out of CCU windows

Reason(s): Function timeout exceeded

Reason(s): Function aborted by CCU

Reason(s): Invalid CMI function code

Reason(s): Dev_name value is unsupported

Reason(s): Unit_name value is unsupported

Reason(s): Dev_class value is unsupported
```

```
Reason(s): Dev_type value is unsupported
Reason(s): CMI revision unsupported by CCU
Reason(s): Defective CMI message (bad parameters)
Reason(s): CMI message is required (not MBS)
Reason(s): Controller already initialized
Reason(s): Controller not initialized
Reason(s): Unable to initialize device queues
Reason(s): Hardware failure on controller
Reason(s): Bad sequence number on firmware load record
Reason(s): Option only valid on first record
Reason(s): Option only valid on last record
Reason(s): Unable to start firmware
Reason(s): Unable to load firmware record
Reason(s): Memory allocation failed
Reason(s): Attempt to write on write protected unit
Reason(s): Retries were required to complete command
Reason(s): Unit is not ready
Reason(s): Could not seek to required location
Reason(s): Unrecoverable error even with ECC
Reason(s): Bad header (checksum failed)
Reason(s): Operation would run beyond last block
Reason(s): Function canceled by CPU
Reason(s): Device address not valid
Reason(s): Bad parity or non-existent memory
Reason(s): Unrecoverable memory error
Reason(s): CCU-detectable physical I/O error
Reason(s): Function invalid for subqueue
Reason(s): Read/write data request too large
Reason(s): Requested data not available
```

```
Reason(s): Dev_type does not match installed hardware

Reason(s): MBS error detected by CCU

Reason(s): No MBS message

Reason(s): Too many drivers in /ioconfig file

Reason(s): CCU driver didn't return a status code

Reason(s): Diagnostic test failed
           Error(s) reported by UltraNet self-test:
           No errors returned by UltraNet self-test
           EPROM checksum incorrect
           RAM check failed
           Interrupt Controller or Interval Timer failed
           Loopback connector check failed
           FIFO memory test failed
           Checksum PAL test failed
           Non-Maskable Interrupt control test failed
           VMEbus Memory test failed
           FIFO memory/DMA test failed
           FIFO control test failed
           Address check test failed:
           Expected address: XXXXXX
           Actual address:   YYYYYY
           Address check test passed
           Unknown command (0xXXXX)

Reason(s): No errors returned by UltraNet self-test

Reason(s): Error(s) reported by UltraNet self-test:
           EPROM checksum incorrect
           RAM check failed
           Interrupt Controller or Interval Timer failed
           Loopback connector check failed
           FIFO memory test failed
           Checksum PAL test failed
           Non-Maskable Interrupt control test failed
           VMEbus Memory test failed
           FIFO memory/DMA test failed
           FIFO control test failed

Reason(s): Address check test failed:
           Expected address: XXXXXX
           Actual address:   YYYYYY

Reason(s): Address check test passed

Reason(s): Unknown command (0xXXXX)
```

THIS PAGE INTENTIONALLY LEFT BLANK

# Appendix A
# Reporting Problems

## A.1 Overview

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

## A.2 Technical Assistance Center

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

## A.3 The *contact* Utility

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

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

## A.4 Prerequisites

To use *contact* requires

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

### A.4.1 UUCP Connection

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

### A.4.2 Finding the Program Path Name

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

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

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

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

### A.4.3 Finding the Program Version Number

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

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

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

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

## A.5 Tips on Using the *contact* Utility

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

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

### A.5.1 Using a *.contact* File

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

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

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

### A.5.2 Aborting the Report

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

### A.5.3 Submitting the *dead.report* File

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

**contact -r**

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

### A.5.4 Suspending a Report

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

### A.5.5 Ending a Response

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

### A.5.6 Tilde-Escape Sequences

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

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

## A.6 Using the *contact* Utility

The *contact* utility prompts for the following information:

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

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

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

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

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

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

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

Enter the name of the product involved
>
```

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

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

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

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

**NOTE**

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

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

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

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

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

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

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

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

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

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

# Index

## S



## T



## U



## V



## W

**CONVEX VMEbus UltraNet Controller**
**(*dev_ultra*) Diagnostics Manual**
Document No. 760-003430-000
First Edition

## Reader's Forum

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

**From:**

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

**FOR ADDITIONAL INFORMATION OR DOCUMENTATION:**

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

Direct mail orders to:

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

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

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

(Fold Here Second)

(Tape or Staple)

**CONVEX VMEbus UltraNet Controller**
**(*dev_ultra*) Diagnostics Manual**
Document No. 760-003430-000
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)